

28th EACSL Annual Conference on Computer Science Logic

CSL 2020, January 13–16, 2020, Barcelona, Spain

Edited by

Maribel Fernández
Anca Muscholl



Editors

Maribel Fernández

King's College London, UK
maribel.fernandez@kcl.ac.uk

Anca Muscholl

University of Bordeaux, France
anca@labri.fr

ACM Classification 2012

Theory of computation → Logic

ISBN 978-3-95977-132-0

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/978-3-95977-132-0>.

Publication date

January, 2020

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>.

License

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



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

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

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.CSL.2020.0

ISBN 978-3-95977-132-0

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.

Editorial Board

- Luca Aceto (*Chair*, Gran Sasso Science Institute and Reykjavik University)
- Christel Baier (TU Dresden)
- Mikolaj Bojanczyk (University of Warsaw)
- Roberto Di Cosmo (INRIA and University Paris Diderot)
- Javier Esparza (TU München)
- Meena Mahajan (Institute of Mathematical Sciences)
- Dieter van Melkebeek (University of Wisconsin-Madison)
- Anca Muscholl (University Bordeaux)
- Luke Ong (University of Oxford)
- Catuscia Palamidessi (INRIA)
- Thomas Schwentick (TU Dortmund)
- Raimund Seidel (Saarland University and Schloss Dagstuhl – Leibniz-Zentrum für Informatik)

ISSN 1868-8969

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

■ Contents

Preface	
<i>Maribel Fernández and Anca Muscholl</i>	0:ix
The Ackermann Award 2019	
<i>Michael Benedikt and Thomas Schwentick</i>	0:xv–0:xviii

Invited Talks

Verification of Security Protocols	
<i>Véronique Cortier</i>	1:1–1:2
Symmetric Computation	
<i>Anuj Dawar</i>	2:1–2:12
Solving Word Equations (And Other Unification Problems) by Recompression	
<i>Artur Jeż</i>	3:1–3:17
Strong Bisimulation for Control Operators	
<i>Delia Kesner, Eduardo Bonelli, and Andrés Viso</i>	4:1–4:23
From Classical Proof Theory to P versus NP: a Guide to Bounded Theories	
<i>Iddo Tzameret</i>	5:1–5:2

Regular Papers

Generalized Connectives for Multiplicative Linear Logic	
<i>Matteo Acclavio and Roberto Maieli</i>	6:1–6:16
On Free Completely Iterative Algebras	
<i>Jiří Adámek</i>	7:1–7:21
Strongly Unambiguous Büchi Automata Are Polynomially Predictable With Membership Queries	
<i>Dana Angluin, Timos Antonopoulos, and Dana Fisman</i>	8:1–8:17
A Robust Class of Linear Recurrence Sequences	
<i>Corentin Barloy, Nathanaël Fijalkow, Nathan Lhote, and Filip Mazowiecki</i>	9:1–9:16
Coverage and Vacuity in Network Formation Games	
<i>Gili Bielous and Orna Kupferman</i>	10:1–10:18
A Complete Axiomatisation of a Fragment of Language Algebra	
<i>Paul Brunet</i>	11:1–11:15
Proof Complexity of Systems of (Non-Deterministic) Decision Trees and Branching Programs	
<i>Sam Buss, Anupam Das, and Alexander Knop</i>	12:1–12:17
Internal Parametricity for Cubical Type Theory	
<i>Evan Cavallo and Robert Harper</i>	13:1–13:17

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl



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

Unifying Cubical Models of Univalent Type Theory <i>Evan Cavallo, Anders Mörtberg, and Andrew W Swan</i>	14:1–14:17
FO-Definability of Shrub-Depth <i>Yijia Chen and Jörg Flum</i>	15:1–15:16
Taylor expansion for Call-By-Push-Value <i>Jules Chouquet and Christine Tasson</i>	16:1–16:16
Tangent Categories from the Coalgebras of Differential Categories <i>Robin Cockett, Jean-Simon Pacaud Lemay, and Rory B. B. Lucyshyn-Wright</i>	17:1–17:17
Reverse Derivative Categories <i>Robin Cockett, Geoffrey Cruttwell, Jonathan Gallagher, Jean-Simon Pacaud Lemay, Benjamin MacAdam, Gordon Plotkin, and Dorette Pronk</i>	18:1–18:16
Internal Calculi for Separation Logics <i>Stéphane Demri, Etienne Lozes, and Alessio Mansutti</i>	19:1–19:18
Monitoring Event Frequencies <i>Thomas Ferrère, Thomas A. Henzinger, and Bernhard Kragl</i>	20:1–20:16
Automatic Equivalence Structures of Polynomial Growth <i>Moses Ganardi and Bakhadyr Khoussainov</i>	21:1–21:16
Guarded Teams: The Horizontally Guarded Case <i>Erich Grädel and Martin Otto</i>	22:1–22:17
Order-Invariant First-Order Logic over Hollow Trees <i>Julien Grange and Luc Segoufin</i>	23:1–23:16
Glueability of Resource Proof-Structures: Inverting the Taylor Expansion <i>Giulio Guerrieri, Luc Pellissier, and Lorenzo Tortora de Falco</i>	24:1–24:18
On the Union Closed Fragment of Existential Second-Order Logic and Logics with Team Semantics <i>Matthias Hoelzel and Richard Wilke</i>	25:1–25:16
Expressive Logics for Coinductive Predicates <i>Clemens Kupke and Jurriaan Rot</i>	26:1–26:18
State Space Reduction For Parity Automata <i>Christof Löding and Andreas Tollkötter</i>	27:1–27:16
Syntactic Interpolation for Tense Logics and Bi-Intuitionistic Logic via Nested Sequents <i>Tim Lyon, Alwen Tiu, Rajeev Goré, and Ranald Clouston</i>	28:1–28:16
The Keys to Decidable HyperLTL Satisfiability: Small Models or Very Simple Formulas <i>Corto Mascle and Martin Zimmermann</i>	29:1–29:16
Revisiting the Duality of Computation: An Algebraic Analysis of Classical Realizability Models <i>Étienne Miquey</i>	30:1–30:18

Separation and Renaming in Nominal Sets <i>Joshua Moerman and Jurriaan Rot</i>	31:1–31:17
Parity Games: Another View on Lehtinen’s Algorithm <i>Paweł Parys</i>	32:1–32:15
De Jongh’s Theorem for Intuitionistic Zermelo-Fraenkel Set Theory <i>Robert Passmann</i>	33:1–33:16
Computing Haar Measures <i>Arno Pauly, Dongseong Seon, and Martin Ziegler</i>	34:1–34:17
The Call-By-Value Lambda-Calculus with Generalized Applications <i>José Espírito Santo</i>	35:1–35:12
Dynamic Complexity Meets Parameterised Algorithms <i>Jonas Schmidt, Thomas Schwentick, Nils Vortmeier, Thomas Zeume, and Ioannis Kokkinis</i>	36:1–36:17
Dynamic Complexity of Parity Exists Queries <i>Nils Vortmeier and Thomas Zeume</i>	37:1–37:16

■ Preface

This volume contains the papers presented at CSL 2020, the 28th edition in the series of Computer Science Logic (CSL), the annual conference of the European Association for Computer Science Logic (EACSL). CSL 2020 was held in Barcelona, Spain, 13-16 January 2020. Until 2018 CSL took place in August/September, and CSL 2020 is the first conference in the series scheduled in January.

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

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

CSL 2020 received 82 submissions from 32 countries. The programme committee selected 32 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

- Véronique Cortier (LORIA, France)
- Anuj Dawar (University of Cambridge, UK)
- Artur Jeż (University of Wrocław, Poland)
- Delia Kesner (University Paris Diderot, France)
- Iddo Tzameret (Royal Holloway, UK)

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

A special regular item in the CSL programme is the Ackermann Award presentation. This is the EACSL Outstanding Dissertation Award for Logic in Computer Science. This year, the jury decided to give the Ackermann Award for 2019 to

Antoine Mottet for his PhD thesis entitled *Dichotomies in Constraint Satisfaction Canonical Functions and Numeric CSPs*,

supervised by Manuel Bodirsky at the Technical University of Dresden. The award was officially presented at the conference on the 15th January 2020. The citation of the award,

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).
Editors: Maribel Fernández and Anca Muscholl



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

an abstract of the thesis and a biographical sketch of the recipient are included in the proceedings.

We are very grateful to all the members of the CSL 2020 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, and in particular the chair, Albert Atserias, for taking care of every detail to make the conference enjoyable for all the participants. It was also a pleasure to work with Thomas Schwentick who, as the EACSL president, provided excellent guidance. The proceedings of CSL 2020 are published as a volume in the LIPIcs series. We thank Michael Wagner and all the Dagstuhl/LIPIcs team for their ongoing support and for the high quality preparation of these proceedings.

Maribel Fernández and Anca Muscholl

October 25, 2019

■ Programme Committee

Sandra Alves, University of Porto, Portugal
Takahito Aoto, Niigata University, Japan
Albert Atserias, Technical University of Catalonia, Spain
Manuel Bodirsky, TU Dresden, Germany
James Brotherston, University College London, UK
Rohit Chadha, University of Missouri, USA
Krishnendu Chatterjee, Institute of Science and Technology, Austria
Adriana Compagnoni, Stevens Institute of Technology, USA
Arnaud Durand, University Paris Diderot, France
Maribel Fernández, King's College London, UK (co-chair)
Bernd Finkbeiner, Saarland University, Germany
Masahito Hasegawa, Kyoto University, Japan
Dietrich Kuske, TU Ilmenau, Germany
Angelo Montanari, University of Udine, Italy
Anca Muscholl, University of Bordeaux, France (co-chair)
Prakash Panangaden, McGill University, Canada
Elaine Pimentel, Federal University of Rio Grande do Norte, Brazil
Damien Pous, CNRS - ENS Lyon, France
Femke van Raamsdonk, Vrije Universiteit Amsterdam, Netherlands
Simona Ronchi Della Rocca, University of Torino, Italy
Manfred Schmidt-Schauss, Goethe University, Germany
Lutz Schröder, Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
Lidia Tendera, Opole University, Poland
Szymon Torunczyk, University of Warsaw, Poland
Glynn Winskel, University of Cambridge, UK



■ Organizing Committee

Albert Atserias (UPC, co-chair)
Juan Carlos Martínez (UB, co-chair)
Ilario Bonacina (UPC)
Michal Garlik (UPC)
Tuomas Hakoniemi (UPC)
Joost Joosten (UB)
Alberto Larrauri (UPC)
Moritz Müller (UPC)



■ Ackermann Award 2019

The fifteenth Ackermann Award is presented at CSL'20 in Barcelona, Spain. The 2020 Ackermann Award was open to any PhD dissertation on any topic represented at the annual CSL and LICS conferences that were formally accepted by a degree-granting institution in fulfillment of the PhD degree between 1 January 2017 and 31 December 2018. The Jury received eleven nominations for the 2019 Award. The candidates came from a number of different countries around the world. The institutions at which the nominees obtained their doctorates represent seven different countries in Europe, North America and South America.

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

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

The wide range of excellent candidates presented the jury with a difficult task. After an extensive discussion, one candidate stood out and the jury unanimously decided to award the **2019 Ackermann Award** to:

Antoine Mottet from France, for his thesis

Dichotomies in Constraint Satisfaction Canonical Functions and Numeric CSPs

approved by *Technische Universität Dresden* in 2018.

Citation

Antoine Mottet receives the *2019 Ackermann Award* of the European Association of Computer Science Logic (EACSL) for his thesis

Dichotomies in Constraint Satisfaction Canonical Functions and Numeric CSPs.

Mottet's thesis is a significant contribution to the area of Constraint Satisfaction Problems over infinite domains. It confirms the Dichotomy conjecture of Bodirsky and Pinsker for α_0 -categorical structures for a substantial special case using a generic approach that establishes ties with CSPs for finite structures. As a corollary it yields a new and simpler proof of the complexity dichotomy for MMSNP. It further proves dichotomies for two significant classes of structures that are first-order reducts of $(\mathbb{Z}, <, +)$. The thesis shows a strong mathematical background and a great maturity in Universal Algebra, Model Theory, and Ramsey theory, as well as a strong sense of Computational Complexity.

Background of the Thesis

A Constraint Satisfaction Problem (CSP) can be formulated as a decision problem where the input consists of two structures A, B , and the aim is to determine whether there is a homomorphism from A to B . Much of the study has focused on *non-uniform CSP*, where the structure B is fixed. E.g., if B is a triangle graph then CSP is just the 3-colourability problem. Feder and Vardi conjectured that, for each finite structure B , CSP is either

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



solvable in polynomial time or NP-complete. This famous *dichotomy conjecture* has spawned considerable research and has been confirmed independently by Bulatov and Zhuk in 2017, one of the most significant results in Theoretical Computer Science of the last decade.

However, many problems such as the feasibility problem for a given system of linear inequalities over the rational numbers can be stated in the form of CSP over a set definable in an infinite structure B , but not for any finite B . The case of a general infinite structure is easily shown to be too broad to admit any kind of classification of CSPs. An active research program has been to look at structures that are well-behaved. One class of structures that have received attention are the ω -categorical structures, which include the rational order and the integers with equality. Within these, particular attention has been paid to structures \mathfrak{M} that are *finitely-bounded* and *homogeneous*. The former means that the finite structures that can be embedded in \mathfrak{M} are universally axiomatizable, while the latter means that every isomorphism on a finite substructure of \mathfrak{M} can be extended to \mathfrak{M} . Bodirsky and Pinsker conjectured that the dichotomy conjecture holds for CSPs defined over a reduct of a finitely bounded homogeneous structure. This *dichotomy conjecture for infinite structures* is still open, and has been a touchstone for much further investigation.

Another line of work has been the study of CSP over structures definable from structures related to arithmetic, such as the integers with addition and order. Here the motivation is not from the dichotomy conjecture but from the high relevance of these problems for applications.

Contributions of the Thesis

Mottet's thesis makes significant contributions to both lines of research, the case of finitely bounded homogeneous structures and the case of structures related to arithmetic.

In the case of finitely bounded homogeneous structures, the first major contribution of the thesis is a reduction that allows one to lift tractability results from the finite case to the infinite case. Tractability here means membership in PTIME, but the reduction can also be used to lift results on definability of a CSP via Datalog, a stronger condition than PTIME membership. The reduction and corresponding lifting results give a uniform approach to many tractability results in the literature. A second contribution is a means to lift intractability results from the finite to the infinite case. By combining the two lifting techniques, Mottet is able to establish the Bodirsky-Pinsker for a special case of finitely bounded homogeneous structures, those that are definable over an infinite set with interpretations for a set of unary predicates.

A second contribution in the same line deals with CSPs that are definable in a certain logic: *Monotone Monadic Strict NP* (MMSNP). Feder and Vardi gave a randomized PTIME reduction between (ordinary finite-domain) CSPs and MMSNP, and Kun showed that this reduction could be de-randomized. Putting this together with the recent proof of the dichotomy conjecture, we see that every MMNSP is either NP-complete or in PTIME. The thesis provides a new proof of this result, avoiding derandomization but instead going through the infinite case. In the process, the proof resolves a number of other questions concerning CSPs definable over infinite structures.

The second half of the thesis turns to CSPs related to arithmetic. A first contribution in this line deals with "numeric CSPs": CSPs over structures whose relations are first-order definable in a reduct of the integers with the linear ordering relation. Mottet proves the dichotomy conjecture for such structures, in the process obtaining a characterization of the tractable cases. A second significant result establishes the dichotomy conjecture for CSPs over the integers with addition and a single constant.

In summary, the thesis contributes a set of fundamental results that are relevant to a number of communities within computer science. The depth and breadth of the techniques applied are also extremely impressive, with the proofs making use of a dazzling variety of techniques, ranging from model theory, universal algebra, combinatorics, and complexity theory.

Biographical Sketch

Antoine Mottet obtained a Bachelor's degree in Computer Science at *École Normale Supérieure de Lyon*, France, and a Master's degree in Computer Science at *École Normale Supérieure de Cachan*, France. His PhD work was carried out at the *Technische Universität Dresden* under the supervision of Manuel Bodirsky. Since completing his PhD in 2018, he has been working as a postdoctoral researcher at *Charles University* in Prague, Czech Republic.

Jury

The jury for the **Ackermann Award 2019** consisted of eight members, two of them *ex officio*, namely, the president and the vice-president of EACSL. In addition, the jury also included a representative of SIGLOG (the ACM Special Interest Group on Logic and Computation).

The members of the jury were:

- Christel Baier (TU Dresden),
- Michael Benedikt (University of Oxford),
- Mikołaj Bojańczyk (University of Warsaw),
- Jean Goubault-Larrecq (ENS Paris-Saclay),
- Dexter Kozen (Cornell University),
- Dale Miller (INRIA Saclay), SigLog representative,
- Simona Ronchi Della Rocca (University of Torino), the vice-president of EACSL,
- Thomas Schwentick (TU Dortmund University), the president of EACSL.

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
Mickaël Randour from Belgium.

2016, Marseille:

Nicolai Kraus from Germany

2017, Stockholm:

Amaury Pouly from France.

2018, Birmingham:

Amina Doumane from France.

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

Verification of Security Protocols

Véronique Cortier

LORIA - CNRS, INRIA, Université de Lorraine, Nancy, France

<https://members.loria.fr/VCortier/>

veronique.cortier@loria.fr

Abstract

Cryptographic protocols aim at securing communications over insecure networks like the Internet. Over the past decades, numerous decision procedures and tools have been developed to automatically analyse the security of protocols. The field has now reached a good level of maturity with efficient techniques for the automatic security analysis of protocols

After an overview of some famous protocols and flaws, we will describe the current techniques for security protocols analysis, often based on logic, and review the key challenges towards a fully automated verification.

2012 ACM Subject Classification Security and privacy → Formal security models

Keywords and phrases Security protocols, automated deduction, security

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.1

Category Invited Talk

Funding This work has been partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research (grant agreement No 645865-SPOOC).

1 Description of the talk

Cryptographic protocols aim at securing communications over insecure networks like the Internet. Over the past decades, numerous decision procedures and tools have been developed to automatically analyse the security of protocols. The field has now reached a good level of maturity with efficient techniques for the automatic security analysis of protocols

After an overview of some famous protocols and flaws, we will describe the current techniques for security protocols analysis, often based on logic, and review the key challenges towards a fully automated verification. For example, one well-established tool for analyzing protocols is ProVerif [1], that internally relies on resolution of Horn clauses. ProVerif performs very well in practice but due to this abstraction, it cannot handle protocols with long term states such counters or tables. We have recently realized [2] that these limitations can be overcome with subtle encodings as well as the integration of mature techniques for integers.

Another major challenge is the coverage of privacy properties (e.g. anonymity, untraceability, ballot secrecy) that are typically expressed as equivalence properties. Such properties require novel verification techniques and many tools have been recently developed, such as SPEC [5], DeepSec [3], SatEquiv [4], with different scope and efficiency compromise.

References

- 1 Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, pages 82–96. IEEE Computer Society, June 2001.
- 2 Vincent Cheval, Véronique Cortier, and Mathieu Turuani. A little more conversation, a little less action, a lot more satisfaction: Global states in ProVerif. In *Proceedings of the 31st IEEE Computer Security Foundations Symposium (CSF'18)*, pages 344–358, 2018.



© Véronique Cortier;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 1; pp. 1:1–1:2

Leibniz International Proceedings in Informatics



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

1:2 Verification of Security Protocols

- 3 Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. The DEEPSEC prover. In *Proceedings of the 30th International Conference on Computer Aided Verification (CAV'18)*. Springer, July 2018.
- 4 Véronique Cortier, Stéphanie Delaune, and Antoine Dallon. SAT-Equiv: an efficient tool for equivalence properties. In *Proceedings of the 30th IEEE Computer Security Foundations Symposium (CSF'17)*. IEEE Computer Society Press, August 2017.
- 5 Jeremy Dawson and Alwen Tiu. Automating open bisimulation checking for the spi-calculus. In *IEEE Computer Security Foundations Symposium (CSF 2010)*, 2010.

Symmetric Computation

Anuj Dawar 

Department of Computer Science and Technology, University of Cambridge, U.K.
anuj.dawar@cl.cam.ac.uk

Abstract

We discuss a recent convergence of notions of symmetric computation arising in the theory of linear programming, in logic and in circuit complexity. This leads us to a coherent and robust definition of problems that are efficiently and symmetrically solvable. This is at once a rich class of problems and one for which we have methods for proving lower bounds. In this paper, we take a tour through results which show applications of these methods in a number of areas.

2012 ACM Subject Classification Theory of computation → Circuit complexity; Theory of computation → Complexity theory and logic; Theory of computation → Finite Model Theory; Theory of computation → Complexity classes

Keywords and phrases Descriptive Complexity, Fixed-point Logic with Counting, Circuit Complexity, Linear Programming, Hardness of Approximation, Arithmetic Circuits

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.2

Category Invited Talk

Funding *Anuj Dawar*: Research funded in part by EPSRC grant EP/S03238X/1.

1 Introduction

It has been said that computer science is the science of *abstraction*. Aho and Ullman in their seminal book *Foundations of Computer Science* [1] call it the “mechanization of abstraction”. To model a part of the world computationally is to forget (or “abstract away”) the features that are unnecessary to the computational task at hand and keep only the essential elements in a suitable data model. For example, a widely used data model in the world of algorithm design is that of graphs, which captures a collection of entities and their pairwise relationships. The relationships could reflect compatibility of kidney donors with patients needing transplants or they could pair riders with drivers in a car-pooling system. Once the details are abstracted away, we can use exactly the same *graph matching* algorithm to find a suitable matching in either system. Yet, in the field of computational complexity, which studies the resources required by algorithms and aims to elucidate why some computational problems are inherently intractable, algorithms are usually modelled as *Turing machines*, a low-level model working on strings of bits. This mismatch between the levels of abstraction at which algorithms are formulated, and at which complexity is analyzed is tied to persistent obstacles in complexity theory.

An important feature that distinguishes an abstract data structure, such as a graph, from its concrete representation, such as a pointer list, are its *symmetries*. In a graph, two vertices may appear identical and therefore interchangeable while their concrete representations are distinguished by some feature (such as actual pointer values) that is hidden by the abstraction. Algorithms that work at the higher level of abstraction must respect the symmetries in the abstract data. We use the term *symmetric computation* to describe computation at the abstract level that respects the inherent symmetries of the data.

The mismatch between algorithms working on high-level data structures and complexity defined in terms of low-level machines is one of the central concerns of the field of *descriptive complexity*, which seeks to formulate a theory of complexity at the level of high-level



© Anuj Dawar;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 2; pp. 2:1–2:12

Leibniz International Proceedings in Informatics



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

descriptions. Usually this takes the form of characterizing complexity in terms of *definability* in a logic. The paradigmatic result being Fagin's theorem [22] that the class of problems definable in existential second-order logic is exactly the class NP. The classic question of descriptive complexity, of whether there is a logic that exactly characterizes polynomial-time computation, first posed by Chandra and Harel [13] can be understood as asking whether it is possible to describe efficient algorithms, whenever they exist, at the level of abstraction of the data?

In the quest for a logic for P (see [24]), *fixed-point logic with counting* (FPC) emerged as a logic of reference. Even though Cai et al. [12] demonstrated three decades ago that the logic does not express all graph properties in P, the logic has been the focus of much research in recent years. This is because it has proved remarkably expressive and at the same time we have powerful techniques for proving inexpressibility results for it. Work in recent years has shown that FPC can be seen as capturing a natural class of *symmetric algorithms* inside P, with equivalent formulations in arising in circuit complexity and the theory of linear programming. Thus, the methods for proving inexpressibility results give techniques for showing lower bounds for such algorithms. In this article, I give a brief survey of such results and methods. The survey does not include any proofs, and not much by way of definitions. I attempt to motivate and state the results, placing them in a wider narrative and provide pointers to the original sources.

2 Counting Width

Put simply, FPC is an extension of first-order logic by means of a mechanism for iteration (usually taken to be an *inflationary fixed-point* construct) and a mechanism for counting. The latter allows us to form numerical terms to denote the cardinality of any definable set. The logic has been extensively studied in the context of descriptive complexity theory. A good account of the logic and work on it in the 1990s can be found in Otto's monograph [34]. It was often said at the time that, though Cai, Fürer and Immerman had shown that FPC cannot express all polynomial-time properties of graphs, all natural properties in P are in FPC. We now know this is not true. In particular, the study of constraint satisfaction problems has turned up a host of natural problems that are in P, but not in FPC. Indeed, the constraint satisfaction problems that are in FPC are exactly the ones of *bounded width* (see [5, 10]).

At the same time, research on FPC since the turn of the century has shown the remarkably rich expressive power of the logic. An important strand of this has been the line of work that shows that FPC captures all of P on classes of sparse graphs, or more generally structures with sparse connectivity. This culminates in the result of Grohe [25] which shows that FPC captures P on any class of graphs which excludes some fixed graph as a minor. Grohe's book [25] which gives the proof of this result also provides an excellent, up-to-date definition of and introduction to the logic FPC in its early chapters. For another overview of the logic and its expressive power, see the survey [15].

The key method for proving that some class \mathcal{C} of structures is not definable in FPC is based on the expressive power of *first-order logic* with counting. To be precise, let C^k denote the fragment of first-order logic where we restrict formulas to have no more than k variables altogether, but we are allowed to use them compactly by allowing *counting quantifiers*. That is, we can write $\exists^i x \theta$ to denote that there are at least i elements x for which $\theta(x)$ holds. It is known that for any formula φ of FPC, there is a fixed value k such that, restricted to structures with at most n elements, φ is equivalent to a formula φ_n of C^k . We write \equiv^k to

denote the relation of elementary equivalence for C^k : two structures \mathbb{A} and \mathbb{B} are said to be C^k -equivalent, written $\mathbb{A} \equiv^k \mathbb{B}$, if every sentence of C^k true in one structure is also true in the other. Thus, to show that a class \mathcal{C} is not definable in FPC, it suffices to show that \mathcal{C} is not closed under \equiv^k for any fixed k . This motivates the following definition [17].

► **Definition 1.** For any isomorphism-closed class of finite structures \mathcal{C} , let \mathcal{C}_n denote the collection of structures in \mathcal{C} with at most n elements. We write $\nu_{\mathcal{C}} : \mathbb{N} \rightarrow \mathbb{N}$ for the function such that $\nu_{\mathcal{C}}(n)$ is the least k for which \mathcal{C}_n is closed under \equiv^k . We call $\nu_{\mathcal{C}}$ the counting width of \mathcal{C} .

If \mathcal{C} is definable in FPC, $\nu_{\mathcal{C}}$ is bounded by a constant. On the other hand, there are classes of structures not definable in FPC, indeed problems of very high complexity, which still have bounded counting width. We can understand the property of having bounded counting width as a non-uniform version of FPC definability. For any \mathcal{C} , the class \mathcal{C}_n is a finite collection of structures (up to isomorphism) which can be defined by a single sentence φ_n of $C^{\nu_{\mathcal{C}}(n)}$. This sequence of sentences is uniform when it is generated by a single sentence of FPC. It is also not hard to see that we never need more than n variables to express φ_n , since any structure on n elements can be described completely, up to isomorphism, by a sentence of first-order logic with at most n variables. Thus, $\nu_{\mathcal{C}}(n) \leq n$ for any \mathcal{C} whatsoever.

Cai, Fürer and Immerman [12] gave the first construction of a class of graphs of unbounded counting width, which we call the *CFI construction* for short. Indeed, they showed that there is a class \mathcal{C} with $\nu_{\mathcal{C}} = \Omega(n)$. Such lower bounds were then established for a number of specific problems, either by a construction inspired by that of Cai et al. (for instance in [14] or [5]) or by means of reductions (see [5] and [9]). The reductions involved are those definable in a logic such as FPC or fragments of it, such as first-order logic or Datalog. In general, if we can show that a class \mathcal{C} is reducible to a class \mathcal{D} by means of such a reduction I , and I takes structures of size n to structures of size n^d , then $\nu_{\mathcal{C}} = O(\nu_{\mathcal{D}}^d)$. In particular, if we can bound the size of $I(\mathbb{A})$ by a linear function in the size of \mathbb{A} , we prove that $\nu_{\mathcal{D}} = \Omega(\nu_{\mathcal{C}})$. As was pointed out in [17], this implies in particular that for any non-uniform constraint satisfaction problem, the counting width is either $O(1)$ or $\Omega(n)$, providing a sharp definability dichotomy.

3 Symmetric Circuits

The claim that definability in FPC is a natural formalization of the notion of solvability by means of a *symmetric* polynomial-time algorithm rests on the characterization of FPC as the class of problems decided by polynomially-uniform *symmetric* circuits with threshold gates.

Circuits models have been studied in the context of computational complexity because they seemed a promising route to proving lower bounds. A circuit is really an unfolding of the behaviour of an algorithm for a fixed size of input. The hope is that the difficulty of the computation that it represents can then be studied purely combinatorially in the structure of the circuit. Formally, we have a decision problem that is a language $L \subseteq \{0, 1\}^*$. Such a language can be described by a family of *Boolean functions*: $(f_n)_{n \in \omega} : \{0, 1\}^n \rightarrow \{0, 1\}$, and each f_n can be represented by a *circuit* C_n which is a directed acyclic graph where we think of the vertices as gates suitably labeled by Boolean operators for the internal gates and by inputs x_1, \dots, x_n for the gates without incoming edges. The operators we allow on the internal gates are the *basis*. The standard Boolean basis (\wedge, \vee, \neg) can sometimes be extended, for instance, with *threshold* or *majority* gates.

From our perspective, circuits provide a very low-level model of computation. When we describe decision problems in a high-level descriptive language, such as FPC, the descriptions can, of course, be translated to circuits. The circuits we get as a result of such a translation

have natural *symmetry* properties. In particular, imagine a circuit C that takes as input an n -vertex (directed) graph. That is to say, the inputs to C are n^2 variables labelled x_{ij} ($1 \leq i, j, \leq n$) representing the potential edges in the graph. Now, if C decides a property of graphs that is invariant under isomorphisms, the output of the circuit is unchanged if we permute the vertices of the graph. Given an n -vertex graph G , there are many ways that it can be mapped onto the inputs of the circuit C , one for each bijection between $V(G)$ – the vertices of G – and $\{1, \dots, n\}$. So the output is unchanged under any permutation $\pi \in S_n$ acting on the inputs by the action $x_{ij} \mapsto x_{\pi(i)\pi(j)}$. For circuits obtained from logic, this invariance property is witnessed by a *syntactic* invariance condition. That is, any permutation $\pi \in S_n$ can be extended to an *automorphism* of C which takes each input x_{ij} to $x_{\pi(i)\pi(j)}$. We call circuits with this syntactic invariance condition *symmetric*. The definition extends naturally to relational signatures beyond just graphs. For a relational signature τ , we say that a circuit is τ -symmetric if it accepts at its inputs the encoding of a τ -structure and every permutation of the elements of the structure extends to an automorphism of the circuit. This notion has been studied previously in [21] under the name of generic circuits and in [33] where they were called explicitly order-invariant circuits.

We establish in [2] that the expressive power of FPC is exactly captured by families of polynomially-uniform symmetric circuits in a basis that includes the standard Boolean functions along with either threshold or majority gates. This shows that FPC is, in fact, a natural circuit complexity class and helps to establish the robustness of the class. Indeed, we can, besides threshold or majority gates, also allow gates for arbitrary Boolean functions that are *fully symmetric* (this is observed in [20], where circuit models are considered using gates which are not fully symmetric). Here a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is fully symmetric if it is unchanged by *any* permutation of its inputs. Equivalently, $f(x) = f(y)$ whenever x and y are two binary strings with the same number of 1s. Hence, FPC consists of exactly those properties that are decidable by symmetric circuits using any fully symmetric functions in the basis, justifying its claim to be a natural symmetric fragment of P.

One consequence of the circuit characterization of FPC is that we can see the CFI construction as giving a circuit lower bound. Indeed, like most circuit lower bounds, this one also works for the non-uniform circuit families. We can prove that any family of polynomial-size τ -symmetric circuits (even a non-uniform one) accepts a class of structures of bounded counting width. The relationship can be stated more generally, not just for polynomial size circuits.

► **Theorem 2** ([2]). *For any $\epsilon > 0$ and any family $(C_n)_{n \in \omega}$ of symmetric circuits of size s where $s(n) \leq 2^{n^{1-\epsilon}}$, the class of structures accepted by the circuits has counting width $O(\frac{\log s}{\log n})$.*

We can, indeed, prove this relationship directly, without going through the translation into logic. The usual method of showing that $\mathbb{A} \equiv^k \mathbb{B}$ for a pair of structures \mathbb{A} and \mathbb{B} is by means of the k -pebble bijection game of Hella [28]. This game can be used directly as a lower bound method for symmetric circuits (see [19] for an exposition). The key idea is that we prove in [2] that in a circuit of size s (within the bounds of Theorem 2), the stabilizer group of each gate has a small *support*, of size $O(\frac{\log s}{\log n})$. We can then show that a symmetric circuit with support of size k cannot distinguish two structures \mathbb{A} and \mathbb{B} on which Duplicator has a winning strategy in the $2k$ -pebble bijection game.

4 Linear Programming

Linear programming is a widely used approach to solving combinatorial optimization problems. It provides a powerful framework within which optimization problems can be represented, as well as efficient methods for solving the resulting programs. In particular, it is known since the work of [29] on the ellipsoid method that there are polynomial-time algorithms that solve linear programs. In [3], we show that this can be expressed in FPC. That is to say, we consider a natural representation of linear programs as relational structures, where the variables and the constraints are unordered sets, and we show that there is an FPC sentence that defines those linear programs that are feasible and an FPC interpretation that, in any such program, defines a representation of the optimum value of a linear objective function.

We also consider linear programs that are not given *explicitly*, but rather in terms of a separation oracle. The classic example is that of the linear program that encodes the *maximum matching* problem. Given a weighted graph G , the linear program that encodes its matching problem is exponential in the size of G . However, the ellipsoid method can be used in such cases as long as we have a means of determining, for any vector x , whether it is in the polytope P described by the constraint matrix and, if it is not, a hyperplane that separates x from the polytope. This is known as a *separation oracle* for P . It is shown in [3] that, as long as a separation oracle for a polytope P is itself definable in FPC, then the corresponding linear programming optimization problem can also be defined in FPC. By showing that a separation for the matching problem is definable in FPC we were able to prove the following, settling a long-standing open question [11].

► **Theorem 3** ([3]). *The size of a maximum matching in a graph is definable in FPC.*

Symmetry of linear programs is a property that has been studied in the literature in its own right. Yannakakis [38] initiated the study of symmetric extended formulations. Again, let us consider graphs over the vertex set $[n]$. We can consider these as functions $G : X \rightarrow \{0, 1\}$ where $X = \{x_{ij} \mid i, j \in [n]\}$ is the set of potential edges. Equivalently, we can think of graph as a 0-1 valued vector in the Euclidean space \mathbb{R}^X .

Consider, in particular, $P \subseteq \{0, 1\}^X$ which is the collection of simple cycles of length n . The *convex hull* of P , $\text{conv}(P) \subseteq \mathbb{R}^X$ is known as the *travelling salesman polytope* or the *Hamiltonian cycle polytope*. Solving the travelling salesman problem amounts to optimizing a linear function over P , and determining whether a graph $G : X \rightarrow \{0, 1\}$ has a Hamiltonian cycle is the same as determining whether there is a point in P consistent with $x \leq G(x)$ for all $x \in X$. Thus, if we could represent $\text{conv}(P)$ by a set of linear constraints of size polynomial in n , we could solve these NP-hard problems in polynomial time. Yannakakis proved that $\text{conv}(P)$ does not have a polynomial-size *symmetric extended formulation*. That is to say, we cannot obtain it as the projection of a polytope $Q \subseteq \mathbb{R}^{X \times Y}$ using additional variables Y , as long as Q is symmetric. The notion of symmetry is the natural one. Any permutation of $[n]$ has a natural action on X and hence on \mathbb{R}^X . The symmetry requirement says that for any such permutation $\pi \in S_n$ we can find a permutation σ of Y such that for $\mathbf{xy} \in \mathbb{R}^{X \times Y}$, $\mathbf{xy} \in Q$ if, and only if, $\pi(\mathbf{x})\sigma(\mathbf{y}) \in Q$. While the lower bound proof of Yannakakis relies heavily on the notion of symmetry, it turns out that this is not essential to the result. A long line of work originating with the result of Yannakakis culminated in a proof by Rothvoß [35] that shows that the Hamiltonian cycle polytope does not have polynomial-size extended formulations, symmetric or not. It is worth remarking that these lower bound results on the size of the Hamiltonian cycle polytope are obtained by means of reductions from the *matching* polytope. That is, even though the problem of determining the maximum matching in a graph is known to be in polynomial-time, there is no polynomial-size extended formulation of it that yields a linear program of polynomial-size.

We now consider a different way of representing the set $\mathcal{C}_n \subseteq \{0, 1\}^X$ as a linear program. We say that a polytope $P \subseteq \mathbb{R}^X$ recognizes \mathcal{C}_n if $\mathcal{C}_n \subseteq P$ and $\{0, 1\}^n \setminus \mathcal{C}_n$ is disjoint from P . Now, a class \mathcal{C} of structures that is decidable in polynomial time necessarily is recognized by a polynomial-size family of extended polytopes. That is, a P recognizing it can always be obtained as the projection of some $Q \subseteq \mathbb{R}^{X \times Y}$ of polynomial size. This was already shown by Yannakakis [38] and really amounts to establishing that linear programming is complete for P under (say) logarithmic-space reductions. But, here we have dropped the symmetry requirement. What classes of structures are recognized by *symmetric* families of extended polytopes? It turns out that they are exactly the classes of bounded counting width [6].

► **Theorem 4** ([6]). *For any $\epsilon > 0$ and any family $(Q_n)_{n \in \omega} \subseteq \mathbb{R}^{X \times Y}$ of symmetric linear programs of size at most s where $s(n) \leq 2^{n^{1-\epsilon}}$, the class of structures accepted has counting width $O(\frac{\log s}{\log n})$.*

This implies, in particular, that there *is* a family of symmetric linear programs that recognizes the class of graphs with a perfect matching in this sense, but there is still provably not one for the class of graphs with a Hamiltonian cycle, in contrast with the results of Yannakakis.

5 Lift-and-Project Hierarchies

In Section 4, we consider extended formulations of linear programs. Such extended formulations are widely used in applications of linear programming (and its extensions) to combinatorial optimization problems. Typically, we can formulate a combinatorial optimization problem as a $\{0, 1\}$ -integer linear programming problem. The solutions to this form a set $S \subseteq \{0, 1\}^X$. The linear programming relaxation takes this formulation and relaxes the condition that a variable x must take values in the set $\{0, 1\}$ and replaces it with $0 \leq x \leq 1$. This defines a polytope $P \subseteq \mathbb{R}^X$ which includes all the points in S . Extended formulations are obtained in an attempt to add additional constraints (including additional variables) so that the projection onto \mathbb{R}^X gives us a better approximation of the convex hull of S . There are several systematic ways of constructing extended formulations that give infinite hierarchies interpolating between P and $\text{conv}(S)$. There are hierarchies, not only of linear programs but also of semidefinite programs. They include the Sherali-Adams, Lovasz-Schrijver and Lasserre hierarchies (see [31]).

These hierarchies have an interesting connection with symmetric computation as we have defined it, and especially with the notion of counting width. The first connection, established independently by Atserias and Maneva [7] and Malkin [32] is in connection with the Sherali-Adams relaxations of the *graph isomorphism* integer program. In essence, they show that the equivalence relation on graphs that is induced by the k th Sherali-Adams relaxation of this program is sandwiched between the equivalences \equiv^k and \equiv^{k+1} . One consequence is that the CFI construction can be used to show that no finite level of the hierarchy gives us graph isomorphism exactly. See [26] for an alternative account, which also constructs a family of relaxations that correspond exactly to \equiv^k . Taking this further, Atserias and Ochremiak [8] show that relaxations of graph isomorphism based on semi-definite programming cannot do better. In particular, they extend the FPC simulation of the ellipsoid method from [3] to show that the class of feasible semidefinite programs itself has bounded counting width. Note that feasibility of semidefinite programs is not known to be in P, so we would not expect it to be in FPC, so the non-uniform definition of counting width is essential here.

A further connection between lift-and-project hierarchies and counting width is established in the context of constraint optimization problems. These form a very general class of combinatorial optimization problems to which the method of systematic extended formulations

can be applied. Constraint satisfaction problems (CSP) are usually defined as decision problems where we are given a collection V of variables, a domain D of values and *constraints*. A constraint is a pair (\mathbf{x}, R) , where $\mathbf{x} \in V^k$ is a tuple of variables and $R \subseteq D^k$ a relation on D . An assignment $h : V \rightarrow D$ satisfies the constraint (\mathbf{x}, R) if $h(\mathbf{x}) \in R$. The problem is to decide if all the constraints can be simultaneously satisfied. If we fix the domain D in advance as well as the set Γ of relations on D that can appear in the constraints, we can see $\mathbb{D} = (D, \Gamma)$ as a finite relational structure. The instance is then a structure \mathbb{A} in the same vocabulary and we define $\text{CSP}(\mathbb{D})$ to be the class of structures \mathbb{A} such that there is a homomorphism from \mathbb{A} to \mathbb{D} . This view of CSP as essentially homomorphism problems is due to Feder and Vardi [23]. As noted above, we have a known dichotomy with respect to the counting width of $\text{CSP}(\mathbb{D})$: either $\nu_{\text{CSP}(\mathbb{D})} = O(1)$ or $\nu_{\text{CSP}(\mathbb{D})} = \Omega(n)$. Interestingly, this lifts to an interesting dichotomy for constraint *optimization* problems formulated as *finite-valued* CSP.

A *finite-valued constraint satisfaction problem* (VCSP) is given by a finite set D and a collection Γ of functions $f : D^k \rightarrow \mathbb{Q}_+$. An instance of the problem is a set V of variables along with a set C of constraints, each of which is a triple $c = (\mathbf{x}, f, w)$ with $f \in \Gamma$, $\mathbf{x} \in V^k$ where k is the arity of f and $w \in \mathbb{Q}_+$. The algorithmic problem is to find an assignment $h : V \rightarrow D$ of values to the variables which minimizes $\sum_{c \in C} w_c f_c(h\mathbf{x}_c)$. As usual, we can obtain a decision problem from the optimization problem by including with an instance an explicit threshold t . Thus, we think of $\text{VCSP}(D, \Gamma)$ as the decision problem of determining, given (V, C, t) whether there is an assignment $h : V \rightarrow D$ such that $\sum_{c \in C} w_c f_c(h\mathbf{x}_c) \leq t$. The following dichotomy is established in [16]:

► **Theorem 5.** *For every D and Γ , either $\nu_{\text{VCSP}(D, \Gamma)} = O(1)$ or $\nu_{\text{VCSP}(D, \Gamma)} = \Omega(n)$.*

Moreover, the cases of unbounded counting width co-incide exactly with the cases known to be NP-hard, while the bounded width ones are all definable in FPC.

The lower bound on counting width established in Theorem 5 has interesting consequences for lift-and-project hierarchies. We can define a basic linear programming (BLP) formulation for any $\text{VCSP}(D, \Gamma)$ and show that it can be constructed by an FPC interpretation from an instance of $\text{VCSP}(D, \Gamma)$. Moreover, each fixed level of the standard lift-and-project hierarchies, such as the Lasserre hierarchy, over this program is also given by an FPC interpretation. Using the fact that solvability of semidefinite programs has bounded counting width, we can then establish that if $\nu_{\text{VCSP}(D, \Gamma)} = \Omega(n)$, then any bounded number of levels of the Lasserre hierarchy cannot yield a solution to $\text{VCSP}(D, \Gamma)$. These are strong algorithmic lower bounds obtained by means of the methods of counting width. In particular, the following dichotomy is established in [17].

► **Theorem 6.** *If for some (D, Γ) , $t : \mathbb{N} \rightarrow \mathbb{N}$ is a function such that any instance (V, C) of $\text{VCSP}(D, \Gamma)$ is solved exactly by considering the $t(n)^{\text{th}}$ Lasserre lift of the basic linear programming relaxation of (V, C) , then $t(n) = \Omega(\nu_{\text{VCSP}(D, \Gamma)}(n))$.*

6 Hardness of Approximation

In the 1990s, the field of computational complexity was transformed by the PCP theorem [4], and the proofs of hardness of approximation that flowed from it. This showed that, assuming $P \neq NP$, not only is it impossible to have efficient algorithms to solve various NP-hard optimization problems exactly, it is also impossible to have efficient algorithms that solve the problem approximately. In the years since then, this has led to the development of a thriving field studying the hardness of approximation.

For many specific NP-hard optimization problems we know an exact ratio α such that there is a polynomial-time algorithm that solves the problem α -approximately, but no such algorithm guaranteeing a better ratio unless $P = NP$. For instance, consider the problem MAX3SAT, where we are given a Boolean formula φ in 3CNF and we are required to determine the maximum number m of clauses of φ that can be simultaneously satisfied. A trivial algorithm can approximate the value of m up to a ratio of $\frac{7}{8}$, but unless $P = NP$, no polynomial-time algorithm can guarantee an approximation ratio of $\frac{7}{8} + \epsilon$ for any ϵ [27]. There are other problems where there is a gap between the best known achievable approximation ratio and the best known lower bound. For instance, it is known that the size of a vertex cover in a graph can be approximated up to a factor of 2 by a polynomial-time algorithm. On the other hand, we know that unless $P = NP$, there is no algorithm that will achieve an approximation ratio better than $\sqrt{2}$ [30]. It is conjectured that the lower bound can be improved to $2 - \epsilon$, but this remains open.

In [9] we were able to reproduce these lower bounds as *unconditional* inexpressibility results for FPC. That is, we can prove that there is no term of FPC that can define, given a Boolean formula φ in 3CNF, a number that is guaranteed to be within $\frac{7}{8} + \epsilon$ of the maximum number of clauses satisfiable in φ . Similarly, we cannot define in FPC a number guaranteed to be within a factor of $\sqrt{2}$ of the size of the minimum vertex cover in a graph. In short, even if $P = NP$ we can still say that there is no polynomial-time *symmetric* algorithm that achieves such approximation ratios. On the other hand, the algorithms that achieve the upper bounds for these problems are easily seen to be in FPC, i.e. they can be implemented symmetrically. Thus, one can see the hardness of approximation results as telling us something fundamental about the limits of symmetric computation, regardless of whether or not $P = NP$.

The PCP theorem established the hardness of approximating MAX3SAT by giving a reduction from the satisfiability decision problem to itself. It gives a way of translating (in polynomial time) a Boolean formula φ into a formula φ' such that if φ is satisfiable then so is φ' and if φ is not satisfiable then in φ' , no more than a $1 - \epsilon$ fraction of the clauses can be simultaneously satisfied (for some explicit constant ϵ). The value of ϵ can then be amplified to be arbitrarily close to $\frac{1}{8}$ by further reductions as in [27]. To prove the undefinability of the approximation in FPC, what we show is that the 3CNF formulas that are satisfiable and those that are at most $\frac{7}{8} + \epsilon$ -satisfiable cannot be separated by any class of bounded counting width.

► **Theorem 7** ([9]). *For any $\epsilon > 0$, if \mathcal{C} is a class of 3CNF formulas that contains all satisfiable formulas and does not contain any that are not $(\frac{7}{8} + \epsilon)$ -satisfiable, then $\nu_{\mathcal{C}} = \Omega(n)$.*

Theorem 7 is established by means of a reduction from the problem MAX3XOR of maximizing the number of satisfiable clauses in a 3XOR formula. A gap similar to that in Theorem 7 is first established for 3XOR.

► **Theorem 8** ([9]). *For any $\epsilon > 0$, if \mathcal{C} is a class of 3XOR formulas that contains all satisfiable formulas and does not contain any that are not $(\frac{1}{2} + \epsilon)$ -satisfiable, then $\nu_{\mathcal{C}} = \Omega(n)$.*

This is proved by means a variant of the CFI construction. In previous versions of the CFI construction, the aim is to construct a pair of structures \mathbb{A} and \mathbb{B} which are \equiv^k -equivalent, but differ minimally with respect to some property of interest (such as satisfiability, or graph 3-colourability). The equivalence with respect to \equiv^k is, of course, proved using Spoiler-Duplicator games. In the present instance, the aim is to show indistinguishability of a pair of structures which differ significantly on some numeric parameter (such as the number of clauses that can be satisfied, or the size of the smallest vertex cover). This poses significant new challenges.

While the work reported in [9] establishes counting-width lower bounds for approximation of MAX3SAT, MAX3XOR and VERTEX COVER, there is a substantial body of work on hardness of approximation that we have only begun to explore from the point of view of definability. It would be interesting to establish bounds for problems like MAXCUT and MAX2SAT where there are gaps between the best known upper and lower bounds for approximability in the context of polynomial-time algorithms. What would be even more exciting would be to show a bound for symmetric algorithms that was stronger than one known for general polynomial-time algorithms.

7 Arithmetic Circuits

As a final topic, we turn to another model of computation, that of *arithmetic circuits*. These are intended to model computation at a level where arithmetic operations such as addition and multiplication are of unit cost (see [37]).

Formally, an arithmetic circuit over a field K and a set of variables X is a directed acyclic graph where every input (i.e. node of indegree 0) is labelled by an element of X or an element of K , and every internal node is labelled either $+$ (a sum gate) or \times (a product gate). A distinguished *output* gate can then be seen as computing a polynomial in the ring $K[X]$. In the field of arithmetic circuit complexity, we are concerned with determining for various polynomials, what is the size of the smallest circuit that computes it.

Two polynomials (strictly speaking they are families of polynomials) that are much studied in the field are the *determinant* and the *permanent*. They are both defined on a set of variables X representing the entries of an $n \times n$ matrix, so $X = \{x_{ij} \mid 1 \leq i, j \leq n\}$. The determinant is defined as $\det(X) = \sum_{\pi \in S_n} (-1)^{\text{sgn}(\pi)} \prod_i x_{i\pi(i)}$, where $\text{sgn}(\pi)$ is 0 if π is an even permutation and 1 if it is an odd permutation. The permanent is defined similarly, but without the sign. So, $\text{perm}(X) = \sum_{\pi \in S_n} \prod_i x_{i\pi(i)}$. Written this way, the size of the expressions defining the polynomials is exponential in n , due to the sum over $n!$ permutations. Nonetheless, it is known that there are polynomial-size circuits for computing the determinant, and these can be easily obtained from polynomial time algorithms for computing it. On the other hand, it is conjectured that there are no polynomial-size circuits for computing the permanent. Indeed, this is equivalent to Valiant's conjecture that VP is different to VNP, the analogue of the $P \neq NP$ conjecture for arithmetic circuits [36].

It is clear from their definitions that both $\det(X)$ and $\text{perm}(X)$ are invariant under permutations of the variables X which are induced by the natural action of S_n . In other words, for any permutation $\pi \in S_n$, if we permute X by mapping x_{ij} to $x_{\pi(i)\pi(j)}$, it does not change either $\det(X)$ or $\text{perm}(X)$. So, it makes sense to ask whether these polynomials can be computed by polynomial-size *symmetric* circuits in the sense of Section 3. The definition of such circuits is an easy extension of the idea presented in that section: an arithmetic circuit on the variable set $X = \{x_{ij} \mid 1 \leq i, j \leq n\}$ is symmetric if every permutation $\pi \in S_n$ acting on the indices extends to an automorphism of the circuit. In recent work [18], we have been able to show that $\det(X)$ can, indeed, be computed by polynomial-size symmetric circuits, and $\text{perm}(X)$ provably cannot. The upper bound for the determinant is obtained by showing that known fast parallel algorithms for computing the determinant can be done symmetrically. Note that the standard algorithm based on Gaussian elimination cannot be carried out symmetrically in polynomial time. This is known from the counting width lower bounds on solvability of systems of linear equations.

For the lower bound on computing the permanent, we rely on another CFI-like construction. To be precise, we show that we can construct for each k , a pair of bipartite graphs G and H such that $G \equiv^k H$ but G and H have different numbers of perfect matchings. This is then

related to arithmetic circuits by means of a translation. If we had a polynomial-size arithmetic circuit for computing $\text{perm}(X)$, we could get a polynomial-size circuit with Boolean inputs which computes the number of perfect matchings in a bipartite graph. But, for such circuits, we can show that the output must be invariant under \equiv^k for some constant k .

One interesting aspect of this proof is the role of perfect matchings in a graph. We know that the decision problem of determining whether or not a graph has a perfect matching has bounded counting width. For bipartite graphs, a construction of Blass, Gurevich and Shelah [11] shows that it has width 2, and the result of Anderson et al. [3] shows that even for general graphs, it is constant. Nevertheless, it turns out that the *number* of perfect matchings is not a \equiv^k -invariant for any k , even for bipartite graphs.

8 Conclusion

The notion of symmetry in computation arises naturally when we consider algorithms described at a high-level of abstraction and how they are translated to low-level models. The tension between preserving symmetry and efficient implementation rests to some extent on the fact that we cannot efficiently detect symmetries, e.g. we do not know how to efficiently determine if two graphs are isomorphic. What is remarkable is that a number of distinct notions of symmetry, arising in different fields, such as database theory, combinatorial optimization and circuit complexity converge on a common core. At the heart of the theory that emerges from this core is a graded approximation of isomorphism – the equivalence relations \equiv^{C^k} – which has itself been widely studied from many independent directions. This leads to a coherent and robust notion of efficient symmetric computation. On the one hand it is a remarkably powerful model and on the other hand, we have methods for proving lower bounds for it. There seems to be a wealth of possible areas of application for it.

References


- 1 A. V. Aho and J. D. Ullman. *Foundations of Computer Science, C Edition*. Computer Science Press / W. H. Freeman, 1992.
- 2 M. Anderson and A. Dawar. On Symmetric Circuits and Fixed-Point Logics. *Theory Comput. Syst.*, 60(3):521–551, 2017. doi:10.1007/s00224-016-9692-2.
- 3 M. Anderson, A. Dawar, and B. Holm. Solving Linear Programs without Breaking Abstractions. *J. ACM*, 62, 2015.
- 4 S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof Verification and the Hardness of Approximation Problems. *J. ACM*, 45(3):501–555, 1998.
- 5 A. Atserias, A. Bulatov, and A. Dawar. Affine Systems of Equations and Counting Infinitary Logic. *Theoretical Computer Science*, 410(18):1666–1683, 2009.
- 6 A. Atserias, A. Dawar, and J. Ochremiak. On the Power of Symmetric Linear Programs. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, pages 1–13, 2019. doi:10.1109/LICS.2019.8785792.
- 7 A. Atserias and E. N. Maneva. Sherali-Adams relaxations and indistinguishability in counting logics. *SIAM J. Comput.*, 42:112–137, 2013.
- 8 A. Atserias and J. Ochremiak. Definable Ellipsoid Method, Sums-of-Squares Proofs, and the Isomorphism Problem. In *33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, 2018.
- 9 Albert Atserias and Anuj Dawar. Definable Inapproximability: New Challenges for Duplicator. In *27th EACSL Annual Conference on Computer Science Logic, CSL*, pages 7:1–7:21, 2018. doi:10.4230/LIPIcs.CSL.2018.7.
- 10 L. Barto and M. Kozik. Constraint Satisfaction Problems Solvable by Local Consistency Methods. *J. ACM*, 61:3:1–3:19, 2014.

- 11 A. Blass, Y. Gurevich, and S. Shelah. Choiceless Polynomial Time. *Annals of Pure and Applied Logic*, 100:141–187, 1999.
- 12 J-Y. Cai, M. Fürer, and N. Immerman. An Optimal Lower Bound on the Number of Variables for Graph Identification. *Combinatorica*, 12(4):389–410, 1992.
- 13 A. Chandra and D. Harel. Structure and Complexity of Relational Queries. *Journal of Computer and System Sciences*, 25:99–128, 1982.
- 14 A. Dawar. A Restricted Second Order Logic for Finite Structures. *Information and Computation*, 143:154–174, 1998.
- 15 A. Dawar. The Nature and Power of Fixed-Point Logic with Counting. *ACM SIGLOG News*, 2(1):8–21, 2015.
- 16 A. Dawar and P. Wang. A Definability Dichotomy for Finite Valued CSPs. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015*, pages 60–77, 2015.
- 17 A. Dawar and P. Wang. Definability of semidefinite programming and Lasserre lower bounds for CSPs. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS, 2017*. doi:10.1109/LICS.2017.8005108.
- 18 A. Dawar and G. Wilsenach. Symmetric Arithmetic Circuits. forthcoming.
- 19 Anuj Dawar. On Symmetric and Choiceless Computation. In *Topics in Theoretical Computer Science - The First IFIP WG 1.8 International Conference, TTCS*, pages 23–29, 2015. doi:10.1007/978-3-319-28678-5_2.
- 20 Anuj Dawar and Gregory Wilsenach. Symmetric Circuits for Rank Logic. In *27th EACSL Annual Conference on Computer Science Logic, CSL*, pages 20:1–20:16, 2018. doi:10.4230/LIPIcs.CSL.2018.20.
- 21 L. Denenberg, Y. Gurevich, and S. Shelah. Definability by Constant-depth Polynomial-size Circuits. *Information and Control*, 70:216–240, 1986.
- 22 R. Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol 7*, pages 43–73, 1974.
- 23 T. Feder and M.Y. Vardi. Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study Through Datalog and Group Theory. *SIAM Journal of Computing*, 28:57–104, 1998.
- 24 M. Grohe. The Quest for a Logic Capturing PTIME. In *Proc. 22nd IEEE Symp. on Logic in Computer Science*, pages 267–271, 2008.
- 25 M. Grohe. *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Lecture Notes in Logic. Cambridge University Press, 2017.
- 26 M. Grohe and M. Otto. Pebble Games and linear equations. *J. Symb. Log.*, 80:797–844, 2015.
- 27 J. Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.
- 28 L. Hella. Logical Hierarchies in PTIME. *Information and Computation*, 129:1–19, 1996.
- 29 L. G. Khachiyan. Polynomial Algorithms in Linear Programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.
- 30 S. Khot, D. Minzer, and M. Safra. Pseudorandom Sets in Grassmann Graph Have Near-Perfect Expansion. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 592–601, 2018. doi:10.1109/FOCS.2018.00062.
- 31 M. Laurent. A Comparison of the Sherali-Adams, Lovász-Schrijver, and Lasserre Relaxations for 0-1 Programming. *Math. Oper. Res.*, 28:470–496, 2003. doi:10.1287/moor.28.3.470.16391.
- 32 P. N. Mankin. Sherali-Adams relaxations of graph isomorphism polytopes. *Discrete Optimization*, 12:73–97, 2014.
- 33 M. Otto. The Logic of Explicitly Presentation-Invariant Circuits. In *Computer Science Logic, 10th International Workshop, CSL '96, Annual Conference of the EACSL*, pages 369–384, 1996.
- 34 M. Otto. *Bounded Variable Logics and Counting — A Study in Finite Models*, volume 9 of *Lecture Notes in Logic*. Springer-Verlag, 1997.
- 35 T. Rothvoß. The Matching Polytope has Exponential Extension Complexity. In *Symp. Theory of Computing, STOC 2014*, pages 263–272, 2014.

2:12 Symmetric Computation

- 36 L. G. Valiant. Completeness Classes in Algebra. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing STOC*, pages 249–261, 1979. doi:10.1145/800135.804419.
- 37 J. von zur Gathen. Algebraic Complexity Theory. *Ann. Rev. Comput. Sci.*, 3:317–347, 1988. doi:10.1146/annurev.cs.03.060188.001533.
- 38 M. Yannakakis. Expressing Combinatorial Optimization Problems by Linear Programs. *J. Comput. Syst. Sci.*, 43(3):441–466, 1991.

Solving Word Equations (And Other Unification Problems) by Recompression

Artur Jeż 

University of Wrocław, Poland

<http://www.ii.uni.wroc.pl/~aje>

aje@cs.uni.wroc.pl

Abstract

In word equation problem we are given an equation $u = v$, where both u and v are words of letters and variables, and ask for a substitution of variables by words that equalizes the sides of the equation. This problem was first solved by Makanin and a different solution was proposed by Plandowski only 20 years later, his solution works in PSPACE, which is the best computational complexity bound known for this problem; on the other hand, the only known lower-bound is NP-hardness. In both cases the algorithms (and proofs) employed nontrivial facts on word combinatorics.

In the paper I will present an application of a recent technique of recompression, which simplifies the known proofs and (slightly) lowers the complexity to linear nondeterministic space. The technique is based on employing simple compression rules (replacement of two letters ab by a new letter c , replacement of maximal repetitions of a by a new letter), and modifying the equations (replacing a variable X by bX or Xa) so that those operations are sound and complete. In particular, no combinatorial properties of strings are used.

The approach turns out to be quite robust and can be applied to various generalizations and related scenarios (context unification, i.e. equations over terms; equations over traces, i.e. partially ordered words; ...).

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Formalisms; Theory of computation → Grammars and context-free languages; Theory of computation → Design and analysis of algorithms; Theory of computation → Tree languages

Keywords and phrases word equation, context unification, equations in groups, compression

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.3

Category Invited Talk

Funding Work supported under National Science Centre, Poland project number 2017/26/E/ST6/00191.

1 Introduction

1.1 Word equations

The word equation problem, i.e. solving equations in the algebra of words, was first investigated by Markov in the fifties. In this problem we get as an input an equation of the form

$$u = v$$

where u and v are strings of letters (from a fixed alphabet) as well as variables and a *solution* is a substitution of words for variables that turns this formal equation into a true equality of strings of letters (over the same fixed alphabet). It is relatively easy to show a reduction of this problem to the Hilbert's 10-th problem, i.e. the question of solving systems of Diophantine equations. Already then it was generally accepted that Hilbert's 10-th problem is undecidable and Markov wanted to show this by proving the undecidability of word equations.



© Artur Jeż;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 3; pp. 3:1–3:17

Leibniz International Proceedings in Informatics



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

3:2 Solving Word Equations (And Other Unification Problems) by Recompression

Alas, while Hilbert's 10-th problem is undecidable, the word equation problem is *decidable*, which was shown by Makanin [36]. The termination proof of his algorithm is very complex and yields a relatively weak bound on the computational complexity, thus over the years several improvements and simplifications over the original algorithm were proposed [21, 56, 27, 19]. Simplifications have many potential advantages: it seems natural that simpler algorithm can be generalised or extended more easily (for instance, to the case of equations in groups) than a complex one. Moreover, simpler algorithm should be more effective in practical applications and should have a lower complexity bounds.

Subcases. It is easy to show NP-hardness for word equations, so far no better computational complexity lower bound is known. Such hardness stimulated a search for a restricted subclasses of the problem for which efficient (i.e. polynomial) algorithms can be given [2]. One of such subclasses is defined by restricting the amount of different variables that can be used in an equation: it is known that equations with one [13, 29] and two [2, 20, 12] variables can be solved in polynomial time. Already for three variables it is not known, whether they are in NP or not [50] and partial results require nontrivial analysis [50].

Generalisations. Since Makanin's original solution much effort was put into extending his algorithm to other structures. Three directions seemed most natural:

- adding constraints to word equations;
- equations in free groups;
- partial commutation;
- equations in terms.

Constraints From the application point of view, it is advantageous to consider word equations that can also use some additional constraints, i.e. we require that the solution for X has some additional properties. This was first done for regular constraints [56], on the other hand, for several types of constraints, for instance length-constraints, it is still open, whether the resulting problem is decidable or not (it becomes undecidable, if we allow counting occurrences of particular letter in the substitutions and arithmetic operations on such counts [1]).

Free groups From the algebraic point of view, the word equation problem is solving equations in a free semigroup. It is natural to try to extend an algorithm from the free semigroup also to the case of free groups and then perhaps even to a larger class of groups (observe, that there are groups and semigroups for which the word problem is undecidable). The first algorithm for the group case was given by Makanin [37, 38], his algorithm was not primitively-recursive [28]. Furthermore, Razborov showed that this algorithm can be used to give a description of all solutions of an equation [48] (more readable description of the Razborov's construction is available in [25]). As a final comment, note that such a description was the first step in proving the Tarski's Conjecture for free groups (that the theory of free groups is decidable) [26].

Partial commutation Another natural generalization is to allow partial commutation between the letters, i.e. for each pair of letters we specify, whether $ab = ba$ or not. Such partially commutative words are usually called traces, after Mazurkiewicz, and the corresponding groups are usually known as Right-Angled Artin Groups, RAAGs for short. Decidability for trace equations was shown by Matiyasevich [39] and for RAAGs by Diekert and Muscholl [11]. In both cases the main step in the proof was a reduction from a partially commutative case to a no-commutative one.

Terms We can view words as very simple terms: each letter is a function symbol of arity 1. In this way word equations are equations over (very simple) terms. It is known, that term unification can be decided in polynomial time, assuming that variables represent closed (full) terms [49]; thus such a problem is unlikely to generalise word equations.

A natural generalisation of term unification and word equations is a *second-order unification*, in which we allow variables to represent functions that take arguments (which need to be closed terms). However, it is known that this problem is undecidable, even in many restricted subcases [18, 14, 30, 32]. *Context unification* [4, 5, 51] is a natural problem “in between”: we allow variables representing functions, but we insist that they use their argument *exactly once*. It is easy to show that such defined problem generalises word equations, on the other hand, the undecidability proofs for second-order unification do not transfer directly to this model.

Being a natural generalisation is not enough to explain the interest in this problem, more importantly, context unification has natural connections with other, well-studied problems (equality up to constraints [40], linear second-order unification [33, 30], one-step term rewriting [41], bounded second order unification [53], ...). Unfortunately, for over two decades the question of decidability of context unification remained open. Despite intensive research, not much is known about the decidability of this problem: only results for some restricted subcases are known: [5, 52, 31, 30, 55, 54, 34, 17].

1.2 Compression and word equations

For more than 20 years since Makanin’s original solution there was very small progress in algorithms for word equations: the algorithm was improved in many places, in particular this lead to a better estimation of the running time; however, the main idea (and the general complexity of the proof) was essentially the same.

The breakthrough was done by Plandowski and Rytter [47], who, for the first time, used the compression to solve word equations. They showed, that the shortest solution (of size N) of the word equation (of size n) has an SLP¹ representation of size $\text{poly}(n, \log N)$; using the algorithm for testing the equality of two SLPs [43] this easily yields a (non-deterministic) algorithm running in time $\text{poly}(n, \log N)$. Unfortunately, this work did not provide any bound on N and the only known bound (4 times exponential in n) came directly from Makanin’s algorithm, together those two results yielded a 3NEXPTIME algorithm. Soon after the bound on the size of the shortest solution was improved to triply exponential [19], which immediately yielded an algorithm from class 2NEXPTIME, however, the same paper [19] improved Makanin’s algorithm, so that it worked in EXPSPACE.

Next, Plandowski gave a better (doubly exponential) bound on the size of the shortest solution [44] and thus obtained a NEXPTIME algorithm, in particular, at that time this was the best known algorithm for this problem. The proof was based on novel factorisations of words. By better exploiting the interplay between factorisations and compression, he improved the algorithm so that it worked in PSPACE [45].

It is worth mentioning, that the solution proposed by Plandowski is essentially different than the one given by Makanin. In particular, it allowed generalisations more easily: Diekert, Gutiérrez and Hagenah [8] showed, that Plandowski’s algorithm can be extended to the case

¹ A *Straight Line Programme* (SLP for short), is simply a context free grammar generating exactly one word.

in which we allow regular constraints in the equation (i.e. we want that the word substituted for X is from a regular language, whose description by a finite automaton is part of the input) and inversion; such an extended algorithm still works in polynomial space. It is easy to show that solving equations in free groups reduces to the above-mentioned problem of word equations with regular constraints and inversion [8] (it is worth mentioning, that in general we do not know whether solving equations in free groups is easier or harder than solving the ones in a free semigroup).

On the other hand, Plandowski showed, that his algorithm can be used to generate a finite representation of all solutions of a word equation [46], which allows solving several decision problems concerning the set of all solutions (finiteness, boundedness, boundedness of the exponent of periodicity etc.). It is not known, whether this algorithm can be generalised so that it generates all solutions also in the case of regular constraints and inversion (or in a free group).

The new, simpler algorithm for word equations and demonstration of connections between compression and word equations gave a new hope for solving the context unification problem. The first results were very promising: by using “tree” equivalents of SLPs computational complexity of some problems related to context unification was established [17, 31, 6]. Unfortunately, this approach failed to fully generalise Plandowski’s algorithm for words: the equivalent of factorisations that were used in the algorithm were not found for trees.

It is worth mentioning, that the approach proposed by Rytter and Plandowski, in which we compress a solution using SLPs (or in the non-deterministic case – we guess the compressed representation of the solution) and then perform the computation directly on the SLP-compressed representations using known algorithm that work in polynomial time, turned out to be extremely fruitful in many branches of computer science. The recent survey by Lohrey gives several such successful applications [35].

► **Remark.** As this is an informal survey presentations, most of the proofs are only sketched or omitted.

2 Recompression for word equations

We begin with a formal definition of the word equations problem: Consider a finite alphabet Σ and set of variables \mathcal{X} ; during the algorithm Σ will be extended by new letters, but it will always remain finite. Word equation is of a form “ $u = v$ ”, where $u, v \in (\Sigma \cup \mathcal{X})^*$ and its solution is a homomorphism $S : \Sigma \cup \mathcal{X} \mapsto \Sigma^*$, which is constant on Σ , that is $S(a) = a$, and satisfies the equation, i.e. words $S(u)$ and $S(v)$ are equal. By n we denote the size of the equation, i.e. $|u| + |v|$. The algorithm requires only small improvements so that it applies also to systems of equations, to streamline the presentation we will not consider this case.

Fix any solution S of the equation $u = v$, without loss of generality we can assume that this is the *shortest solution*, i.e. the one minimising $|S(u)|$; let N denote the *length of the solution*, that is $|S(u)|$. By the earlier work of Plandowski and Rytter [47] we know that $S(u)$ (and also $S(X)$ for each variable X) has an SLP (of size $\text{poly}(n, \log N)$), in fact the same conclusion can be drawn from the later works of Plandowski [44, 45, 46]. Regardless of the form of S and SLP, we know, that at least one of the productions in this SLP is of the form $c \rightarrow ab$, where c is a nonterminal of the SLP while $a, b \in \Sigma$ are letters. Let us “reverse” this production, i.e. replace in $S(u)$ all pairs of letters ab by c . It is relatively easy to formalise this operation for words, it is not so clear, what should be done in case of equations, so let us inspect the easier fragment first.

Algorithm 1 PairComp(ab, w) Compression of pair ab .

- 1: let $c \in \Sigma$ be an unused letter
 - 2: replace all occurrences of ab in w by c
-

Consider an explicitly given word w . Performing the “ ab -pair compression” on it is easy (we replace each pair ab by c), as long as $a \neq b$: replacing pairs aa is ambiguous, as such pairs can “overlap”. Instead, we replace *maximal blocks* of a letter a : block a^ℓ is *maximal*, when there is no letter a to left and to the right of it (in particular, there could be no letter at all).

Formally, the operations are defined as follows:

- *ab pair compression* For a given word w replace all occurrences of ab in w by a fresh letter c .
- *a block compression* For a given word w replace all occurrences of maximal blocks a^ℓ for $\ell > 1$ in w by fresh letters a_ℓ .

We always assume, that in the ab -pair compression the letters a and b are different.

Observe, that those operations are indeed “inverses” of SLP productions: replacing ab with c corresponds to a production $c \rightarrow ab$, similarly replacing a^ℓ with a_ℓ corresponds to a production $a_\ell \rightarrow a^\ell$.

Algorithm 2 BlockComp(a, w) Block compression for a .

- 1: **for** $\ell > 1$ **do**
 - 2: let $a_\ell \in \Sigma$ be an unused letter
 - 3: replace all maximal blocks a^ℓ in w by a_ℓ
-

Iterating the pair and blocks compression results in a compression of word w , assuming that we treat the introduced symbols as normal letters. There are several possible ways to implement such iteration, different results are obtained by altering the order of the compressions, exact treatment of new letters and so on. Still, essentially each “reasonable” variant works.

Observe, that if we compress two words, say w_1 and w_2 , in parallel then the resulting words w'_1 and w'_2 are equal if and only if w_1 and w_2 are. This justifies the usage of compression operations to both sides of the word equation in parallel, it remains to show, how to do that.

Let us fix a solution S , a pair ab (where $a \neq b$); consider how does a particular occurrence of ab got into $S(u)$.

► **Definition 1.** For an equation $u = v$, solution S and pair ab an occurrence of ab in $S(u)$ (or $S(v)$) is

- explicit, if it consists solely of letters coming from u (or v);
- implicit, if it consists solely of letters coming from a substitution $S(X)$ for a fixed occurrence of some variable X ;
- crossing, otherwise.

A pair ab is crossing (for a solution S) if it has at least one crossing occurrence and non-crossing (for a solution S) otherwise.

We similarly define explicit, implicit and crossing occurrences for blocks of letter a ; a is crossing, if at least one of its blocks has a crossing occurrence. (In other words: aa is crossing).

3:6 Solving Word Equations (And Other Unification Problems) by Recompression

► **Example 2.** Equation

$$aaXbbabababa = XaabbYabX$$

has a unique solution $S(X) = a$, $S(Y) = abab$, under which sides evaluate to

$$aaabbabababa = aaabbabababa .$$

Pair ba is crossing (as the first letter of $S(Y)$ is a and first Y is preceded by a letter b , moreover, the last letter of $S(Y)$ is b and the second Y is succeeded by a letter a), pair ab is non-crossing. Letter b is non-crossing, letter a is crossing (as X is preceded by a letter a on the left-hand side of the equation and on the right-hand side of the equation X is succeeded by a letter a).

■ **Algorithm 3** $\text{PairComp}(ab, 'u = v')$ Pair compression for ab in an equation $u = v$.

-
- 1: let $c \in \Sigma$ be a fresh letter
 - 2: replace all occurrences of ab in ' $u = v$ ' by c
-

■ **Algorithm 4** $\text{BlockComp}(a, 'u = v')$ Block compression for a letter a in an equation ' $u = v$ '.

-
- 1: **for** $\ell > 1$ **do**
 - 2: let $a_\ell \in \Sigma$ be a fresh letter
 - 3: replace all occurrences of maximal blocks a^ℓ in ' $u = v$ ' by a_ℓ
-

Fix a pair ab and a solution S of the equation $u = v$. If ab is non-crossing, performing $\text{PairComp}(ab, S(u))$ is easy: we need to replace every explicit occurrence (which we do directly on the equation) as well as each implicit occurrence, which is done “implicitly”, as the solution is not stored, nor written anywhere. Due to the similarities to PairComp we will simply use the name $\text{PairComp}(ab, 'u = v')$, when we make the pair compressions on the equation. The argument above shows, that if the equation had a solution for which ab is non-crossing then also the obtained equation has a solution. The same applies to the block compression, called $\text{BlockComp}(a, 'u = v')$ for simplicity. On the other hand, if the obtained equation has a solution, then also the original equation had one (this solution is obtained by replacing each letter c by ab , the argument for the block compressions the same).

► **Lemma 3.** *Let the equation $u = v$ have a solution S , such that ab is non-crossing for S . Then $u' = v'$ obtained by $\text{PairComp}(ab, 'u = v')$ is satisfiable.*

If the obtained equation $u' = v'$ is satisfiable, then also the original equation $u = v$ is.

The same applies to $\text{BlockComp}(a, 'u = v')$.

Unfortunately Lemma 3 is not enough to simulate $\text{Compression}(w)$ directly on the equation: In general there is no guarantee that the pair ab (letter a) is non-crossing, moreover, we do not know what are the pairs that have only implicit occurrences. It turns out, that the second problem is trivial: if we restrict ourselves to the shortest solutions then every pair that has an implicit occurrence has also a crossing or explicit one, a similar statement holds also for blocks of letters.

► **Lemma 4** ([47]). *Let S be a shortest solution of an equation ' $u = v$ '. Then:*

- *If ab is a substring of $S(u)$, where $a \neq b$, then a , b have explicit occurrences in the equation and ab has an explicit or crossing occurrence.*
- *If a^k is a maximal block in $S(u)$ then a has an explicit occurrence in the equation and a^k has an explicit or crossing occurrence.*

The proof is simple: suppose that a pair (block) has only implicit occurrences. Then we could remove them and the obtained solution is shorter, contradicting the assumption.

Getting back to the crossing pairs (and blocks), if we fix a pair ab (letter a), then it is easy to “uncross” it: by Definition 1 we can conclude that the pair ab is crossing if and only if for some variables X and Y (not necessarily different) one of the following conditions holds (we assume that the solution does not assign an empty word to any variable – otherwise we could simply remove such a variable from the equation):

- (CP1) aX occurs in the equation and $S(X)$ begins with b ;
- (CP2) Yb occurs in the equation and $S(Y)$ ends with a ;
- (CP3) YX occurs in the equation, $S(X)$ begins with b while $bS(Y)$ ends with a .

In each of these cases the “uncrossing” is natural: in (CP1) we “pop” from X a letter b to the left, in (CP2) we pop a to the right from Y , in (CP3) we perform both operations. It turns out that in fact we can be even more systematic: we do not have to look at the occurrences of variables, it is enough to consider the first and last letter of $S(X)$ for each variable X :

- If $S(X)$ begins with b then we replace X with bX (changing implicitly the solution $S(X) = bw$ to $S'(X) = w$), if in the new solution $S(X) = \epsilon$, i.e. it is empty, then we remove X from the equation;
- if $S(X)$ ends with a then we apply a symmetric procedure.

Such an algorithm is called **Pop**.

■ **Algorithm 5** $\text{Pop}(a, b, 'u = v')$.

```

1: for  $X$ : variable do
2:   if the first letter of  $S(X)$  is  $b$  then ▷ Guess
3:     replace every  $X$  w  $'u = v'$  by  $bX$ 
▷ Implicitly change solution  $S(X) = bw$  to  $S(X) = w$ 
4:     if  $S(X) = \epsilon$  then ▷ Guess
5:       remove  $X$  from  $u$  and  $v$ 
6:     ... ▷ Perform a symmetric operation for the last letter and  $a$ 

```

It is easy to see, that for appropriate non-deterministic choices the obtained equation has a solution for which ab is non-crossing: for instance, if aX occurs in the equation and $S(X)$ begins with b then we make the corresponding non-deterministic choices, popping b to the left and obtaining abX ; a simple proof requires a precise statement of the claim as well as some case analysis.

► **Lemma 5.** *If the equation $'u = v'$ has a solution S then for an appropriate run of $\text{Pop}(a, b, 'u = v')$ (for appropriate non-deterministic choices) the obtained equation $u' = v'$ has a corresponding solution S' , i.e. $S(u) = S'(u')$, for which ab is a non-crossing pair.*

If the obtained equation has a solution then also the original equation had one.

Thus, we know how to proceed with a crossing ab -pair compression: we first turn ab into a non-crossing pair (**Pop**) and then compress it as a non-crossing pair (**PairComp**).

We would like to perform similar operations also for block compression. For non-crossing blocks we can naturally define a similar algorithm $\text{BlockComp}(a, 'u = v')$. It remains to show how to “uncross” a letter a . Unfortunately, if aX occurs in the equation and $S(X)$ begins with a then replacing X with aX is not enough, as $S(X)$ may still begin with a . In such a case we iterate the procedure until the first letter of X is not a (this includes the case in

3:8 Solving Word Equations (And Other Unification Problems) by Recompression

which we remove the whole variable X). Observe, that instead of doing this letter by letter, we can uncross a in one step: it is enough to remove from variable X its whole a -prefix and a -suffix of $S(X)$ (if $w = a^\ell w' a^r$, where w' does not begin nor end with a , a -prefix w is a^ℓ and a -suffix is a^r ; if $w = a^\ell$ then a -suffix is empty). Such an algorithm is called **CutPrefSuff**.

■ **Algorithm 6** **CutPrefSuff**($a, 'u = v'$) Popping prefixes and suffixes.

```

1: for  $X$ : variable do
2:   guess the lengths  $\ell, r$  of  $a$ -prefix and suffix of  $S(X)$            ▷  $S(X) = a^\ell w a^r$ 
                                                                    ▷ If  $S(X) = a^\ell$  then  $r = 0$ 
3:   replace occurrences of  $X$  in  $u$  and  $v$  by  $a^\ell X a^r$    ▷  $a^\ell, a^r$  are stored in a compressed
   way
4:                                     ▷ Implicitly change the solution  $S(X) = a^\ell w b^r$  to  $S(X) = w$ 
5:   if  $S(X) = \epsilon$  then                                           ▷ Guess
6:     remove  $X$  from  $u$  and  $v$ 

```

Similarly as in **Pop** we can show that after an appropriate run of **CutPrefSuff** the obtained equation has a (corresponding) solution for which a is non-crossing. Unfortunately, there is another problem: we need to write down the lengths ℓ and r of a -prefixes and suffixes. We can write them as binary numbers, in which case they use $\mathcal{O}(\log \ell + \log r)$ bits of memory. However in general those still can be arbitrarily large numbers. Fortunately, we can show that in *some* solution those values are at most exponential (and so their description is polynomial-size). This easily follows from the exponential bound on exponent of periodicity [27]. For the moment it is enough that we know that:

► **Lemma 6** ([27]). *In the shortest solution of the equation ' $u = v$ ' each a -prefix and a -suffix has at most exponential length (in terms of $|u| + |v|$).*

Thus in **Pop** we can restrict ourselves to a -prefixes and suffixes of at most exponential length.

► **Lemma 7.** *Let S be a shortest solution of ' $u = v$ '. After some run of **CutPrefSuff**($a, 'u = v'$) (for appropriate non-deterministic choices) the obtained equation ' $u' = v'$ ' has a corresponding solution S' , such that $S'(u') = S(u)$, and a is a non-crossing letter for S' , moreover, the explicit a blocks in ' $u' = v'$ ' have at most exponential length.*

If the obtained equation has a solution then also the original equation had one.

After **Pop** we can compress a -blocks using **BlockComp**($a, 'u = v'$), observe that afterwards long a -blocks are replaced with single letters.

We are now ready to simulate **Compression** directly on the equation. The question is, in which order we should compress pairs and blocks? We make the choice nondeterministically: if there are any non-crossing pairs or letters, we compress them. This is natural, as such compression decreases both the size of the equation and the size of the length-minimal solution of the equation. If all pairs and letters are crossing, we choose greedily, i.e. the one that leads to the smallest equation (in one step). It is easy to show that such a strategy keeps the equation quadratic, more involved strategy, in which we compress many pairs/blocks in parallel, leads to a linear-length equation.

Call one iteration of the main loop a *phase*.

The correctness of the algorithm follows from the earlier discussion on the correctness of **BlockComp**, **CutPrefSuff**, **PairComp** and **Pop**. In particular, the length of the length-minimal solution drops by at least 1 in each iteration, thus the algorithm terminates.

■ **Algorithm 7** WordEqSAT Deciding the satisfiability of word equations.

```

1: while  $|u| > 1$  or  $|v| > 1$  do                                ▷ The equation is nontrivial
2:    $L \leftarrow$  list of letters in  $u, v$                             ▷ Occurring in the equation
3:   Choose a pair  $ab \in P^2$  or a letter  $a \in P$ 
4:   if it is crossing then
5:     uncross it
6:     compress it
7: Solve the problem naively                                ▷ The problem is simple when both sides have length 1

```

► **Lemma 8.** *Algorithm WordEqSAT has $\mathcal{O}(N)$ phases, where N is the length of the shortest solution of the input equation.*

Let us try to bound the space needed by the algorithm: we claim that for appropriate nondeterministic choices the stored equation has at most $8n^2$ letters (and n variables). To see this, observe first that each **Pop** introduces at most $2n$ letters, one at each side of the variable. The same applies to **CutPrefSuff** (formally, **CutPrefSuff** introduces long blocks but they are immediately replaced with single letters, and so we can think that in fact we introduce only $2n$ letters). By (CP1)–(CP3) we know that there are at most $2n$ crossing pairs and crossing letters (as each crossing pair / each crossing letter corresponds to one occurrence of a variable and one “side” of such an occurrence). If the equation has m letters (and at most n occurrences of variables) and there is an occurrence of a non-crossing pair or block then we choose it for compression. Otherwise, there are m letters in the equation and each is covered by at least one pair/block, so for one of $2n$ choice at least $\frac{m}{2n}$ letters is covered, so at least $\frac{m}{4n}$ letters are removed. Thus the new equation has at most

$$\underbrace{m}_{\text{previous}} - \underbrace{\frac{m}{4n}}_{\text{removed}} + \underbrace{2n}_{\text{popped}} \leq 8n^2 - 2n + 2n = 8n^2,$$

where the inequality follows by the inductive assumption that $m \leq 8n^2$. Going for the bit-size, each symbol requires at most logarithmic number of bits, and so

► **Lemma 9.** *WordEqSAT runs in $\mathcal{O}(n^2 \log n)$ space.*

With some effort we can make the above if analysis much tighter:

► **Theorem 10** ([24]). *The recompression based algorithm (nondeterministically) decides word equations problem in $\mathcal{O}(n \log n)$ bit-space; moreover, the stored equation has linear length.*

As a reminder: a PSPACE algorithm for this problem is already known [45]. Its memory consumption is not stated explicitly in that work, however, it is much larger than $\mathcal{O}(n \log n)$: the stored equations are of length $\mathcal{O}(n^3)$ and during the transformations the algorithm uses essentially more memory.

3 Similar applications

Generating a representation of all solutions

So far we have only considered the satisfiability of word equations. In general, there can be many solutions of such an equation and it is desirable to have a (finite) representation of

3:10 Solving Word Equations (And Other Unification Problems) by Recompression

all of them. The first such description was given by Plandowski [46], his algorithm works in PSPACE and generates an exponential representation of all solutions. We show that a similar description can be created using the recompression approach. It is easy to believe that the compression of pairs and blocks “preserves” the set of solutions: if S is a solution of an equation $u = v$ then we can compress the pairs (or blocks) in the word $S(u)$ and simulate such a compression directly on the equation $u = v$ obtaining $u' = v'$ with a “corresponding” solution $S'(u')$. In this way we naturally obtain a graph: its nodes are labelled with equations and an edge between equations $u = v$ and $u' = v'$ will describe how to transform the solutions of $u' = v'$ into solutions of $u = v$ (note that a node labelled with $u = v$ can have several edges to many other nodes). The mentioned description is fairly natural: we replace a letter c by a pair ab or replace a_ℓ with a^ℓ or prepend or append some letters to $S(X)$. It remains to guarantee that both the nodes and the edges have reasonable size description (in our case: polynomial).

Unfortunately, there is a problem: consider an equation $aXXXX = XaYY$, it has solutions of the form $S(X) = a^{\ell_X}$, $S(Y) = a^{\ell_Y}$, where additionally $4\ell_X + 1 = \ell_X + 1 + 2\ell_Y$. There are infinitely many such solutions and replacing X by a^{ℓ_X} during the CutPrefSuff can use arbitrarily large memory and transform this equation into an infinite number of other equations. On the other hand, as a next step we replace a -blocks of length $4\ell_X + 1 = \ell_X + 1 + 2\ell_Y$ by a new letter and the precise length of those blocks is unimportant, what matters is that they are of the same length. In general, we can improve the block compression so that it uses numerical parameters (for lengths) instead of concrete values of prefixes and suffixes. As a first step, in CutPrefSuff when we pop an a -prefix of length a^{ℓ_X} from X , the ℓ_X is not a number, but rather a numerical parameter, the same applies to the a -suffix r_X . Next we (non-deterministically) identify maximal blocks of the same length and verify, whether indeed such blocks can be of equal length. The guessed equalities correspond to a system of linear Diophantine equations. Moreover, each solution of such a system corresponds to a solution of the word equation and vice-versa. In this way we no longer need to consider large numbers and long equations and can guarantee that the considered equations are always of polynomial length (observe that this modification in fact removes the necessity of using the bound on the Σ -exponent of periodicity). Unfortunately, as a side effect we get that the edges in our graph representation of all solutions are labelled with systems of linear Diophantine equations and each solution of such a system corresponds to one transformation of the solution of the word equation.

► **Theorem 11** ([24]). *Using the recompression based algorithm we can compute (in PSPACE) a finite graph representation of all solutions of a word equation. Each node and edge have a polynomial description, the whole graph has at most exponential number of nodes and edges.*

As in the case of the decision variant, the recompression-based algorithm has much lower space consumption, than previously known [46], the same applies to the size of the constructed representation.

The above characterization is combinatorial in nature. On the other hand, it is natural to characterize the class of languages that can be obtained as sets of solutions of a word equation. For instance, the question of whether it is an indexed language, in the sense of Aho, was posed a long time ago. Using extensions of the recompression technique and interpreting it in the algebraic setting it can be shown that the language of all solutions of a given word equation is an EDTOL language [3] (so, in particular, an index language). This is by no means an easy task, in particular, the block compression needs to be redesigned essentially from scratch.

► **Theorem 12** ([3]). *The set of solutions of a given word equation is an EDT0L language.*

Equations with one variable

As already mentioned, one of the investigated subclasses of word equations are the equations with one variable. It is easy to show that they can be decided in $\mathcal{O}(n^3)$ and improving to quadratic running time requires only a couple of observations [7]. The first nontrivial algorithm for this problem had an $\mathcal{O}(n \log n)$ running time [42], while Dąbrowski and Plandowski gave an algorithm with $\mathcal{O}(n + \#_X \log n)$ running time [13], where $\#_X$ denotes the number of occurrences of a variable X in the original equation.

It is easy to see that the recompression based algorithm becomes deterministic in case of one variable equations: it makes the following non-deterministic choices:

- What is the first (last) letter of $S(X)$?
- What is the length of the a -prefix a^ℓ (suffix a^r) of $S(X)$?

When the equation has only one variable, answers to both of those questions can be easily obtained from the equation.

► **Lemma 13.** *Without loss of generality word equation with one variable are of the form*

$$A_0 X A_1 \dots A_{k-1} X A_k = X B_1 \dots B_{\ell-1} X B_\ell, \quad (1)$$

where A_0 is a non-empty word and exactly one of the words A_k, B_ℓ is empty.

Let the first letter of A_0 be a . Each solution $S(X) \notin a^+$ has the same a -prefix as A_0 ; symmetric fact holds also for a -suffixes.

Lemma 13 yields a simple recompression based algorithm: in each phase we identify candidate solutions from a^+ , where a is the first letter of A_0 , and verify whether indeed such a candidate is a solution. Next we perform recompression: all remaining solutions have the same a -prefix (and suffix).

A natural implementation of this algorithm has the same running time as the algorithm by Dąbrowski and Plandowski, i.e. $\mathcal{O}(n + \#_X \log n)$. It is possible to improve the running time to linear, this requires several non-trivial improvements of the algorithm and usage of efficient data structures (suffix arrays with a structure for computing the *longest common prefix*, i.e. lcp). In particular:

- Instead of one equation the algorithm actually stores a system of equations and sometimes splits one equation into two smaller ones, in this way we save space.
- We keep track, which words are the same and for set of identical words we store one copy and represent all of them by pointers.
- We prove that for a certain class of solutions the algorithm reports such solutions within $\mathcal{O}(1)$ phases. In many places of the proofs we show that the corresponding solution is from this class.
- We improve the testing procedure: some of the candidate solutions are rejected based on their structural properties, moreover we use a much more precise cost analysis: we calculate for each word separately, whether it took part in a particular test or not. In this way we can establish that some tests took less time than linear (which is the time needed for reading the whole equation).

► **Theorem 14** ([23]). *Using a recompression based algorithm we can in linear time return all solutions of a word equation with one variable.*

Equations with regular constraints and inversion; equations in free groups

As already mentioned, it is natural and important to extend the word equations by regular constraints and inversion, in particular this leads to an algorithm for equations in free groups [8] (the reduction between those two problems is fully syntactical and does not depend on the particular algorithm for solving word equations). Note that it is unknown, whether the algorithm generating a representation of all solutions can be also extended by regular constraints and inversion. Thus the only previously known algorithm for representation of all solutions of an equation in a free group was due to Razborov [48], and it was based on Makanin's algorithm for word equations in free groups.

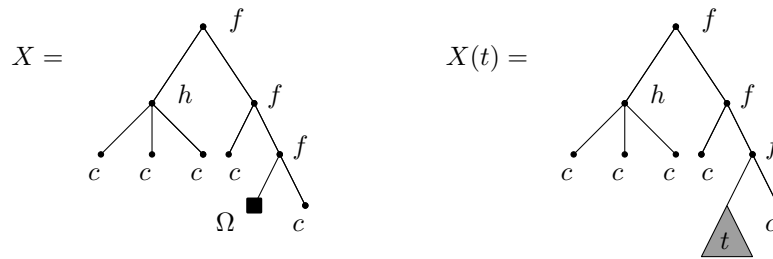
Adding the regular constraints to the recompression based algorithm WordEqSAT is fairly standard: We can encode all constraints using one non-deterministic finite automaton (the constraints for particular variables differ only in the set of accepting states). For each letter c we store its *transition function*, i.e. a function $f_c : Q \mapsto 2^Q$, which says that the automaton in state q after reading a letter c reaches a state in $f_c(q)$. This function is naturally extended to words: it still defines which states can be reached from q after reading w . Formally $f_{wa} = (f_w \circ f_a)(q) = \{p \mid \exists q' \in f_w(q) \text{ i } p \in f_a(q')\}$ for a letter a . If we introduce a new letter c (which replaces a word w) then we naturally define the transition function $f_c \leftarrow f_w$. We can express the regular constraints in terms of this function: saying that $S(X)$ is accepted by an automaton means that $f_{S(X)}(q_0)$ is one of the accepting states. So it is enough to guess the value of $f_{S(X)}$ which satisfies this condition; in this way we can talk about the value f_X for a variable X . Popping letters from a variable means that we need to adjust the transition function, i.e. when we replace X by aX then $f_X = f_a \circ f_{X'}$, we similarly define \bar{f}_X when we pop letters to the right.

More problems are caused by the *inversion*: intuitively it corresponds to taking the inverse element in the group and on the semigroup level we this is simulated by requiring that $\bar{\bar{a}} = a$ for each letter a and $\bar{a_1 a_2 \dots a_m} = \bar{a_m} \dots \bar{a_2} \bar{a_1}$. This has an impact on the compression: when we compress a pair ab to c , then we should also replace $\bar{ab} = \bar{b}\bar{a}$ by a letter \bar{c} . At the first sight this looks easy, but becomes problematic, when those two pairs are not disjoint, i.e. when $\bar{a} = a$ (or $\bar{b} = b$); in general we cannot exclude such a case and if it happens, in a sequence $ba\bar{b}$ during the pair compression for ba we want to simultaneously replace ba and $a\bar{b}$, which is not possible. Instead, we replace maximal fragments that can be fully covered with pairs ab or $\bar{b}\bar{a}$, in this case this: the whole triple $ba\bar{b}$. In the worst case (when $a = \bar{a}$ and $b = \bar{b}$) we need to replace whole sequences of the form $(ab)^n$, which is a common generalisation of both pairs and blocks compression.

As in the case of semigroups, this representation can be interpreted in the algebraic setting, which is even more natural, and can be used to show that the set of solutions is an EDTOL language.

► **Theorem 15** ([10], [3]). *A recompression based algorithm generates in polynomial space the description of all solutions of a word equation in free semigroups with inversion and regular constraints. This in particular provides a similar description in case of free groups with regular constraints and shows that the set of solutions is an EDTOL language.*

Trace equations. For our purposes it is better to view partially commuting words, i.e. traces, in terms of resources of letters: we equip letters of the alphabet Σ with resources, formally there is a function $\rho : \Sigma \rightarrow R$, where R is some finite set. Then two different letters commute if and only if they do not share a resource, i.e. $ab = ba$ for $a \neq b$ if and only if $\rho(a) \cap \rho(b) = \emptyset$. It is easy to see the equivalence of words with resources and traces.



■ **Figure 1** A context and the same context applied on an argument.

It is difficult to apply compression operations in trace equation to letters of different resources. On the other hand, when the set of resources of some letters are the same, they behave exactly like ordinary non-commuting words and the recompression approach can be applied to them. A natural approach to solving trace equation using recompression involves also another operation of “lifting” letters, i.e. increasing the set of resources of a letter. In this way the trace is partially “linearized”, as part of the commutation is removed.

It turns out that this approach can be implemented, and the algorithm alternates the lifting and compression operations, which is in contrast to previous approaches to trace equations, which linearized the trace once at the beginning. In particular, the results concerning the involution, regular constraints and equations in the corresponding groups, which are the well-known Right-angled Artin groups, also generalize to traces. The details are rather technical and are beyond the scope of a survey aimed at presenting recompression technique.

► **Theorem 16** ([9]). *The set of solutions of trace equation (with involution and regular constraints) is an EDTOL, its nonemptiness can be decided in PSPACE.*

The same results holds also for Right-angled Artin Groups.

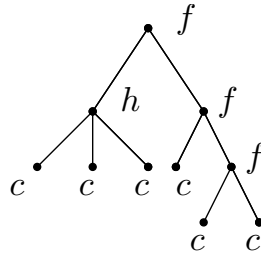
Context unification

Recall that the context unification is a generalisation of word equations to the case of terms. What type of equations we would like to consider? Clearly we consider terms over a fixed signature (which is usually part of the input), and allow occurrences of constants and variables. If we allow only that the variables represent full terms, then the satisfiability of such equations is decidable in polynomial time [49] and so probably does not generalise the word equations (which are NP-hard). This is also easy to observe when we look closer at a word equation: the words represented by the variables can be concatenated at both ends, i.e. they represent terms with a missing argument.

We arrive at a conclusion that our generalisation should use variables *with arguments*, i.e. the (second-order) variables take an argument that is a full term and can use it, perhaps several times. Such a definition leads to a *second-order unification*, which is known to be undecidable even in very restricted subcases [18, 14, 30, 32].

Thus we would like to have a subclass of second order unification that still generalises word equations. In order to do that we put additional restriction on the solutions: each argument can be used by the term *exactly once*. Observe that this still generalise the word equations: using the argument exactly once naturally corresponds to concatenation.

Formally, in the context unification problem [4, 5, 51], we consider an equation $u = v$ in which we use variables (representing closed terms), which we denote by letters x, y , as well as context variables (representing terms with one “hole” for the argument, they are usually



■ **Figure 2** Term $f(h(c, c, c), f(c, f(c, c)))$ represented as a tree, f is of arity 2, h arity 3, while c : 0.

called *contexts*), which we denote by letters X, Y . Syntactically, u and v are terms that use letters from signature Σ (which is part of the input), variables and context variables, the former are treated as symbols of arity 0, while the latter as symbols of arity 1. A *substitution* S assigns to each variable a closed term over Σ and to each context variable it assigns a *context*, i.e. a term over $\Sigma \cup \{\Omega\}$ in which the special symbol Ω has arity 0 and is used exactly once. (Intuitively it corresponds to a place in which we later substitute the argument). S is extended to u, v in a natural way, note that for a context variable X the term $S(X(t))$ is obtained by replacing in $S(X)$ the unique symbol Ω by $S(t)$. A *solution* is a substitution satisfying $S(u) = S(v)$.

► **Example 17.** Consider a signature $\{f, c, c'\}$, where f has arity 2 while c, c' have arity 0 and consider an equation $X(c) = Y(c')$, where X and Y are context variables. The equation has a solution $S(X) = f(\Omega, c')$, $S(Y) = f(c, \Omega)$ and then $S(X(c)) = f(c, c') = S(Y(c'))$.

We try to apply the main idea of the recompression also in the case of terms: we iterate local compression operations and we guarantee that the word (term) equation is polynomial size. Since several term problems were solved using compression-based methods [17, 31, 6, 15, 16], there is a reasonable hope that our approach may succeed.

Pair and block compression easily generalise to sequences of letters of arity 1 (we can think of them as words), unfortunately, there is no guarantee that a term has even one such letter. Intuitively, we rather expect that it has mostly leaves and symbols of larger arity. This leads us to another local compression operation: *leaf compression*. Consider a node labelled with f and its i -th child that is a leaf. We want to compress f with this child, leaving other children (and their subtrees) unchanged. Formally, given f of arity at least 1, position $1 \leq i \leq \text{ar}(f)$ and a letter c of arity 0 the $\text{LeafComp}(f, i, c, t)$ operation (*leaf compression*) replaces in term t nodes labelled with f and subterms $t_1, \dots, t_{i-1}, c, t_{i+1}, \dots, t_{\text{ar}(f)}$ (where c and position i are fixed, while other terms $t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_{\text{ar}(f)}$ – varying) by a term labelled with f' and subterms $t'_1, \dots, t'_{i-1}, t'_{i+1}, \dots, t'_{\text{ar}(f)}$ that are obtained by applying recursively LeafComp to terms $t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_{\text{ar}(f)}$; in other words, we first change the label from f to f' and then remove the i -th child, which has a label c and we apply such a compression to all occurrences of f and c in parallel.

The notion of crossing pair generalizes to this case in a natural way and the uncrossing replaces a term variable with a constant or replaces $X(t)$ with $X(f(x_1, \dots, x_i, t, x_{i+1}, \dots, x_\ell))$. Note that this introduces new variables.

Now the whole algorithm looks similar as in the case of word equations, we simply use additional compression operation. However, the analysis is much more involved, as the new uncrossing introduces fresh term variables. However, their number at any point can be linearly bounded and the polynomial upper-bound follows.

► **Theorem 18** ([22]). *Recompression based algorithm solves context unification in non-deterministic polynomial space.*

References

- 1 Julius Richard Büchi and Steven Senger. Definability in the existential theory of concatenation and undecidable extensions of this theory. *Mathematical Logic Quarterly*, 34(4):337–342, 1988. doi:10.1002/malq.19880340410.
- 2 Witold Charatonik and Leszek Pacholski. Word equations with two variables. In *IWWERT*, pages 43–56, 1991. doi:10.1007/3-540-56730-5_30.
- 3 Laura Ciobanu, Volker Diekert, and Murray Elder. Solution sets for equations over free groups are EDT0L languages. *IJAC*, 26(5):843–886, 2016. doi:10.1142/S0218196716500363.
- 4 Hubert Comon. Completion of rewrite systems with membership constraints. Part I: Deduction rules. *J. Symb. Comput.*, 25(4):397–419, 1998. doi:10.1006/jsc.1997.0185.
- 5 Hubert Comon. Completion of rewrite systems with membership constraints. Part II: Constraint solving. *J. Symb. Comput.*, 25(4):421–453, 1998. doi:10.1006/jsc.1997.0186.
- 6 Carles Creus, Adria Gascón, and Guillem Godoy. One-context unification with STG-compressed terms is in NP. In Ashish Tiwari, editor, *23rd International Conference on Rewriting Techniques and Applications (RTA'12)*, volume 15 of *LIPICs*, pages 149–164, Dagstuhl, Germany, 2012. Schloss Dagstuhl — Leibniz Zentrum fuer Informatik. doi:10.4230/LIPICs.RTA.2012.149.
- 7 Volker Diekert. Makanin’s Algorithm. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, chapter 12, pages 342–390. Cambridge University Press, 2002.
- 8 Volker Diekert, Claudio Gutiérrez, and Christian Hagenah. The existential theory of equations with rational constraints in free groups is PSPACE-complete. *Inf. Comput.*, 202(2):105–140, 2005. doi:10.1016/j.ic.2005.04.002.
- 9 Volker Diekert, Artur Jež, and Manfred Kufleitner. Solutions of word equations over partially commutative structures. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *ICALP*, volume 55 of *LIPICs*, pages 127:1–127:14. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPICs.ICALP.2016.127.
- 10 Volker Diekert, Artur Jež, and Wojciech Plandowski. Finding all solutions of equations in free groups and monoids with involution. *Inf. Comput.*, 251:263–286, 2016. doi:10.1016/j.ic.2016.09.009.
- 11 Volker Diekert and Anca Muscholl. Solvability of equations in free partially commutative groups is decidable. *International Journal of Algebra and Computation*, 16:1047–1070, 2006. Conference version in Proc. ICALP 2001, 543–554, LNCS 2076. doi:10.1142/S0218196706003372.
- 12 Robert Dąbrowski and Wojciech Plandowski. Solving two-variable word equations. In *ICALP*, pages 408–419, 2004. doi:10.1007/978-3-540-27836-8_36.
- 13 Robert Dąbrowski and Wojciech Plandowski. On word equations in one variable. *Algorithmica*, 60(4):819–828, 2011. doi:10.1007/s00453-009-9375-3.
- 14 William M. Farmer. Simple second-order languages for which unification is undecidable. *Theor. Comput. Sci.*, 87(1):25–41, 1991. doi:10.1016/S0304-3975(06)80003-4.
- 15 Adria Gascón, Guillem Godoy, and Manfred Schmidt-Schauß. Context matching for compressed terms. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 93–102. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.17.
- 16 Adria Gascón, Guillem Godoy, and Manfred Schmidt-Schauß. Unification and matching on compressed terms. *ACM Trans. Comput. Log.*, 12(4):26, 2011. doi:10.1145/1970398.1970402.
- 17 Adria Gascón, Guillem Godoy, Manfred Schmidt-Schauß, and Ashish Tiwari. Context unification with one context variable. *J. Symb. Comput.*, 45(2):173–193, 2010. doi:10.1016/j.jsc.2008.10.005.
- 18 Warren D. Goldfarb. The undecidability of the second-order unification problem. *Theor. Comput. Sci.*, 13:225–230, 1981. doi:10.1016/0304-3975(81)90040-2.
- 19 Claudio Gutiérrez. Satisfiability of word equations with constants is in exponential space. In *FOCS*, pages 112–119, 1998. doi:10.1109/SFCS.1998.743434.
- 20 Lucian Ilie and Wojciech Plandowski. Two-variable word equations. *ITA*, 34(6):467–501, 2000. doi:10.1051/ita:2000126.

- 21 Joxan Jaffar. Minimal and complete word unification. *J. ACM*, 37(1):47–85, 1990.
- 22 Artur Jež. Context unification is in PSPACE. In Elias Koutsoupias, Javier Esparza, and Pierre Fraigniaud, editors, *ICALP*, volume 8573 of *LNCS*, pages 244–255. Springer, 2014. full version at <http://arxiv.org/abs/1310.4367>. doi:10.1007/978-3-662-43951-7_21.
- 23 Artur Jež. One-variable word equations in linear time. *Algorithmica*, 74:1–48, 2016. doi:10.1007/s00453-014-9931-3.
- 24 Artur Jež. Recompression: a simple and powerful technique for word equations. *J. ACM*, 63(1):4:1–4:51, March 2016. doi:10.1145/2743014.
- 25 Olga Kharlampovich and Alexei Myasnikov. Irreducible affine varieties over a free group. ii: Systems in triangular quasi-quadratic form and description of residually free groups. *Journal of Algebra*, 200:517–570, 1998.
- 26 Olga Kharlampovich and Alexei Myasnikov. Elementary theory of free non-abelian groups. *Journal of Algebra*, 302:451–552, 2006.
- 27 Antoni Kościelski and Leszek Pacholski. Complexity of Makanin’s algorithm. *J. ACM*, 43(4):670–684, 1996. doi:10.1145/234533.234543.
- 28 Antoni Kościelski and Leszek Pacholski. Makanin’s algorithm is not primitive recursive. *Theor. Comput. Sci.*, 191(1-2):145–156, 1998. doi:10.1016/S0304-3975(96)00321-0.
- 29 Markku Laine and Wojciech Plandowski. Word equations with one unknown. *Int. J. Found. Comput. Sci.*, 22(2):345–375, 2011. doi:10.1142/S0129054111008088.
- 30 Jordi Levy. Linear second-order unification. In Harald Ganzinger, editor, *RTA*, volume 1103 of *LNCS*, pages 332–346. Springer, 1996. doi:10.1007/3-540-61464-8_63.
- 31 Jordi Levy, Manfred Schmidt-Schauß, and Mateu Villaret. On the complexity of bounded second-order unification and stratified context unification. *Logic Journal of the IGPL*, 19(6):763–789, 2011. doi:10.1093/jigpal/jzq010.
- 32 Jordi Levy and Margus Veanes. On the undecidability of second-order unification. *Inf. Comput.*, 159(1–2):125–150, 2000. doi:10.1006/inco.2000.2877.
- 33 Jordi Levy and Mateu Villaret. Linear second-order unification and context unification with tree-regular constraints. In Leo Bachmair, editor, *RTA*, volume 1833 of *LNCS*, pages 156–171. Springer, 2000. doi:10.1007/10721975_11.
- 34 Jordi Levy and Mateu Villaret. Curryng second-order unification problems. In Sophie Tison, editor, *RTA*, volume 2378 of *LNCS*, pages 326–339. Springer, 2002. doi:10.1007/3-540-45610-4_23.
- 35 Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.
- 36 Gennadiĭ Makanin. The problem of solvability of equations in a free semigroup. *Matematicheskii Sbornik*, 2(103):147–236, 1977. (in Russian).
- 37 Gennadiĭ Makanin. Equations in a free group. *Izv. Akad. Nauk SSR, Ser. Math.* 46:1199–1273, 1983. English transl. in *Math. USSR Izv.* 21 (1983).
- 38 Gennadiĭ Makanin. Decidability of the universal and positive theories of a free group. *Izv. Akad. Nauk SSSR, Ser. Mat.* 48:735–749, 1984. In Russian; English translation in: *Math. USSR Izvestija*, 25, 75–88, 1985.
- 39 Yuri Matiyasevich. Some decision problems for traces. In Sergej Adian and Anil Nerode, editors, *LFCS*, volume 1234 of *LNCS*, pages 248–257. Springer, 1997. Invited lecture.
- 40 Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. On equality up-to constraints over finite trees, context unification, and one-step rewriting. In William McCune, editor, *CADE*, volume 1249 of *LNCS*, pages 34–48. Springer, 1997. doi:10.1007/3-540-63104-6_4.
- 41 Joachim Niehren, Manfred Pinkal, and Peter Ruhrberg. A uniform approach to underspecification and parallelism. In Philip R. Cohen and Wolfgang Wahlster, editors, *ACL*, pages 410–417. Morgan Kaufmann Publishers / ACL, 1997. doi:10.3115/979617.979670.
- 42 S. Eyono Obono, Pavel Goralcik, and M. N. Maksimenko. Efficient solving of the word equations in one variable. In *MFCS*, pages 336–341, 1994. doi:10.1007/3-540-58338-6_80.

- 43 Wojciech Plandowski. Testing equivalence of morphisms on context-free languages. In Jan van Leeuwen, editor, *ESA*, volume 855 of *LNCS*, pages 460–470. Springer, 1994. doi:10.1007/BFb0049431.
- 44 Wojciech Plandowski. Satisfiability of word equations with constants is in NEXPTIME. In *STOC*, pages 721–725. ACM, 1999. doi:10.1145/301250.301443.
- 45 Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. *J. ACM*, 51(3):483–496, 2004. doi:10.1145/990308.990312.
- 46 Wojciech Plandowski. An efficient algorithm for solving word equations. In Jon M. Kleinberg, editor, *STOC*, pages 467–476. ACM, 2006. doi:10.1145/1132516.1132584.
- 47 Wojciech Plandowski and Wojciech Rytter. Application of Lempel-Ziv encodings to the solution of word equations. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *ICALP*, volume 1443 of *LNCS*, pages 731–742. Springer, 1998. doi:10.1007/BFb0055097.
- 48 Alexander A. Razborov. *On Systems of Equations in Free Groups*. PhD thesis, Steklov Institute of Mathematics, 1987. In Russian.
- 49 John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.
- 50 Aleksa Saarela. On the complexity of Hmelevskii’s theorem and satisfiability of three unknown equations. In Volker Diekert and Dirk Nowotka, editors, *Developments in Language Theory*, volume 5583 of *LNCS*, pages 443–453. Springer, 2009. doi:10.1007/978-3-642-02737-6_36.
- 51 Manfred Schmidt-Schauß. Unification of stratified second-order terms. Internal Report 12/94, Johann-Wolfgang-Goethe-Universität, 1994.
- 52 Manfred Schmidt-Schauß. A decision algorithm for stratified context unification. *J. Log. Comput.*, 12(6):929–953, 2002. doi:10.1093/logcom/12.6.929.
- 53 Manfred Schmidt-Schauß. Decidability of bounded second order unification. *Inf. Comput.*, 188(2):143–178, 2004. doi:10.1016/j.ic.2003.08.002.
- 54 Manfred Schmidt-Schauß and Klaus U. Schulz. On the exponent of periodicity of minimal solutions of context equation. In *RTA*, volume 1379 of *LNCS*, pages 61–75. Springer, 1998. doi:10.1007/BFb0052361.
- 55 Manfred Schmidt-Schauß and Klaus U. Schulz. Solvability of context equations with two context variables is decidable. *J. Symb. Comput.*, 33(1):77–122, 2002. doi:10.1006/j.sco.2001.0438.
- 56 Klaus U. Schulz. Makanin’s algorithm for word equations—two improvements and a generalization. In Klaus U. Schulz, editor, *IWWERT*, volume 572 of *LNCS*, pages 85–150. Springer, 1990. doi:10.1007/3-540-55124-7_4.


Strong Bisimulation for Control Operators

Delia Kesner

IRIF, Université de Paris and CNRS, France
Institut Universitaire de France (IUF), France

Eduardo Bonelli

Stevens Institute of Technology, Hoboken, NJ, USA

Andrés Viso 

Universidad de Buenos Aires, Argentina
Universidad Nacional de Quilmes, Bernal, Buenos Aires, Argentina

Abstract

The purpose of this paper is to identify programs with control operators whose reduction semantics are in exact correspondence. This is achieved by introducing a relation \simeq , defined over a revised presentation of Parigot’s $\lambda\mu$ -calculus we dub ΛM .

Our result builds on two fundamental ingredients: (1) factorization of $\lambda\mu$ -reduction into multiplicative and exponential steps by means of explicit term operators of ΛM , and (2) translation of ΛM -terms into Laurent’s polarized proof-nets (PPN) such that cut-elimination in PPN simulates our calculus. Our proposed relation \simeq is shown to characterize structural equivalence in PPN. Most notably, \simeq is shown to be a strong bisimulation with respect to reduction in ΛM , i.e. two \simeq -equivalent terms have the exact same reduction semantics, a result which fails for Regnier’s σ -equivalence in λ -calculus as well as for Laurent’s σ -equivalence in $\lambda\mu$.

2012 ACM Subject Classification Theory of computation \rightarrow Lambda calculus; Theory of computation \rightarrow Linear logic; Theory of computation \rightarrow Operational semantics

Keywords and phrases Lambda-mu calculus, proof-nets, strong bisimulation

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.4

Category Invited Talk

Related Version A full version of the paper is available at <https://arxiv.org/abs/1906.09370>.

Funding This work was partially supported by LIA INFINIS, and the ECOS-Sud program PA17C01.

Acknowledgements To Olivier Laurent for fruitful discussions.

1 Introduction

An important topic in the study of programming language theories is unveiling structural similarities between expressions denoting programs. They are widely known as *structural equivalences*; equivalent expressions behaving exactly in the same way. Process calculi are a rich source of examples. In CCS expressions stand for processes in a concurrent system. For example, $P \parallel Q$ denotes the parallel composition of processes P and Q . Structural equivalence includes equations such as the one stating that $P \parallel Q$ and $Q \parallel P$ are equivalent. This minor reshuffling of subexpressions has little impact on the behavior of the overall expression: structural equivalence is a *strong bisimulation* for process reduction. This paper

$$\begin{array}{ccc} o & \simeq & p \\ \downarrow & & \downarrow \\ \downarrow & & \downarrow \\ o' & \simeq & p' \end{array}$$

is concerned with such notions of reshuffling of expressions in λ -calculus with control operators.



© Delia Kesner, Eduardo Bonelli, and Andrés Viso;
licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 4; pp. 4:1–4:23



Leibniz International Proceedings in Informatics

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

4:2 Strong Bisimulation for Control Operators

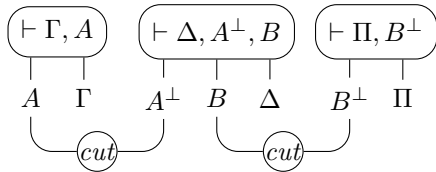
The induced notion of structural equivalence, in the sequel \simeq , should identify terms having exactly the same reduction semantics too, that is, should be a strong bisimulation with respect to reduction in these calculi. In other words, \simeq should be symmetric and moreover $o \simeq p$ and $o \rightsquigarrow o'$ should imply the existence of p' such that $p \rightsquigarrow p'$ and $o' \simeq p'$, where \rightsquigarrow denotes some given notion of reduction for control operators (see figure on the right).

Formulating such structural equivalences for the λ -calculus is hindered by the sequential (left-to-right) orientation in which expressions are written. Consider for example the terms $(\lambda x.(\lambda y.t) u) v$ and $(\lambda x.\lambda y.t) v u$. They seem to have the same redexes, only permuted, similar to the situation captured by the above mentioned CCS equation. A closer look, however, reveals that this is not entirely correct. The former has two redexes (one indicated below by underlining and another by overlining) and the latter has only one (underlined):

$$(\lambda x.\overline{(\lambda y.t) u}) v \text{ and } (\lambda x.\underline{(\lambda y.t)}) v u \quad (1)$$

The overlined redex on the left-hand side is not visible on the right-hand side; it will only reappear, as a newly *created* redex, once the underlined redex is computed. Despite the fact that the syntax gets in the way, Regnier [27] proved that these terms behave in *essentially* the same way. More precisely, he introduced a structural equivalence for λ -terms, known as σ -equivalence and he proved that σ -equivalent terms have head, leftmost, perpetual and, more generally, maximal reductions of the same length. However, the mismatch between the terms in (1) is unsatisfying since there clearly seems to be an underlying strong bisimulation, which is not showing itself due to a notational shortcoming. It turns out that through the graphical intuition provided by linear logic *proof-nets*, one can define an enriched λ -calculus that unveils a strong bisimulation for the intuitionistic case [4]. Further details are described below. In this paper, we resort to this same intuition to explore whether it is possible to uncover a strong bisimulation behind a notion of structural equivalence for the more challenging setting of classical logic. Thus, we will not only capture structural equivalence on pure functions, but also on *programs with control operators*. In our case it is polarized proof-nets (PPN) that will serve as semantic yardstick. We next briefly revisit proof-nets and discuss how they help unveil structural equivalence as strong bisimulation for λ -calculi. An explanation of the challenges that we face in addressing the classical case will follow.

Proof-Nets. A *proof-net* is a graph-like structure whose nodes denote logical inferences and whose edges or wires denote the formula they operate on (cf. Sec. 6). Proof-nets were introduced in the setting of linear logic [12], which provides a mechanism to explicitly control the use of resources by restricting the application of the *structural* rules of weakening and contraction. Proof-nets are equipped with an operational semantics specified by graph transformation rules which captures cut elimination in sequent calculus. The resulting cut elimination rules on proof-nets are split into two different kinds: *multiplicative*, that essentially (linearly) reconfigure wires, and *exponential*, which are the only ones that are able to erase or duplicate (sub)proof-nets. The latter are considered to introduce interesting or *meaningful* computation. Most notably, proof-nets abstract away the order in which certain rules occur in a sequent derivation. As an example, assume three derivations of the judgements $\vdash \Gamma, A$, $\vdash \Delta, A^\perp, B$ and $\vdash \Pi, B^\perp$, resp. The order in which these derivations are composed via cuts into a single derivation is abstracted away in the resulting proof-net:



In other words, *different* terms/derivations are represented by the *same* proof-net. Hidden structural similarity between terms can thus be studied by translating them to proof-nets. Moreover, following the Curry-Howard isomorphism which relates computation and logic, this correspondence can be extended not only to terms themselves [10, 8, 18, 5] but also to their reduction behavior [2]. In this paper, however, we concentrate on identifying those different classical derivations which translate to the same graph representation. As is standard in the literature, the notion of proof-net identity we adopt includes simple equalities such as *associativity* of contraction nodes and other similar rewirings (cf. notion of *structural equivalence* of proof-nets).

Intuitionistic σ -Equivalence. As mentioned before, Regnier introduced a notion of σ -equivalence on λ -terms (written here \simeq_σ and depicted in Fig. 1), and proved that σ -equivalent terms behave in essentially identical way. This equivalence relation involves permuting certain redexes, and was unveiled through the study of proof-nets. In particular, following Girard's encoding of intuitionistic into linear logic [12], σ -equivalent terms are mapped to the same proof-net (modulo multiplicative cuts and structural equivalence).

$$\begin{aligned} (\lambda x.\lambda y.t) u &\simeq_{\sigma_1} \lambda y.(\lambda x.t) u & y \notin u \\ (\lambda x.t v) u &\simeq_{\sigma_2} (\lambda x.t) u v & x \notin v \end{aligned}$$

■ **Figure 1** Regnier's σ -equivalence for λ -terms.

The reason why Regnier's result is not immediate is that redexes present on one side of an equation may disappear on the other side of it, as illustrated in the terms in (1). One might rephrase this observation by stating that \simeq_σ is *not a strong bisimulation* over the set of λ -terms. If it were, then establishing that σ -equivalent terms behave essentially in the same way would be trivial.

Adopting a more refined view of λ -calculus, as suggested by linear logic, which splits cut elimination on logical derivations into multiplicative and exponential steps yields a decomposition of β -reduction into multiplicative/exponential steps *on terms*. The theory of *explicit substitutions* (a survey can be found in [17]) provides a convenient syntax to reflect these steps at the term level. Indeed, β -reduction can be decomposed into two steps, namely B (for Beta), which acts *at a distance* [5] in the sense that the abstraction and the argument may be separated by an arbitrary number of explicit substitutions, and S (for Substitution):

$$\begin{aligned} (\lambda x.t)[x_1 \setminus v_1] \dots [x_n \setminus v_n] u &\mapsto_B t[x \setminus u][x_1 \setminus v_1] \dots [x_n \setminus v_n] \\ t[x \setminus u] &\mapsto_S t\{x \setminus u\} \end{aligned} \quad (2)$$

Firing the B-rule creates an *explicit substitution* operator, written $t[x \setminus u]$, so that B essentially reconfigures symbols, and indeed reads as a multiplicative cut in proof-nets. The S-rule executes the substitution by performing a replacement of all free occurrences of x in t with u , written $t\{x \setminus u\}$, so that it is S that performs interesting or *meaningful* computation and reads as an exponential cut in proof-nets.

A term without any occurrence of the left-hand side of rule B is called a B-normal form; we shall refer to these terms as *canonical forms*. Decomposition of β -reduction by means of the rules in (2) prompts one to replace \simeq_σ (Fig. 1) with a new relation \simeq_{σ_B} (Fig. 2). The latter is formed essentially by taking the B-normal form of each side of the \simeq_σ equations¹.

¹ Also included in \simeq_{σ_B} is equation \simeq_{σ_3} allowing commutation of orthogonal (independent) substitutions.

4:4 Strong Bisimulation for Control Operators

$$\begin{aligned}
(\lambda y.t)[x \setminus u] &\simeq_{\sigma_1^B} \lambda y.t[x \setminus u] & y \notin u \\
(t v)[x \setminus u] &\simeq_{\sigma_2^B} t[x \setminus u] v & x \notin v \\
t[y \setminus v][x \setminus u] &\simeq_{\sigma_3^B} t[x \setminus u][y \setminus v] & y \notin u, x \notin v
\end{aligned}$$

■ **Figure 2** Strong bisimulation for λ -terms with explicit substitutions.

Since B-reduction corresponds only to multiplicative cuts in proof-nets, the translation of \simeq_{σ^B} -equivalent typed terms also yields *structurally equivalent* proof-nets. In other words, \simeq_{σ^B} -equivalence classes of λ -terms with explicit substitutions in B-normal form are in one-to-one correspondence with intuitionistic linear logic proof-nets [5]. Moreover, \simeq_{σ^B} is a strong bisimulation with respect to meaningful reduction (i.e. S-reduction) over the extended set of terms that includes explicit substitutions [5, 4]. Indeed, \simeq_{σ^B} is symmetric, and moreover, $u \simeq_{\sigma^B} v$ and $u \rightarrow_S u'$ implies the existence of v' such that $v \rightarrow_S v'$ and $u' \simeq_{\sigma^B} v'$. Note also that the B-normal form of both sides of (1) are \simeq_{σ^B} -equivalent, thus repairing the mismatch.

Classical σ -Equivalence. This work sets out to explore structural equivalence for λ -calculi with control operators. These calculi include operations to manipulate the context in which a program is executed. We focus here on Parigot’s $\lambda\mu$ -calculus [25], which extends the λ -calculus with two new operations: $[\alpha] t$ (*named term*) and $\mu\alpha.c$ (μ -*abstraction*). The former may be read as “call continuation α with t as argument” and the latter as “record the current continuation as α and continue as c ”. Reduction in $\lambda\mu$ consists of the β -rule together with:

$$(\mu\alpha.c) u \mapsto_{\mu} \mu\alpha.c\{\{\alpha \setminus u\}\}$$

where $c\{\{\alpha \setminus u\}\}$, called here *replacement*, replaces all subexpressions of the form $[\alpha] t$ in c with $[\alpha](t u)$. Regnier’s notion of σ -equivalence for λ -terms was extended to $\lambda\mu$ by Laurent [23] (cf. Fig. 4 in Sec. 4). Here is an example of terms related by this extension, where the redexes are underlined/overlined:

$$(\underline{(\lambda x.\mu\alpha.[\gamma] u) w}) v \simeq_{\sigma} \overline{(\mu\alpha.[\gamma] (\underline{\lambda x.u}) w)} v$$

Once again, the fact that a harmless permutation of redexes has taken place is not obvious. The term on the right has two redexes (μ and β) but the one on the left only has one (β) redex. Another, more subtle, example of terms related by Laurent’s extension clearly suggests that operational indistinguishability cannot rely on relating arbitrary μ -redexes; the underlined μ -redex on the left does not appear at all on the right:

$$(\underline{\mu\alpha.[\alpha] x}) y \simeq_{\sigma} x y \tag{3}$$

Clearly, σ -equivalence on $\lambda\mu$ -terms *fails to be a strong bisimulation*. Nonetheless, Laurent proved properties for \simeq_{σ} in $\lambda\mu$ similar to those of Regnier for \simeq_{σ} in λ . Again, one has the feeling that there is a strong bisimulation hiding behind σ -equivalence for $\lambda\mu$.

Towards a Strong Bisimulation for Control Operators. We seek to formulate a notion of equivalence for $\lambda\mu$ in the sense that it is concerned with harmless permutation of redexes possibly involving control operators and inducing a strong bisimulation. As per the Curry-Howard isomorphism, proof normalization in classical logic corresponds to computation in

Notice however that the B-expansion of $\simeq_{\sigma_3^B}$ -equivalent terms yields \simeq_{σ} -equivalent terms again. For example, the B-expansion of $t[y \setminus v][x \setminus u] \simeq_{\sigma_3^B} t[x \setminus u][y \setminus v]$ yields $(\lambda y.(\lambda x.t) u) v \simeq_{\sigma_1, \sigma_2} (\lambda x.(\lambda y.t) v) u$.

$$\begin{array}{ccc}
\mu\alpha'.([\alpha]x)[\alpha\backslash_{\alpha'}y] & \simeq & xy \\
\downarrow \mathbf{R} & & \downarrow \mathbf{R} \\
\mu\alpha'.[\alpha']xy & \simeq & xy
\end{array}$$

■ **Figure 3** Failure of strong bisimulation.

λ -calculi with control operators [13, 25]. Moreover, since classical logic can be translated into *polarized proof-nets* (PPN), as defined by O. Laurent [22, 23], we use PPNs to guide the development in this work. A first step towards our goal involves decomposing the μ -rule as was done for the β -rule with the rules in (2): this produces a rule **M** (for μ), to introduce an *explicit replacement*, that also acts at a distance, and another rule **R** (for Replacement), that executes replacements:

$$\begin{array}{ccc}
(\mu\alpha.c)[x_1\backslash v_1] \dots [x_n\backslash v_n]u & \mapsto_{\mathbf{M}} & (\mu\alpha'.c[[\alpha\backslash_{\alpha'}u]])[x_1\backslash v_1] \dots [x_n\backslash v_n] \\
c[[\alpha\backslash_{\alpha'}u]] & \mapsto_{\mathbf{R}} & c\{\{\alpha\backslash_{\alpha'}u\}\}
\end{array} \quad (4)$$

where $c\{\{\alpha\backslash_{\alpha'}u\}\}$ replaces each sub-expression of the form $[\alpha]t$ in c by $[\alpha']tu$. Meaningful computation is seen to be performed by **R** rather than **M**. This observation is further supported by the fact that both sides of the **M**-rule translate into the same proof-net (cf. Sec. 6).

Therefore, we tentatively fix our notion of meaningful reduction to be $\mathbf{S} \cup \mathbf{R}$ over the set of canonical forms, the latter now obtained by taking *both* **B** and **M**-normal forms. However, in contrast to the intuitionistic case where the decomposition of β into a multiplicative rule **B** and an exponential rule **S** suffices for unveiling the strong bisimulation behind Regnier's σ -equivalence in λ -calculus, it turns out that splitting the μ -rule into **M** and **R** is not fine-grained enough. There are various examples, that will be developed in this paper, that illustrate that the resulting relation is still not a strong bisimulation. One such example results from taking the **BM** normal form of the terms in equation (3), as depicted in Fig. 3. This particular use of **R** on the left seems innocuous. In fact we show that in our proposed calculus and its corresponding translation to PPNs, both terms $\mu\alpha'.([\alpha]x)[\alpha\backslash_{\alpha'}y]$ and $\mu\alpha'.[\alpha']xy$ denote structurally equivalent PPNs (cf. Sec. 6 for a detailed discussion). In any case, this example prompts us to further inquire on the fine structure of **R**. In particular, we will argue (Sec. 4) that rule **R** should be further decomposed into *several* independent notions, each one behaving differently with respect to PPNs, and thus with respect to our strong bisimulation \simeq . Identifying these notions and their interplay in order to expose the strong bisimulation hidden behind Laurent's σ -equivalence is the challenge we address in this work.

Contributions. The multiplicative/exponential splitting of the intuitionistic case applied to the classical case, falls noticeably short in identifying programs with control operators whose reduction semantics are in exact correspondence. The need to further decompose rule **R** is rather unexpected, and our proposed decomposition turns out to be subtle yet admits a natural translation to PPN. Moreover, it allows us to obtain a novel and far from obvious strong bisimulation result, highlighting the deep correspondence between PPNs and classical term calculi. Our contributions may be summarized as follows:

1. A refinement of $\lambda\mu$, called ΛM -calculus, including explicit substitutions for variables (resp. explicit replacement for names), and being confluent (Thm. 5).
2. A natural interpretation of ΛM into PPN. More precisely, ΛM -reduction can be implemented by PPN cut elimination (Thm. 14).

3. A notion of structural equivalence \simeq for ΛM which:
 - a. characterizes PPN modulo structural equivalence (Thm. 21);
 - b. is conservative over Laurent's original equivalence \simeq_σ (Thm. 22);
 - c. is a strong bisimulation with respect to meaningful steps (Thm. 25).

Structure of the Paper. Sec. 2 and 3 present $\lambda\mu$ and ΛM , resp. Sec. 4 presents a further refinement of ΛM . Sec. 5 defines typed ΛM -objects, Sec. 6 defines polarized proof-nets, and presents the translation from the former to the latter. Sec. 7 presents our equivalence \simeq . Its properties are discussed and proved in Sec. 8 and Sec. 9. Finally, Sec. 10 concludes and describes related work. Most proofs can be found in [7] including extended details on PPNs, whose presentation has been abridged in this paper.

2 The $\lambda\mu$ -calculus

Preliminary Concepts. A rewrite system \mathcal{R} is a set of objects and a binary (one-step) *reduction* relation $\rightarrow_{\mathcal{R}}$ over those objects. We write $\rightarrow_{\mathcal{R}}$ (resp. $\rightarrow_{\mathcal{R}}^+$) for the reflexive-transitive (resp. transitive) closure of $\rightarrow_{\mathcal{R}}$. A term t is in \mathcal{R} -normal form, written $t \in \mathcal{R}$ -nf or simply $t \in \mathcal{R}$, if there is no t' s.t. $t \rightarrow_{\mathcal{R}} t'$.

Syntax. We fix a countable infinite set of **variables** x, y, z, \dots and **continuation names** $\alpha, \beta, \gamma, \dots$. The set of **objects** $\mathcal{O}(\lambda\mu)$, **terms** $\mathcal{T}(\lambda\mu)$, **commands** $\mathcal{C}(\lambda\mu)$ and **contexts** of the $\lambda\mu$ -calculus are given by the following grammar:

Objects	$o ::= t \mid c$
Terms	$t ::= x \mid tt \mid \lambda x.t \mid \mu\alpha.c$
Commands	$c ::= [\alpha]t$
Contexts	$\mathcal{O} ::= \mathsf{T} \mid \mathsf{C}$
Term Context	$\mathsf{T} ::= \square \mid \mathsf{T}t \mid t\mathsf{T} \mid \lambda x.\mathsf{T} \mid \mu\alpha.\mathsf{C}$
Command Context	$\mathsf{C} ::= \sqsupset \mid [\alpha]\mathsf{T}$

The term $(\dots((tu_1)u_2)\dots)u_n$ abbreviates as $tu_1u_2\dots u_n$. The grammar extends λ -terms with two new constructors: commands $[\alpha]t$ and μ -abstractions $\mu\alpha.c$. Regarding contexts, there are two holes \square and \sqsupset of sort term (t) and command (c) respectively. We write $\mathcal{O}\langle o \rangle$ to denote the replacement of the hole \square (resp. \sqsupset) by a term (resp. by a command). We often decorate contexts or functions over expressions with sorts t and c. For example, \mathcal{O}_t is a context \mathcal{O} with a hole of sort *term*. The subscript is omitted if it is clear from the context.

Free and bound variables of objects are defined as expected, in particular $\mathbf{fv}(\mu\alpha.c) \stackrel{\text{def}}{=} \mathbf{fv}(c)$ and $\mathbf{fv}([\alpha]t) \stackrel{\text{def}}{=} \mathbf{fv}(t)$. **Free names** of objects are defined as follows: $\mathbf{fn}(x) \stackrel{\text{def}}{=} \emptyset$, $\mathbf{fn}(\lambda x.t) \stackrel{\text{def}}{=} \mathbf{fn}(t)$, $\mathbf{fn}(tu) \stackrel{\text{def}}{=} \mathbf{fn}(t) \cup \mathbf{fn}(u)$, $\mathbf{fn}(\mu\alpha.c) \stackrel{\text{def}}{=} \mathbf{fn}(c) \setminus \{\alpha\}$, and $\mathbf{fn}([\alpha]t) \stackrel{\text{def}}{=} \mathbf{fn}(t) \cup \{\alpha\}$. **Bound names** are defined accordingly. We use $\mathbf{fv}_x(o)$ (resp. $\mathbf{fn}_\alpha(o)$) to denote the number of free occurrences of the variable x (resp. name α) in o . We write $x \notin o$ (resp. $\alpha \notin o$) if $x \notin \mathbf{fv}(o)$ and $x \notin \mathbf{bv}(o)$ (resp. $\alpha \notin \mathbf{fn}(o)$ and $\alpha \notin \mathbf{bn}(o)$). This notion is extended to contexts as expected.

We work with the standard notion of α -**conversion** i.e. renaming of bound variables and names, thus for example $[\delta](\mu\alpha.[\alpha](\lambda x.x))z \equiv_{\alpha} [\delta](\mu\beta.[\beta](\lambda y.y))z$. In particular, when using two different symbols to denote bound variables (resp. names), we assume that they are distinct, without explicitly mentioning it.

Semantics. **Application** of the **substitution** $\{x \setminus u\}$ to the object o , written $o\{x \setminus u\}$, may require α -conversion in order to avoid capture of free variables/names, and it is defined as expected. **Application** of the **replacement** $\{\alpha \setminus_{\alpha'} u\}$ to an object o , where $\alpha \neq \alpha'$, written $o\{\alpha \setminus_{\alpha'} u\}$, passes the term u as an argument to any sub-command of o of the form $[\alpha]t$ and changes the name of α to α' . This operation is also defined modulo α -conversion in order to avoid the capture of free variables/names. Formally:

$$\begin{aligned} x\{\alpha \setminus_{\alpha'} u\} &\stackrel{\text{def}}{=} x \\ (tv)\{\alpha \setminus_{\alpha'} u\} &\stackrel{\text{def}}{=} t\{\alpha \setminus_{\alpha'} u\}v\{\alpha \setminus_{\alpha'} u\} \\ (\lambda x.t)\{\alpha \setminus_{\alpha'} u\} &\stackrel{\text{def}}{=} \lambda x.t\{\alpha \setminus_{\alpha'} u\} && x \notin u \\ (\mu\beta.c)\{\alpha \setminus_{\alpha'} u\} &\stackrel{\text{def}}{=} \mu\beta.c\{\alpha \setminus_{\alpha'} u\} && \beta \notin (u, \alpha, \alpha') \\ ([\alpha]c)\{\alpha \setminus_{\alpha'} u\} &\stackrel{\text{def}}{=} [\alpha']c\{\alpha \setminus_{\alpha'} u\}u \\ ([\beta]c)\{\alpha \setminus_{\alpha'} u\} &\stackrel{\text{def}}{=} [\beta]c\{\alpha \setminus_{\alpha'} u\} && \beta \neq \alpha \end{aligned}$$

For example, if $I = \lambda z.z$, then $((\mu\alpha.[\alpha]x)(\lambda z.zx))\{x \setminus I\}$ is equal to $(\mu\alpha.[\alpha]I)(\lambda z.zI)$, and $([\alpha]x(\mu\beta.[\alpha]y))\{\alpha \setminus_{\alpha'} I\} = [\alpha']x(\mu\beta.[\alpha']yI)I$.

► **Definition 1.** The $\lambda\mu$ -calculus is given by the set $\mathcal{O}(\lambda\mu)$ and the $\lambda\mu$ -reduction relation $\rightarrow_{\lambda\mu}$, defined as the closure by all contexts of the following rewriting rules² (equivalently, $\rightarrow_{\lambda\mu} \stackrel{\text{def}}{=} \mathcal{O}_t(\mapsto_{\beta} \cup \mapsto_{\mu})$):

$$\begin{aligned} (\lambda x.t)u &\mapsto_{\beta} t\{x \setminus u\} \\ (\mu\alpha.c)u &\mapsto_{\mu} \mu\alpha'.c\{\alpha \setminus_{\alpha'} u\} \end{aligned}$$

Various control operators can be expressed in the $\lambda\mu$ -calculus [11, 21]. A typical example of expressiveness of the $\lambda\mu$ -calculus is the control operator **call-cc** [13], specified by the term $\lambda x.\mu\alpha.[\alpha]x(\lambda y.\mu\delta.[\alpha]y)$. The term **call-cc** is assigned the type $((A \rightarrow B) \rightarrow A) \rightarrow A$ (Peirce's Law) in the simply typed $\lambda\mu$ -calculus, thus capturing classical logic.

The Notion of σ -Equivalence for $\lambda\mu$ -Terms. As in λ -calculus, structural equivalence for $\lambda\mu$ captures inessential permutation of redexes, but this time also involving the control constructs. Laurent's notion of σ -equivalence for $\lambda\mu$ -terms [23] (written here also \simeq_{σ}) is depicted in Fig. 4. The first two equations are exactly those of Regnier (hence \simeq_{σ} on $\lambda\mu$ -terms strictly extends \simeq_{σ} on λ -terms); the remaining ones involve interactions between control operators themselves or control operators and application and abstraction.

Laurent proved properties for \simeq_{σ} similar to those of Regnier for \simeq_{σ} . More precisely, $u \simeq_{\sigma} v$ implies that u is normalizable (resp. is head normalizable, strongly normalizable) iff v is normalizable (resp. is head normalizable, strongly normalizable) [23, Prop. 35]. Based on Girard's encoding of classical into linear logic [12], he also proved that the translation of the left and right-hand sides of the equations of \simeq_{σ} , in a typed setting, yield structurally equivalent PPNs [23, Thm. 41]. These results are non-trivial because the left and right-hand side of the equations in Fig. 4 do not have the same β and μ redexes. For example, $(\mu\alpha.[\alpha]x)y$ and xy are related by equation σ_8 , however the former has a μ -redex (more precisely it has a *linear* μ -redex) and the latter has none. Indeed, as mentioned in Sec. 1 (cf. the terms in (3)), \simeq_{σ} is not a strong bisimulation with respect to $\lambda\mu$ -reduction (cf. Fig. 5). There are other

² Parigot [25]'s μ -rule $(\mu\alpha.c)u \mapsto_{\mu} \mu\alpha.c\{\alpha \setminus u\}$, relies on a binary replacement operation $\{\alpha \setminus u\}$ assigning $[\alpha](t\{\alpha \setminus u\})u$ to $[\alpha]t$ (thus not changing the name of the command). We remark that $\mu\alpha.c\{\alpha \setminus u\} \equiv_{\alpha} \mu\alpha'.c\{\alpha \setminus_{\alpha'} u\}$. We adopt here the ternary presentation of the replacement operator [20], because it naturally extends to that of the ΛM -calculus in Sec. 3.

$$\begin{array}{ll}
 (\lambda y. \lambda x. t) v \simeq_{\sigma_1} \lambda x. (\lambda y. t) v & x \notin v \\
 (\lambda x. t v) u \simeq_{\sigma_2} (\lambda x. t) u v & x \notin v \\
 (\lambda x. \mu \alpha. [\beta] u) w \simeq_{\sigma_3} \mu \alpha. [\beta] (\lambda x. u) w & \alpha \notin w \\
 [\alpha'] (\mu \alpha. [\beta'] (\mu \beta. c) w) v \simeq_{\sigma_4} [\beta'] (\mu \beta. [\alpha'] (\mu \alpha. c) v) w & \alpha \notin w, \beta \notin v, \beta \neq \alpha', \alpha \neq \beta' \\
 [\alpha'] (\mu \alpha. [\beta'] \lambda x. \mu \beta. c) v \simeq_{\sigma_5} [\beta'] \lambda x. \mu \beta. [\alpha'] (\mu \alpha. c) v & x \notin v, \beta \notin v, \beta \neq \alpha', \alpha \neq \beta' \\
 [\alpha'] \lambda x. \mu \alpha. [\beta'] \lambda y. \mu \beta. c \simeq_{\sigma_6} [\beta'] \lambda y. \mu \beta. [\alpha'] \lambda x. \mu \alpha. c & \beta \neq \alpha', \alpha \neq \beta' \\
 [\alpha] \mu \beta. c \simeq_{\sigma_7} c \{ \beta \setminus \alpha \} & \\
 \mu \alpha. [\alpha] v \simeq_{\sigma_8} v & \alpha \notin v
 \end{array}$$

■ **Figure 4** Laurent's σ -equivalence for $\lambda\mu$ -terms.

$$\begin{array}{ccc}
 (\mu \alpha. [\alpha] x) y & \simeq_{\sigma_8} & x y \\
 \mu \downarrow & & \downarrow \mu \\
 \mu \alpha. [\alpha] x y & \simeq_{\sigma_8} & x y
 \end{array}$$

■ **Figure 5** Laurent's \simeq_{σ} equivalence not a strong bisimulation.

examples illustrating that \simeq_{σ} is not a strong bisimulation (cf. Sec. 7). It seems natural to wonder whether, just like in the intuitionistic case, a more refined notion of $\lambda\mu$ -reduction could change this state of affairs; that is a challenge we take up in this paper.

3 The ΛM -calculus

We now extend the syntax of $\lambda\mu$ to that of ΛM . We again fix a countable infinite set of **variables** x, y, z, \dots and **continuation names** $\alpha, \beta, \gamma, \dots$. The set of **objects** $\mathcal{O}(\Lambda M)$, **terms** $\mathcal{T}(\Lambda M)$, **commands** $\mathcal{C}(\Lambda M)$, **stacks** and **contexts** are given by the following grammar:

Objects	$o ::= t \mid c \mid s$
Terms	$t ::= x \mid t t \mid \lambda x. t \mid \mu \alpha. c \mid t[x \setminus t]$
Commands	$c ::= [\alpha] t \mid c \llbracket \alpha \setminus \alpha' s \rrbracket$
Stacks	$s ::= \# \mid t \cdot s$
Contexts	$\mathbf{0} ::= \mathbf{T} \mid \mathbf{C} \mid \mathbf{S}$
Term Contexts	$\mathbf{T} ::= \square \mid \mathbf{T} t \mid t \mathbf{T} \mid \lambda x. \mathbf{T} \mid \mu \alpha. \mathbf{C} \mid \mathbf{T}[x \setminus t] \mid t[x \setminus \mathbf{T}]$
Command Contexts	$\mathbf{C} ::= \square \mid [\alpha] \mathbf{T} \mid \mathbf{C} \llbracket \alpha \setminus \alpha' s \rrbracket \mid c \llbracket \alpha \setminus \alpha' \mathbf{S} \rrbracket$
Stack Contexts	$\mathbf{S} ::= \mathbf{T} \cdot s \mid t \cdot \mathbf{S}$
Substitution Contexts	$\mathbf{L} ::= \square \mid \mathbf{L}[x \setminus t]$
Replacement Contexts	$\mathbf{R} ::= \square \mid \mathbf{R} \llbracket \alpha \setminus \alpha' s \rrbracket$

Terms of $\lambda\mu$ are enriched with **explicit substitutions** of the form $t[x \setminus u]$. Commands of $\lambda\mu$ are enriched with **explicit replacements** of the form $c \llbracket \alpha \setminus \alpha' s \rrbracket$, where $\alpha \neq \alpha'$, and α' is called a **replacement name**. Stacks are empty ($\#$) or non-empty ($t \cdot s$). Explicit replacements with empty stacks (i.e. $\llbracket \beta \setminus \alpha \# \rrbracket$) are called **renaming replacements**, otherwise **stack replacements**.

Stack concatenation, denoted $s \cdot s'$, is defined as expected, where $_ \cdot _$ is associative and $\#$ is the neutral element. We often write $t_1 \dots t_n \cdot \#$ simply as $t_1 \dots t_n$ (thus, in particular, $u \cdot \#$ is abbreviated as u). Moreover, given a term u , we use the abbreviation $u :: s$

for the term u if $s = \#$ and $((u t_1) \dots) t_n$ if $s = t_1 \cdot \dots \cdot t_n$. This operation is left-associative, hence $u :: s_1 :: s_2$ means $(u :: s_1) :: s_2$.

Free and bound variables of ΛM -objects are extended as expected. In particular, $\text{fv}(t[x \setminus u]) \stackrel{\text{def}}{=} \text{fv}(t) \setminus \{x\} \cup \text{fv}(u)$, and $\text{fv}(c[\gamma \setminus \gamma' s]) \stackrel{\text{def}}{=} \text{fv}(c) \cup \text{fv}(s)$, while $\text{fn}(t[x \setminus u]) \stackrel{\text{def}}{=} \text{fv}(t) \cup \text{fv}(u)$, and $\text{fn}(c[\gamma \setminus \gamma' s]) \stackrel{\text{def}}{=} \text{fn}(c) \setminus \{\gamma\} \cup \{\gamma'\} \cup \text{fn}(s)$. We work, as usual, modulo α -conversion so that bound variables and names can be renamed. Thus e.g. $x[x \setminus u] \equiv_{\alpha} y[y \setminus u]$ and $\mu\gamma.[\gamma] x \equiv_{\alpha} \mu\beta.[\beta] x$. In particular, we will always assume by α -conversion, that $x \notin \text{fv}(u)$ in the term $t[x \setminus u]$ and $\alpha \notin \text{fn}(s)$ in the command $c[\alpha \setminus \alpha' s]$.

The notions of free and bound variables and names are extended to contexts by defining $\text{fv}(\square) = \text{fv}(\sqsupset) = \text{fn}(\square) = \text{fn}(\sqsupset) = \emptyset$. A variable x **occurs bound** in \mathcal{O} if, for any fresh variable $y \neq x$, it occurs in $\mathcal{O}\langle y \rangle$ but not free. Similarly for names. Thus for example x is bound in $\lambda x.\square$ and $(\lambda x.x)\square$ and α is bound in $\square[\alpha \setminus \alpha' s]$. We use $\text{fv}(o_1, o_2)$ (resp. $\text{fn}(o_1, o_2)$) to abbreviate $\text{fv}(o_1) \cup \text{fv}(o_2)$. (resp. $\text{fn}(o_1) \cup \text{fn}(o_2)$) and also $\text{fn}(o, \alpha)$ to abbreviate $\text{fn}(o) \cup \{\alpha\}$. An object o is **free for a context** \mathcal{O} , written $\text{fc}(o, \mathcal{O})$, if the bound variables and bound names of \mathcal{O} do not occur free in o . Thus for example $\text{fc}(zy, \lambda x.(\square[x' \setminus w]))$ holds but $\text{fc}(xy, \lambda x.\square)$ does not hold. This notation is naturally extended to sets, i.e. $\text{fc}(S, \mathcal{O})$ means that the bound variables and bound names of \mathcal{O} do not occur free in any element of S . We write $x \notin o$ (resp. $\alpha \notin o$) if $x \notin \text{fv}(o)$ and $x \notin \text{bv}(o)$ (resp. $\alpha \notin \text{fn}(o)$ and $\alpha \notin \text{bn}(o)$). This notion is extended to contexts as expected.

As in Sec. 2, we use $o\{x \setminus u\}$ and $o\{\alpha \setminus \alpha' s\}$ to denote, respectively, the natural extensions of the **substitution** and **replacement** operations to ΛM -objects. Both are defined modulo α -conversion to avoid capture of free variables/names. While the first notion is standard, we formalise the second one.

► **Definition 2.** Given $\alpha \notin \text{fn}(s)$, the replacement $o\{\alpha \setminus \alpha' s\}$ is defined as follows:

$$\begin{aligned}
x\{\alpha \setminus \alpha' s\} &\stackrel{\text{def}}{=} x \\
(tu)\{\alpha \setminus \alpha' s\} &\stackrel{\text{def}}{=} t\{\alpha \setminus \alpha' s\}u\{\alpha \setminus \alpha' s\} \\
(\lambda x.t)\{\alpha \setminus \alpha' s\} &\stackrel{\text{def}}{=} \lambda x.t\{\alpha \setminus \alpha' s\} && x \notin s \\
(\mu\beta.c)\{\alpha \setminus \alpha' s\} &\stackrel{\text{def}}{=} \mu\beta.c\{\alpha \setminus \alpha' s\} && \beta \notin (s, \alpha, \alpha') \\
t[x \setminus u]\{\alpha \setminus \alpha' s\} &\stackrel{\text{def}}{=} t\{\alpha \setminus \alpha' s\}[x \setminus u\{\alpha \setminus \alpha' s\}] && x \notin s \\
([\alpha]t)\{\alpha \setminus \alpha' s\} &\stackrel{\text{def}}{=} [\alpha'](t\{\alpha \setminus \alpha' s\} :: s) \\
([\beta]t)\{\alpha \setminus \alpha' s\} &\stackrel{\text{def}}{=} [\beta]t\{\alpha \setminus \alpha' s\} && \alpha \neq \beta \\
c[\gamma \setminus \beta s']\{\alpha \setminus \alpha' s\} &\stackrel{\text{def}}{=} c\{\alpha \setminus \alpha' s\}[\gamma \setminus \beta s'\{\alpha \setminus \alpha' s\}] && \alpha \neq \beta \\
c[\gamma \setminus \alpha \#]\{\alpha \setminus \alpha' s\} &\stackrel{\text{def}}{=} c\{\alpha \setminus \alpha' s\}[\gamma \setminus \beta s][\beta \setminus \alpha' \#] && s \neq \#, \beta \text{ fresh} \\
c[\gamma \setminus \alpha \#]\{\alpha \setminus \alpha' \#\} &\stackrel{\text{def}}{=} c\{\alpha \setminus \alpha' \#\}[\gamma \setminus \alpha' \#] \\
c[\gamma \setminus \alpha s']\{\alpha \setminus \alpha' s\} &\stackrel{\text{def}}{=} c\{\alpha \setminus \alpha' s\}[\gamma \setminus \alpha' s'\{\alpha \setminus \alpha' s\} \cdot s] && s' \neq \# \\
\#\{\alpha \setminus \alpha' s\} &\stackrel{\text{def}}{=} \# \\
(t \cdot s')\{\alpha \setminus \alpha' s\} &\stackrel{\text{def}}{=} t\{\alpha \setminus \alpha' s\} \cdot s'\{\alpha \setminus \alpha' s\}
\end{aligned}$$

E.g. $([\alpha]x)\{\alpha \setminus \gamma y_1 \cdot y_2\} = [\gamma]x y_1 y_2$, and $([\alpha]x)[\beta \setminus \alpha z_1]\{\alpha \setminus \gamma y_1\} = ([\gamma]x y_1)[\beta \setminus \gamma z_1 \cdot y_1]$, while $([\alpha]x)[\beta \setminus \alpha \#]\{\alpha \setminus \gamma y_1 \cdot y_2\} = ([\gamma]x y_1 y_2)[\beta \setminus \gamma' y_1 \cdot y_2][\gamma' \setminus \gamma \#]$.

When $s = \#$, the replacement operation $_ \{\alpha \setminus \alpha' s\}$ is called a **renaming**. Most of the cases in the definition above are straightforward, we only comment on the interesting ones. When the (meta-level) replacement operator affects a renaming replacement, i.e. in the case $c[\gamma \setminus \alpha \#]\{\alpha \setminus \alpha' s\}$, the renaming $[\gamma \setminus \alpha \#]$ is *blocking* the replacement, so that an explicit replacement $[\gamma \setminus \beta s]$ with a fresh name β is created, and β is then renamed to α' . Regarding the last clause of the definition for commands, since the explicit replacement $[\gamma \setminus \alpha' s']$ is blocking, it accumulates any additional arguments for γ , hence why *stacks* (i.e. sequences of terms) are used, instead of terms, as target for names.

4:10 Strong Bisimulation for Control Operators

The two operations $o\{x\backslash u\}$, and $o\{\alpha\backslash_{\alpha'}s\}$ are extended to contexts as expected.

► **Definition 3.** The ΛM -calculus is given by the set of objects $\mathcal{O}(\Lambda M)$ and the ΛM -reduction relation $\rightarrow_{\Lambda M}$, defined as the closure by all contexts of the rewriting rules:

$$\begin{array}{llll} L\langle \lambda x.t \rangle u & \mapsto_{\mathbf{B}} & L\langle t[x\backslash u] \rangle & L\langle \mu\alpha.c \rangle u & \mapsto_{\mathbf{M}} & L\langle \mu\alpha'.c[\alpha\backslash_{\alpha'}u \cdot \#] \rangle \\ t[x\backslash u] & \mapsto_{\mathbf{S}} & t\{x\backslash u\} & c[\alpha\backslash_{\alpha'}s] & \mapsto_{\mathbf{R}} & c\{\alpha\backslash_{\alpha'}s\} \end{array}$$

where $\mapsto_{\mathbf{B}}$ and $\mapsto_{\mathbf{M}}$ are both constrained by the condition $\mathbf{fc}(u, L)$, and $\mapsto_{\mathbf{M}}$ also requires $\alpha' \notin (c, u, \alpha, L)$, thus both rules pull the list context L out by avoiding the capture of free variables/names of u . Equivalently, $\rightarrow_{\Lambda M} \stackrel{\text{def}}{=} \mathbf{0}_t(\mapsto_{\mathbf{B}} \cup \mapsto_{\mathbf{S}} \cup \mapsto_{\mathbf{M}}) \cup \mathbf{0}_c(\mapsto_{\mathbf{R}})$. Given $X \in \{\mathbf{B}, \mathbf{S}, \mathbf{M}, \mathbf{R}\}$, we write \rightarrow_X for the closure by all contexts of \mapsto_X .

Note that \mathbf{B} and \mathbf{M} above, also presented in (2) and (4) of the introduction, operate at a distance [5], a characteristic in line with our semantical development being guided by Proof-Nets. Also, following Parigot [25], one might be tempted to rephrase the reduct of \mathbf{M} with a binary constructor, writing $L\langle \mu\alpha.c[\alpha\backslash u] \rangle$. But this is imprecise since free occurrences of α in c cannot be bound to both $\mu\alpha$ and $[\alpha\backslash u]$. The subscript α' in $c[\alpha\backslash_{\alpha'}u]$ shall replace α in c as described above. The ΛM -calculus implements the $\lambda\mu$ -calculus by means of more atomic steps, i.e.

► **Lemma 4.** Let $o \in \mathcal{O}(\lambda\mu)$. If $o \rightarrow_{\lambda\mu} o'$, then $o \rightarrow_{\Lambda M} o'$.

Just like $\lambda\mu$, the ΛM -calculus is confluent too. This is proved by using the interpretation method [15], where ΛM is interpreted into $\lambda\mu$ by means of a suitable projection function.

► **Theorem 5.** The $\rightarrow_{\Lambda M}$ relation is confluent.

Proof. By the interpretation method [15], using confluence of $\rightarrow_{\lambda\mu}$ [25]. Details in [7]. ◀

4 Refining Replacement

Now that we have introduced the relation ΛM and hence the reader has a clearer picture of the presentation/implementation of $\lambda\mu$ we will be working with, we briefly revisit our objective. We seek to identify a relation \simeq on a subset of ΛM -terms (which we call *canonical*) that is a strong bisimulation for a subset of ΛM -reduction (which we call *meaningful*). The proof-net translation of the intuitionistic case suggests that \simeq be defined on the subset of ΛM terms that are in \mathbf{BM} -normal form, since a term and its \mathbf{BM} -reduct are essentially different syntactic presentations of the same thing. Consequently, we can in principle declare $\mathbf{S} \cup \mathbf{R}$ as the subset of ΛM -reduction that is *meaningful*. However the proof-net translation of ΛM suggests that meaningful reduction for ΛM is the exponential one, manipulating *boxes* (cf. Sec. 6), which allow in particular their erasure and duplication, and unfortunately \mathbf{R} also includes cases without any box manipulation. Moreover, as hinted at in the introduction, when incorporating the \mathbf{BM} -normal form of Laurent's σ -equivalence equations into \simeq , one immediately realizes that strong bisimulation fails. The heart of the matter is that \mathbf{R} is too course-grained and that we should break it down, weeding out those instances that present an obstacle to strong bisimulation. Indeed, one can distinguish between linear and non-linear instances of \mathbf{R} , the translation of the latter involving boxes while the former's not. This section presents a refinement of \mathbf{R} , in four stages, identifying a subset of replacement \mathbf{R} we dub *meaningful replacement reduction*. The latter, together with \mathbf{S} , will conform the whole notion of *meaningful reduction* (Def 9).

$$\begin{array}{ccc}
([\alpha] \mu\beta.c)[\alpha \setminus_{\alpha'} s] & \simeq_{\sigma_7} & c\{\beta \setminus \alpha\}[\alpha \setminus_{\alpha'} s] \\
\downarrow \mathbf{R} & & \downarrow \mathbf{R} \\
[\alpha'](\mu\beta.c\{\{\alpha \setminus_{\alpha'} s\}\})s & & \\
\downarrow \mathbf{BM} & & \\
[\alpha']\mu\beta'.\mathbf{BM}(c\{\{\alpha \setminus_{\alpha'} s\}\}\{\{\beta \setminus_{\beta'} s\}\}) & \text{???} & \mathbf{BM}(c\{\beta \setminus \alpha\}\{\{\alpha \setminus_{\alpha'} s\}\})
\end{array}$$

■ **Figure 6** Implicit renaming and strong bisimulation.

Stage 1: Renaming vs Stack Replacement. In this first stage we split \mathbf{R} according to the nature of the explicit replacement, renaming or stack:

$$\begin{array}{l}
c[\alpha \setminus_{\alpha'} \#] \mapsto_{\mathbf{R}_{\#}} c\{\{\alpha \setminus_{\alpha'} \#\}\} \\
c[\alpha \setminus_{\alpha'} s] \mapsto_{\mathbf{R}_{\neg\#}} c\{\{\alpha \setminus_{\alpha'} s\}\} \quad \text{if } s \neq \#
\end{array}$$

Accordingly, we call $\mathbf{R}_{\#}$ the renaming replacement rule and $\mathbf{R}_{\neg\#}$ the stack replacement rule. The renaming replacement rule unfortunately throws away important information that is required for our strong-bisimulation. As an example, consider the equation σ_7 of Fig. 4, but under a ΛM context containing an explicit replacement $[\alpha' \setminus_{\alpha} s]$, as depicted in Fig. 6. Firing $[\alpha' \setminus_{\alpha} s]$ on the left leads to the creation of another stack replacement redex on the left, with no such redex mimicking it on the right. Indeed, the term on the lower left hand corner has a pending explicit replacement and hence cannot be equated with the term on the lower right hand corner.

As for our notion of *meaningful replacement* reduction, this leads us to disregard the renaming replacement rule, leaving renaming replacements in terms as is, that is, *without executing them*. An adaptation of σ_7 to ΛM will later be adopted in our \simeq relation (cf. Sec. 7) where (implicit) renaming is left pending as an explicit renaming replacement. In summary, at this stage, our notion of meaningful replacement reduction is taken to be just $\mathbf{R}_{\neg\#}$.

Stage 2: A First Refinement of Stack Replacement. The next stage in the refinement process is to further split the stack replacement rule $\mathbf{R}_{\neg\#}$ based on the number of free occurrences of the name to be replaced, as indicated by $\mathbf{fn}_{\alpha}(c)$ below. The motivation behind this split is that some instances of the replacement rule that replace exactly one name, actually relate terms that are structurally equivalent when translated to PPNs, hence perform no meaningful computation. Thus we consider the following rules:

$$\begin{array}{l}
c[\alpha \setminus_{\alpha'} s] \mapsto_{\mathbf{R}_{\neg\#}^{\neq 1}} c\{\{\alpha \setminus_{\alpha'} s\}\} \quad \text{if } \mathbf{fn}_{\alpha}(c) \neq 1 \text{ and } s \neq \# \\
c[\alpha \setminus_{\alpha'} s] \mapsto_{\mathbf{R}_{\neg\#}^{\equiv 1}} c\{\{\alpha \setminus_{\alpha'} s\}\} \quad \text{if } \mathbf{fn}_{\alpha}(c) = 1 \text{ and } s \neq \#
\end{array}$$

In rule $\mathbf{R}_{\neg\#}^{\neq 1}$ we immediately recognize as involving substantial work due to duplication or erasure of the stack s . Hence, we update our current notion of meaningful replacement computation by replacing our former selection of $\mathbf{R}_{\neg\#}$ with the more restricted $\mathbf{R}_{\neg\#}^{\neq 1}$.

We next have to determine whether rule $\mathbf{R}_{\neg\#}^{\equiv 1}$, or refinements thereof, should too be judged as meaningful. For that we must take a closer look at its behavior for specific instances of the command c based on the possible (unique) occurrence of the name α : on a named term (Stage 3) or on an explicit replacement (Stage 4).

Stage 3: Stack Replacement on Named Term. In the right-hand side of rule $\mathbf{R}_{\neg\#}^{\equiv 1}$ the command c is traversed by the meta-level replacement $\{\{\alpha \setminus_{\alpha'} s\}\}$ until the unique free occurrence α is reached. Since free names only occur in commands, the left hand-side of $\mathbf{R}_{\neg\#}^{\equiv 1}$ has

4:12 Strong Bisimulation for Control Operators

necessarily one of the following forms:

$$\mathbf{C}\langle[\alpha]t\rangle\llbracket\alpha\backslash_{\alpha'}s\rrbracket \quad \mathbf{C}\langle c'\llbracket\beta\backslash_{\alpha}s'\rrbracket\rangle\llbracket\alpha\backslash_{\alpha'}s\rrbracket$$

for some context \mathbf{C} and where α does not occur free in \mathbf{C}, t, c', s' . We next focus on the first case leaving the second to Stage 4. The first case gives rise to the rule **name**:

$$\mathbf{C}\langle[\alpha]t\rangle\llbracket\alpha\backslash_{\alpha'}s\rrbracket \mapsto_{\mathbf{name}} \mathbf{C}\langle[\alpha']t :: s\rangle \quad \text{if } \alpha \notin (t, \mathbf{C}), s \neq \#$$

At this point in our development, we pause and briefly discuss *linear μ -redexes* before getting back to **name**. From the very beginning, there was never any hope for σ -equivalence (Fig. 4) to be a strong bisimulation since, as mentioned by Laurent [23], it does not distinguish between terms with *linear μ -redexes*. A μ -redex in $\lambda\mu$ is *linear* if it has the form $(\mu\alpha.Q\langle[\alpha]u\rangle)v$ with $\alpha \notin (u, Q)$ and Q defined as follows:

$$\mathbf{P} ::= \square \mid \mathbf{P}t \mid \lambda x.\mathbf{P} \mid \mu\alpha.[\beta]\mathbf{P} \quad \mathbf{Q} ::= \square \mid [\beta]\mathbf{P}\langle\mu\gamma.\square\rangle$$

An example is the term $(\mu\alpha.[\alpha]x)y$, one of the two terms of (3) mentioned in Sec. 1. Such μ -reduction steps reducing linear μ -redexes hold no operational meaning: if o linearly μ -reduces to o' , then their graphical interpretation yield PPNs whose multiplicative normal form are structurally equivalent [23, Thm. 41] (revisited in Sec. 8 as Thm. 19).

Returning to our development, we next need to identify what a linear/non-linear split of rule **name** looks like in our setting of ΛM , the intuition arising, again, from PPN. For that we introduce *linear contexts*.

► **Definition 6.** *There are four sets of linear contexts, each denoted using the expressions XY , with $X, Y \in \{\mathbf{T}, \mathbf{C}\}$. The letters X and Y in the expression XY denote the sort of the object with which the hole will be filled and the sort of the resulting term, resp.: e.g. **LTC** denotes a context that takes a command and outputs a term.*

$$\begin{aligned} \text{(Linear TT Contexts)} \quad \mathbf{LTT} &::= \square \mid \mathbf{LTT}t \mid \lambda x.\mathbf{LTT} \mid \mu\alpha.\mathbf{LCT} \mid \mathbf{LTT}[x\backslash t] \\ \text{(Linear TC Contexts)} \quad \mathbf{LTC} &::= \mathbf{LTC}t \mid \lambda x.\mathbf{LTC} \mid \mu\alpha.\mathbf{LCC} \mid \mathbf{LTC}[x\backslash t] \\ \text{(Linear CC Contexts)} \quad \mathbf{LCC} &::= \square \mid [\alpha]\mathbf{LTC} \mid \mathbf{LCC}\llbracket\alpha\backslash_{\alpha'}s\rrbracket \\ \text{(Linear CT Contexts)} \quad \mathbf{LCT} &::= [\alpha]\mathbf{LTT} \mid \mathbf{LCT}\llbracket\alpha\backslash_{\alpha'}s\rrbracket \end{aligned}$$

For example, $[\alpha]\square$ is a **LTC** context and $[\beta](\square v)\llbracket\alpha\backslash_{\alpha'}u\rrbracket$ is a **LTT** context.

Given the above definition of linear contexts we can now split **name** into its linear and non-linear versions:

$$\begin{aligned} \mathbf{C}\langle[\alpha]t\rangle\llbracket\alpha\backslash_{\alpha'}s\rrbracket &\mapsto_{\mathbf{N}_{-1\text{in}}} \mathbf{C}\langle[\alpha']t :: s\rangle \quad \text{if } \mathbf{C} \text{ not linear, } \alpha \notin (t, \mathbf{C}), s \neq \# \\ \mathbf{LCC}\langle[\alpha]t\rangle\llbracket\alpha\backslash_{\alpha'}s\rrbracket &\mapsto_{\mathbf{N}} \mathbf{LCC}\langle[\alpha']t :: s\rangle \quad \text{if } \alpha \notin (t, \mathbf{LCC}), s \neq \# \end{aligned}$$

The named term $[\alpha]t$ in the non-linear rule $\mathbf{N}_{-1\text{in}}$ could be duplicated or erased and thus this rule joins $\mathbf{R}_{-1\text{in}}^{\neq 1}$ as part of meaningful replacement reduction. However, as was the case above for linear μ -redexes, rule **N** has no meaningful computational content and hence will be incorporated into \simeq by taking its canonical normal form (**LCC** and t below are assumed in canonical normal form). The new equation is called **lin**:

$$\mathbf{LCC}\langle[\alpha]t\rangle\llbracket\alpha\backslash_{\alpha'}s\rrbracket \simeq_{1\text{in}} \mathbf{LCC}\langle[\alpha']\mathfrak{C}(t :: s)\rangle \quad \text{if } \alpha \notin (t, \mathbf{LCC}), s \neq \#$$

The notation $\mathfrak{C}(_)$ denotes the canonical form of an object. A precise definition will be presented in Stage 4.

Summarizing our results of Stage 3, meaningful replacement reduction consists for the moment of $\mathbf{R}_{-1\text{in}}^{\neq 1}$ and $\mathbf{N}_{-1\text{in}}$. In the next and final stage, we analyze the case where the left hand-side of $\mathbf{R}_{-1\text{in}}^{\neq 1}$ has the form $\mathbf{C}\langle c'\llbracket\beta\backslash_{\alpha}s'\rrbracket\rangle\llbracket\alpha\backslash_{\alpha'}s\rrbracket$.

Stage 4: Stack Replacement on Explicit Replacements. Suppose the left hand-side of $R_{-\#}^1$ has the form $C\langle c'[\beta \setminus_{\alpha} s'] \rangle [\alpha \setminus_{\alpha'} s]$. This gives rise to the two instances **swap** and **comp**:

$$\begin{array}{l} C\langle c'[\beta \setminus_{\alpha} \#] \rangle [\alpha \setminus_{\alpha'} s] \mapsto_{\text{swap}} C\langle c'[\beta \setminus_{\alpha} s] \rangle [\alpha \setminus_{\alpha'} \#] \quad \text{if } \alpha \notin (c', C), s \neq \# \\ C\langle c'[\beta \setminus_{\alpha} s'] \rangle [\alpha \setminus_{\alpha'} s] \mapsto_{\text{comp}} C\langle c'[\beta \setminus_{\alpha} s' \cdot s] \rangle \quad \text{if } \alpha \notin (c', C, s'), s', s \neq \# \end{array}$$

If we now consider linear/non-linear variants of the above two rules, depending on whether the context C in their LHSs is a linear context or not (in the sense of Def. 6), we end up with the following four rules, where we use letters r and r' to denote non-empty stacks.

$$\begin{array}{l} C\langle c'[\beta \setminus_{\alpha} \#] \rangle [\alpha \setminus_{\alpha'} r] \mapsto_{W_{\text{-lin}}} C\langle c'[\beta \setminus_{\alpha} r] \rangle [\alpha \setminus_{\alpha'} \#] \quad C \text{ not linear, } \alpha \notin (c', C) \\ C\langle c'[\beta \setminus_{\alpha} r'] \rangle [\alpha \setminus_{\alpha'} r] \mapsto_{C_{\text{-lin}}} C\langle c'[\beta \setminus_{\alpha} r' \cdot r] \rangle \quad C \text{ not linear, } \alpha \notin (c', C, s') \\ LCC\langle c'[\beta \setminus_{\alpha} \#] \rangle [\alpha \setminus_{\alpha'} r] \mapsto_W LCC\langle c'[\beta \setminus_{\alpha} r] \rangle [\alpha \setminus_{\alpha'} \#] \quad \alpha \notin (c', LCC), fc(\{s, \alpha'\}, LCC) \\ LCC\langle c'[\beta \setminus_{\alpha} r'] \rangle [\alpha \setminus_{\alpha'} r] \mapsto_C LCC\langle c'[\beta \setminus_{\alpha} r' \cdot r] \rangle \quad \alpha \notin (c', LCC, s'), fc(\{s, \alpha'\}, LCC) \end{array}$$

The rules involving non-linear contexts, namely $W_{\text{-lin}}$ and $C_{\text{-lin}}$ join $R_{-\#}^{\neq 1}$ and $N_{\text{-lin}}$ in conforming meaningful replacement computation. Indeed, although there is a unique occurrence of α which is target of the explicit replacement on the LHS of these rules, the stack s could be duplicated or erased. This concludes our deconstruction of rule R .

► **Definition 7. Meaningful replacement reduction**, written \rightarrow_{R^\bullet} , is defined by the contextual closure of the reduction rules $\{R_{-\#}^{\neq 1}, N_{\text{-lin}}, W_{\text{-lin}}, C_{\text{-lin}}\}$.

Rules M , which together with B compute canonical forms, create new explicit replacements. These explicit replacements may be rearranged using rules W and C leading to the following notion of canonical form computation:

► **Definition 8. Canonical forms** are terms in B, M, C and W -normal form. The reduction relation \rightarrow_{BMCW} is easily seen to be confluent and terminating, thus, from now on, the notation $\mathfrak{C}(o)$ stands for the (unique) $BMCW$ -normal form of an object o . It will be shown later that \mathfrak{C} -reduction on ΛM -objects corresponds to multiplicative cuts in PPNs (cf. Thm. 14).

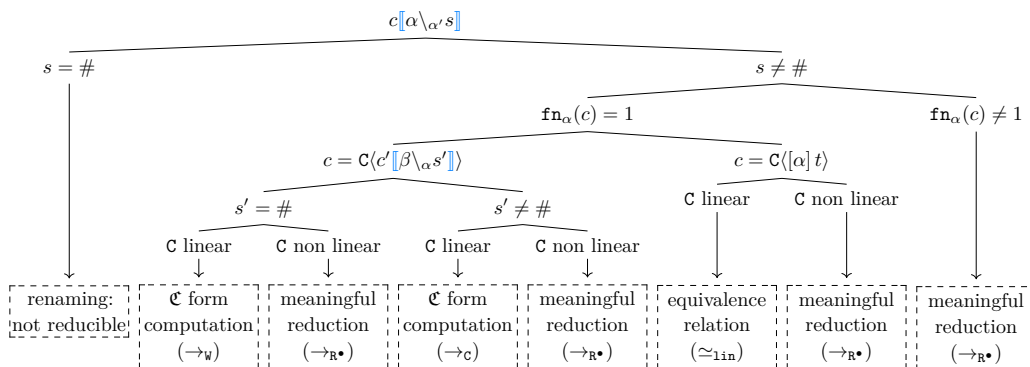
In summary, we shall define a strong bisimulation \simeq (Sec. 7) defined exclusively on canonical forms and where reduction is taken to be meaningful:

► **Definition 9. Meaningful reduction** is a relation of canonical forms defined as follows:

$$t \rightsquigarrow t' \text{ iff } t \rightarrow_{SR^\bullet} u \text{ and } t' = \mathfrak{C}(u)$$

where \rightarrow_{SR^\bullet} is $\rightarrow_S \cup \rightarrow_{R^\bullet}$. We may write \rightsquigarrow_S or $\rightsquigarrow_{R^\bullet}$ to emphasize a meaningful step corresponding to rule S or R^\bullet respectively.

The following diagram summarizes this section's findings.



4:14 Strong Bisimulation for Control Operators

$$\begin{array}{c}
\frac{}{x : A \vdash x : A \mid \emptyset} \text{(ax)} \quad \frac{\Gamma \vdash t : A \rightarrow B \mid \Delta \quad \Gamma' \vdash u : A \mid \Delta'}{\Gamma \cup \Gamma' \vdash tu : B \mid \Delta \cup \Delta'} \text{(app)} \\
\\
\frac{\Gamma, (x : A)^{\leq 1} \vdash t : B \mid \Delta}{\Gamma \vdash \lambda x. t : A \rightarrow B \mid \Delta} \text{(abs)} \quad \frac{\Gamma \vdash c \mid \Delta, (\alpha : A)^{\leq 1}}{\Gamma \vdash \mu \alpha. c : A \mid \Delta} \text{(\mu)} \quad \frac{\Gamma \vdash t : A \mid \Delta, (\alpha : A)^{\leq 1}}{\Gamma \vdash [\alpha] t \mid \Delta, \alpha : A} \text{(name)} \\
\\
\frac{\Gamma, (x : B)^{\leq 1} \vdash t : A \mid \Delta \quad \Gamma' \vdash u : B \mid \Delta'}{\Gamma \cup \Gamma' \vdash t[x \setminus u] : A \mid \Delta \cup \Delta'} \text{(sub)} \\
\\
\frac{\Gamma \vdash c \mid \Delta, (\alpha : S \rightarrow B)^{\leq 1}, (\alpha' : B)^{\leq 1} \quad \Gamma' \vdash s : S \mid \Delta', (\alpha' : B)^{\leq 1}}{\Gamma \cup \Gamma' \vdash c[[\alpha \setminus_{\alpha'} s]] \mid \Delta \cup \Delta', \alpha' : B} \text{(repl)} \\
\\
\frac{}{\emptyset \vdash \# : \epsilon \mid \emptyset} \text{(st}_h\text{)} \quad \frac{\Gamma \vdash t : A \mid \Delta \quad \Gamma' \vdash s : S \mid \Delta'}{\Gamma \cup \Gamma' \vdash t \cdot s : A \cdot S \mid \Delta \cup \Delta'} \text{(st}_t\text{)}
\end{array}$$

■ **Figure 7** Typing Rules for the ΛM -calculus.

5 Types

In this section we introduce simple types for ΛM , which extends the type system in [25] to our syntax. **Types** are generated by the following grammar:

$$\begin{array}{l}
\textbf{Term Types} \quad A ::= \iota \mid A \rightarrow B \\
\textbf{Stack Types} \quad S ::= \epsilon \mid A \cdot S
\end{array}$$

where ι is a base type. The type constructor $_ \cdot _$ should be understood as a non-commutative conjunction, which translates to a tensor in linear logic (see [7]). The arrow is right associative. We use the abbreviation $A_1 \cdot A_2 \cdot \dots \cdot A_n \cdot \epsilon \rightarrow B$ for the type $A_1 \rightarrow A_2 \dots \rightarrow A_n \rightarrow B$ (in particular, $\epsilon \rightarrow B$ is equal to B so ϵ is the left neutral element for the functional type). **Variable assignments** (Γ), are functions from variables to types; we write \emptyset for the empty variable assignment. Similarly, **name assignments** (Δ), are functions from names to types. We write $\Gamma \cup \Gamma'$ and $\Delta \cup \Delta'$ for the **compatible union** between assignments meaning that if $x \in \text{dom}(\Gamma \cap \Gamma')$ then $\Gamma(x) = \Gamma'(x)$, and similarly for Δ and Δ' . When $\text{dom}(\Gamma)$ and $\text{dom}(\Gamma')$ are disjoint we may write Γ, Γ' . The same for name assignments.

The **typing rules** are presented in Fig. 7. There are three kinds of **typing judgements**: $\Gamma \vdash t : A \mid \Delta$ for terms, $\Gamma \vdash c \mid \Delta$ for commands and $\Gamma \vdash s : S \mid \Delta$ for stacks. The notation $\Gamma, (x : A)^{\leq 1}$ (resp. $\Delta, (\alpha : A)^{\leq 1}$) is used to denote either $\Gamma, x : A$ or Γ (resp. either $\Delta, \alpha : A$ or Δ), i.e. the assumption $x : A$ occurs at most once in $\Gamma, (x : A)^{\leq 1}$. Commands have no type, cf. rules **(name)** and **(repl)**, and stacks are typed with stack types, which are heterogeneous lists, i.e. each component of the list can be typed with a different type. The interesting rule is **(repl)**, which is a logical modus ponens rule, where the fresh variable α' may be already present in the name assignment of the command c or the stack s , thus the notation $\Delta \cup \Delta', \alpha' : B$ means in particular that α' is neither in Δ nor in Δ' .

We use the abbreviation $\Gamma \vdash o : T \mid \Delta$ if $o = t$ and $T = A$, or $o = c$ and there is no type, or $o = s$ and $T = S$. We write $\pi \triangleright \Gamma \vdash o : T \mid \Delta$ if π is a type derivation concluding with $\Gamma \vdash o : T \mid \Delta$. The typing system enjoys the following properties:

► **Lemma 10 (Relevance)**. *Let $o \in \mathcal{O}(\Lambda M)$. If $\pi \triangleright \Gamma \vdash o : T \mid \Delta$, then $\text{dom}(\Gamma) = \text{fv}(o)$ and $\text{dom}(\Delta) = \text{fn}(o)$.*

► **Lemma 11** (Preservation of Types for \simeq_σ). *Let $o \in \mathcal{O}(\lambda\mu)$. If $\pi \triangleright \Gamma \vdash o : T \mid \Delta$ and $o \simeq_\sigma o'$, then there exist $\pi' \triangleright \Gamma \vdash o' : T \mid \Delta'$.*

► **Lemma 12** (Subject Reduction). *Let $o \in \mathcal{O}(\Lambda M)$ s.t. $\pi_o \triangleright \Gamma \vdash o : T \mid \Delta$. If $o \rightarrow_{\Lambda M} o'$, then there exist $\Gamma' \subseteq \Gamma$ and $\Delta' \subseteq \Delta$ and $\pi_{o'}$ s.t. $\pi_{o'} \triangleright \Gamma' \vdash o' : T \mid \Delta'$.*

Remark that free variables and names of objects decrease in the case of erasing reduction steps, as for example $(\lambda x.y)z \rightarrow y$ or $(\mu\alpha.[\gamma]x)z \rightarrow \mu\alpha.[\gamma]x$.

From now on, when $o \simeq_\sigma o'$ (resp. $o \rightarrow_{\Lambda M} o'$), we will refer to π_o and $\pi_{o'}$ as two **related** typing derivations, i.e. $\pi_{o'}$ is obtained from π_o by the proof of Lem. 11 (resp. Lem. 12).

6 Polarized Proof Nets

Laurent [22] introduced Polarized Linear Logic (LLP), a proof system based on polarities on linear logic formulae. It is equipped with a corresponding notion of Polarized Proof-Nets (PPN) which allows for a simpler correctness criterion.

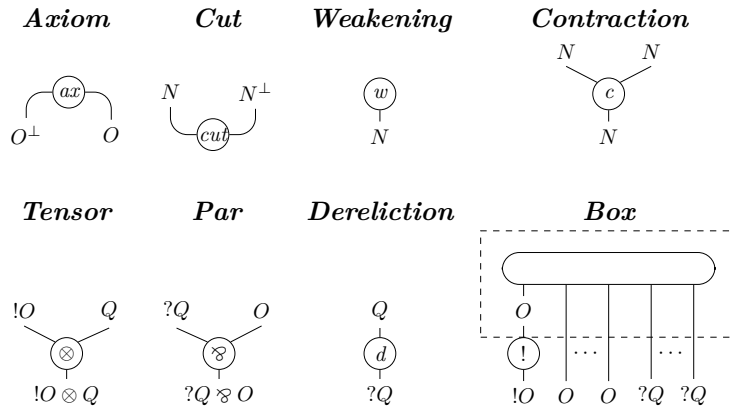
One particularly interesting feature is the translation of classical logic into LLP obtained by interpreting $A \rightarrow B$ as $!A \multimap B$, which is a straightforward extension of that from intuitionistic logic to LLP, thus capturing the translation from λ -calculus to LLP.

As mentioned in Sec. 1, it is possible to translate ΛM -objects to PPNs. More precisely, we translate typing derivations $\pi \triangleright \Gamma \vdash o : T \mid \Delta$. The translations of T and (formulae in) Δ are called *output* formulae whereas, the translations of formulae in Γ are called *input* formulae, given that they correspond to the λ -variables. Following [23], this gives the following formulae categories:

Formulae	$F ::= N \mid P$
Negative formulae	$N ::= O \mid ?Q$
Positive formulae	$P ::= Q \mid !O$
Output formulae	$O ::= \iota \mid ?Q \wp O$
Anti-output formulae	$Q ::= \iota^\perp \mid !O \otimes Q$

Negation is involutive ($\iota^{\perp\perp} = \iota$) with $(?Q \wp O)^\perp = !Q^\perp \otimes O^\perp$ and $(?Q)^\perp = !Q^\perp$.

► **Definition 13.** A *proof-structure* is a finite acyclic oriented graph built over the alphabet of nodes represented below (where the orientation is the top-bottom one):



Remark that each wire is labelled with a formula. In particular, conclusions in boxes have only one bang formula $!O$, all others being negative formulae.

A **polarized proof-net** (PPN) is a proof-structure satisfying a simple correctness criterion [23]. We refer the reader to *op.cit.* and simply mention that since our proof-structures are obtained from translating *typed terms*, this criterion is always met. **Structural equivalence** for PPNs, is based on a set of axioms that allow a reordering of weakening and contraction nodes. By lack of space we don't provide all the technical details here, but we refer the interested to [7] for further details on this standard relation. In the sequel, **polarized proof-net equality**, written \equiv , is always taken modulo structural equivalence.

The reduction relation for PPNs, denote by \rightarrow_{PPN} , is given by a set of *cut elimination rules*, split into *multiplicative* and *exponential* rules. By lack of space we don't include the rules here, but we refer the interested to [7] for further technical details. The major point is that only exponential cuts deal with erasure and duplication of boxes and \otimes -trees, these last ones used to interpret stacks in the term language. Exponential cuts are considered as the meaningful rules of PPNs, because of their erasure/duplication power. Multiplicative cuts are confluent and terminating, and we thus use **multiplicative normal-forms** as a technical tool to define the translation of typed ΛM -objects to PPNs.

More precisely, the translation from ΛM to PPNs guiding the semantical development of our work is an extension of that introduced in [23] for $\lambda\mu$ -terms, based in turn on Girard's translation of classical formulae to linear logic. We do not give any technical detail in this abstract, formal definitions are fully developed in [7]. For the sequel, it is sufficient to keep in mind that formulae are translated to polarized linear logic, and type derivations to PPNs. We use the notation π^\diamond to denote the translation of the typing derivation π .

Without entering into details, it is worth mentioning that π^\diamond does not preserve ΛM -reduction, so that we extend it to a new one, written $_^\diamond$, in such a way that π^\diamond is the multiplicative normal-form of π^\diamond . Then, the following property holds.

► **Theorem 14.** *Let o, p be typed ΛM -objects. If $o \rightarrow_{\Lambda M} p$, and π_o, π_p are two related corresponding type derivations for o and p resp., then $\pi_o^\diamond \rightarrow_{\text{PPN}} \pi_p^\diamond$.*

Proof. By induction on $\rightarrow_{\Lambda M}$ by adapting Lem. 18 and 19 in [22]. More precisely, the only ΛM -reduction steps involving *exponential* rules on the PPNs side are \rightarrow_{S} and the non-linear instances of \rightarrow_{R} (called $\rightarrow_{\text{R}\bullet}$ in Sec. 4), while \rightarrow_{B} , \rightarrow_{C} and \rightarrow_{W} are translated to *multiplicative* cuts, and both \rightarrow_{M} and \rightarrow_{N} give the identity. ◀

7 Structural Equivalence for ΛM

We introduce our notion of structural equivalence for ΛM , written \simeq , breaking down the presentation into the three key tools on which we have based our development: canonical forms, linear contexts and renaming replacements. Finally, we introduce \simeq itself.

Canonical Forms. As discussed in Sec. 1, the initial intuition in defining a strong bisimulation for ΛM arises from the intuitionistic case: Regnier's equivalence \simeq_σ is not a strong bisimulation, but taking the **B-normal form** of the left and right hand sides of these equations, results in a strong bisimulation \simeq_{σ^B} on λ -terms with explicit substitutions. In the classical case, we similarly begin from Laurent's \simeq_σ relation on $\lambda\mu$ -terms and consider the canonical forms, realised by the **C-nf** (Def. 8), resulting in the relation \simeq_{σ^B} on ΛM -terms (Fig. 8). This equational theory would be the natural candidate for our strong bisimulation, but unfortunately it is not the case as we explain below.

Linear Contexts. The first three equations in Fig. 8 together with equation σ_9^B can be generalized by noting that explicit substitution commutes with *linear* contexts (cf. Def. 6),

$$\begin{array}{lcl}
(\lambda y.t)[x \setminus u] & \simeq_{\sigma_1^B} & \lambda y.t[x \setminus u] \\
(t v)[x \setminus u] & \simeq_{\sigma_2^B} & t[x \setminus u] v \\
(\mu \alpha.[\beta] u)[x \setminus v] & \simeq_{\sigma_3^B} & \mu \alpha.[\beta] u[x \setminus v] \\
[\alpha'] (\mu \alpha''. [\beta'] (\mu \beta'' . c[\beta \setminus \beta'' v])) [\alpha \setminus \alpha'' u] & \simeq_{\sigma_4^B} & [\beta'] (\mu \beta'' . [\alpha'] (\mu \alpha'' . c[\alpha \setminus \alpha'' u])) [\beta \setminus \beta'' v] \\
[\alpha'] (\mu \alpha'' . ([\beta'] \lambda x . \mu \beta . c) [\alpha \setminus \alpha'' v]) & \simeq_{\sigma_5^B} & [\beta'] \lambda x . \mu \beta . [\alpha'] (\mu \alpha'' . c[\alpha \setminus \alpha'' v]) \\
[\alpha'] \lambda x . \mu \alpha . [\beta'] \lambda y . \mu \beta . c & \simeq_{\sigma_6^B} & [\beta'] \lambda y . \mu \beta . [\alpha'] \lambda x . \mu \alpha . c \\
[\alpha] \mu \beta . c & \simeq_{\sigma_7^B} & c\{\beta \setminus \alpha\} \\
\mu \alpha . [\alpha] t & \simeq_{\sigma_8^B} & t \\
t[y \setminus v][x \setminus u] & \simeq_{\sigma_9^B} & t[x \setminus u][y \setminus v]
\end{array}$$

Conditions for the equations: $\sigma_1^B : y \notin u$, $\sigma_2^B : x \notin v$, $\sigma_3^B : \alpha \notin v, \beta \notin w, \beta'' \neq \alpha', \alpha'' \neq \beta'$, $\sigma_4^B : x \notin v, \beta \notin v, \beta'' \neq \alpha', \alpha'' \neq \beta'$, $\sigma_5^B : \alpha \notin t, \sigma_6^B : y \notin u, x \notin v$.

■ **Figure 8** A first reformulation of Laurent's \simeq_σ on ΛM -terms.

the latter being the contexts that cannot be erased, nor duplicated, i.e. in proof-net parlance, they do not lay inside a box. The same situation arises between linear contexts and explicit replacements (cf. equations σ_4^B - σ_5^B). Thus, linear contexts can be traversed by any *independent* explicit operator (substitution/replacement). Thanks to linear contexts, equations σ_1^B - σ_2^B - σ_3^B - σ_9^B and also σ_4^B - σ_5^B from Fig. 8 can be subsumed by (and hence replaced with) the commutation between linear contexts and explicit operators as specified by the following equations, which will be part of our equivalence \simeq .

$$\begin{array}{lcl}
\text{LTT}\langle v \rangle[x \setminus u] & \simeq_{\text{exs}} & \text{LTT}\langle v[x \setminus u] \rangle \\
\text{LCC}\langle c \rangle[\alpha \setminus \alpha' s] & \simeq_{\text{exr}} & \text{LCC}\langle c[\alpha \setminus \alpha' s] \rangle
\end{array}$$

The first equation is constrained by the condition $x \notin \text{LTT}$ and $\text{fc}(u, \text{LTT})$ while the second one by $\alpha \notin \text{LCC}$ and $\text{fc}(\{s, \alpha'\}, \text{LCC})$, which essentially prevent any capture of free variables/names.

Renaming Replacements. As mentioned in Sec. 4, we adapt $\simeq_{\sigma_7^B}$ by transforming implicit (meta-level) renaming into explicit replacement. The new equation becomes:

$$[\alpha] \mu \beta . c \simeq_\rho c[\beta \setminus \alpha \#]$$

and the situation of the diagram in Fig. 6 is modified as follows:

$$\begin{array}{ccc}
([\alpha] \mu \beta . c) [\alpha \setminus \alpha' s] & \simeq_\rho & c[\beta \setminus \alpha \#] [\alpha \setminus \alpha' s] \\
\mathbf{R} \cdot \downarrow & & \mathbf{R} \cdot \downarrow \\
[\alpha'] (\mu \beta . c \{\alpha \setminus \alpha' s\}) :: s & & \mathbf{C}(c[\beta \setminus \alpha \#] \{\alpha \setminus \alpha' s\}) \\
\mathbf{e} \downarrow & & \mathbf{def} \parallel \\
[\alpha'] \mu \beta' . \mathbf{C}(c \{\alpha \setminus \alpha' s\} [\beta \setminus \beta' s]) & \simeq_\rho & \mathbf{C}(c \{\alpha \setminus \alpha' s\}) [\beta \setminus \beta' s] [\beta' \setminus \alpha' \#]
\end{array}$$

Note how the behaviour of replacement over renaming replacements (cf. Def. 2) plays a key role in the bottom right corner of the previous diagram.

The Relation \simeq and Admissible Equalities. Given all these considerations, we define:

► **Definition 15.** Σ -*equivalence*, written \simeq , is a relation over terms in canonical normal form. It is defined as the smallest reflexive, symmetric and transitive relation over terms in canonical normal form, that is closed under the following axioms:

$$\begin{array}{lll}
 \text{LTT}\langle v \rangle [x \setminus u] & \simeq_{\text{exs}} & \text{LTT}\langle v [x \setminus u] \rangle & x \notin \text{LTT}, \text{fc}(u, \text{LTT}) \\
 \text{LCC}\langle c \rangle [\alpha \setminus_{\alpha'} s] & \simeq_{\text{exr}} & \text{LCC}\langle c [\alpha \setminus_{\alpha'} s] \rangle & \alpha \notin \text{LCC}, \text{fc}(\{s, \alpha'\}, \text{LCC}) \\
 ([\alpha] u) [\alpha \setminus_{\alpha'} s] & \simeq_{\text{lin}} & [\alpha'] \mathfrak{C}(u :: s) & \alpha \notin u, s \neq \# \\
 [\alpha'] \lambda x. \mu \alpha. [\beta'] \lambda y. \mu \beta. u & \simeq_{\text{pp}} & [\beta'] \lambda y. \mu \beta. [\alpha'] \lambda x. \mu \alpha. u & \alpha \neq \beta', \alpha' \neq \beta \\
 [\alpha] \mu \beta. c & \simeq_{\rho} & c [\beta \setminus_{\alpha} \#] & \\
 \mu \alpha. [\alpha] t & \simeq_{\theta} & t & \alpha \notin t
 \end{array}$$

We conclude this section by showing some interesting *admissible* \simeq -equalities. First, we state a permutation result between substitution (resp. replacement) contexts and linear term (resp. command) contexts that will be useful later in the paper:

► **Lemma 16.**

1. Let $t \in \mathcal{T}(\Lambda M)$. Then $\mathfrak{C}(\text{L}(\text{LTT}\langle t \rangle)) \simeq \mathfrak{C}(\text{LTT}\langle \text{L}\langle t \rangle \rangle)$, if $\text{bv}(\text{L}) \notin \text{LTT}$ and $\text{fc}(\text{L}, \text{LTT})$.
2. Let $c \in \mathcal{C}(\Lambda M)$. Then $\mathfrak{C}(\text{R}(\text{LCC}\langle c \rangle)) \simeq \mathfrak{C}(\text{LCC}\langle \text{R}\langle c \rangle \rangle)$, if $\text{bn}(\text{R}) \notin \text{LCC}$ and $\text{fc}(\text{R}, \text{LCC})$.

Some further admissible equations are:

1. $t[x \setminus u][y \setminus v] \simeq t[y \setminus v][x \setminus u]$, where $x \notin v$, and $y \notin u$.
2. $c[\alpha' \setminus_{\alpha} s][\beta' \setminus_{\beta} s'] \simeq c[\beta' \setminus_{\beta} s'][\alpha' \setminus_{\alpha} s]$, where $\alpha \neq \beta'$, $\beta \neq \alpha'$, $\alpha' \notin s'$ and $\beta' \notin s$.
3. $[\alpha'] \mu \alpha. [\beta'] \mu \beta. c \simeq [\beta'] \mu \beta. [\alpha'] \mu \alpha. c$.

Finally, as already mentioned in Sec. 4, *linear* μ -steps are captured by σ -equivalence [23]. In our setting, they are essentially captured by the axiom \simeq_{lin} of the \simeq -equivalence.

A last remark of this section concerns preservation of types for our equivalence:

► **Lemma 17** (Preservation of Types for \simeq). Let $o \in \mathcal{O}(\Lambda M)$. If $\pi \triangleright \Gamma \vdash o : T \mid \Delta$ and $o \simeq o'$, then there exists $\pi' \triangleright \Gamma \vdash o' : T \mid \Delta$.

As before, when $o \simeq o'$ we will refer to π_o and $\pi_{o'}$ as two **related** typing derivations.

8 Two Correspondence Results

We now show how our \simeq -equivalence relates to σ , and hence, to PPN equality modulo structural equivalence. In particular, we want to understand whether reshufflings captured by σ -equivalence may have been left out by \simeq . One such set of reshufflings are those captured by the equation σ_7^B in Fig. 8. As discussed in Sec. 4 (cf. Fig. 6), this equation breaks strong bisimulation and motivates the introduction of renaming replacements into ΛM , as well as the inclusion of equation $[\alpha] \mu \beta. c \simeq_{\rho} c[\beta \setminus_{\alpha} \#]$ into our relation \simeq . This gives us:

$$[\alpha] \mu \beta. c \simeq_{\sigma_7^B} c\{\beta \setminus_{\alpha}\} \text{ in } \lambda \mu \quad \text{vs.} \quad [\alpha] \mu \beta. c \simeq_{\rho} c[\beta \setminus_{\alpha} \#] \text{ in } \Lambda M$$

The question that arises is whether the reshufflings captured by $\simeq_{\sigma_7^B}$ are the only ones that are an obstacle to obtaining a strong bisimulation. We prove in this section that this is indeed the case. This observation is materialized by the following property (cf. Thm. 22):

$$o \simeq_{\sigma} p \text{ if and only if } \mathfrak{C}(o) \simeq_{\text{er}} \mathfrak{C}(p)$$

where \simeq_{er} is the **renaming equivalence** generated by our strong bisimulation \simeq plus the following axiom $c\{\{\beta \setminus_{\alpha} \#\}\} \simeq_{\text{ren}} c[\beta \setminus_{\alpha} \#]$, which equates the implicit renaming used in

equation σ_7^B with the renaming replacement used in equation ρ . This sheds light on the unexpected importance that renaming replacement plays in our strong bisimulation result.

The (\Rightarrow) direction of Thm. 22, stated below, is relatively straightforward to prove:

► **Lemma 18.** *If $o \simeq_\sigma p$, then $\mathfrak{C}(o) \simeq_{er} \mathfrak{C}(p)$.*

In what follows we focus on the (\Leftarrow) direction of Thm. 22. We first discuss the soundness and completeness properties of the equivalence relation \simeq_{er} . This result is based on Laurent's completeness result for σ -equivalence. Indeed, a first translation from typed $\lambda\mu$ -objects to polarized proof-nets, written $_^\circ$, is defined in [23], together with a second translation, written $_^\bullet$, which is defined as the multiplicative normal-form of $_^\circ$, where multiplicative normal-forms are obtained by reducing all the so-called multiplicative cuts.

The σ -equivalence relation on $\lambda\mu$ -terms has the remarkable property that o and p are σ -equivalent iff their proof-net (second) translation $_^\bullet$ are structurally equivalent (cf. \equiv -equivalence introduced in Sec. 6). That is, two σ -equivalent objects have the same *structural* proof-net representation modulo multiplicative cuts.

► **Theorem 19** ([23]). *Let o and p be typed $\lambda\mu$ -objects such that $o \simeq_\sigma p$ and let π_o, π_p be two related corresponding typing derivations. Then $o \simeq_\sigma p$ iff $\pi_o^\bullet \equiv \pi_p^\bullet$.*

As mentioned in Sec. 6, we have extended the translation $_^\circ$ to the new constructors of ΛM , the resulting function is written $_^\diamond$. Moreover, we have also extended the translation $_^\diamond$ to a new one, written $_^\blacklozenge$, in such a way that π^\blacklozenge is the multiplicative normal-form of π^\diamond . Since $\mathcal{O}(\lambda\mu) \subset \mathcal{O}(\Lambda M)$, then for every $\lambda\mu$ -object o we have $\pi_o^\diamond \equiv \pi_o^\circ$ and $\pi_o^\blacklozenge \equiv \pi_o^\bullet$.

Our relation \simeq , together with the equivalence \simeq_{ren} , which are both defined on our calculus ΛM , represent equivalent proof-nets as well:

► **Lemma 20** (Soundness). *Let $o, p \in \mathcal{O}(\Lambda M)$ in \mathfrak{C} -nf. If $o \simeq_{er} p$, then $\pi_o^\blacklozenge \equiv \pi_p^\blacklozenge$, where π_o and π_p are two related corresponding typing derivations.*

Moreover, \simeq_{er} -equivalent terms in \mathfrak{C} -nf have an exact correspondence with PPNs.

► **Theorem 21** (Correspondence I). *Let $o, p \in \mathcal{O}(\lambda\mu)$. Then $\mathfrak{C}(o) \simeq_{er} \mathfrak{C}(p)$ iff $\pi_{\mathfrak{C}(o)}^\blacklozenge \equiv \pi_{\mathfrak{C}(p)}^\blacklozenge$, where $\pi_{\mathfrak{C}(o)}, \pi_{\mathfrak{C}(p)}$ are two related corresponding typing derivations for $\mathfrak{C}(o)$ and $\mathfrak{C}(p)$.*

On the other hand, we have related σ -equivalence in $\lambda\mu$ to Σ -equivalence in ΛM in such a way that $o \simeq_\sigma p$ implies $\mathfrak{C}(o) \simeq_{er} \mathfrak{C}(p)$ (Lem. 18), where er is the equivalence relation generated by converting the renaming replacement into an implicit renaming. The full picture is given by the following result, stating that the converse implication also holds.

► **Theorem 22** (Correspondence II). *Let $o, p \in \mathcal{O}(\lambda\mu)$. Then $o \simeq_\sigma p$ iff $\mathfrak{C}(o) \simeq_{er} \mathfrak{C}(p)$.*

The main results of this section can be depicted in the diagram below:

$$\begin{array}{ccc}
 o \simeq_\sigma p & \xleftrightarrow{\text{Thm. 22}} & \mathfrak{C}(o) \simeq_{er} \mathfrak{C}(p) \\
 \uparrow \text{Thm. 19} & & \uparrow \text{Thm. 21} \\
 \pi_o^\bullet \equiv \pi_p^\bullet & & \pi_{\mathfrak{C}(o)}^\blacklozenge \equiv \pi_{\mathfrak{C}(p)}^\blacklozenge
 \end{array}$$

9 The Strong Bisimulation Result

As stated in Thm. 19, $\lambda\mu$ -objects that map to the same PPN (modulo structural equivalence) are captured exactly by Laurent's σ -equivalence, which may also be seen as providing a

natural relation of *reshuffling*. Unfortunately, as explained in the introduction, σ -equivalence is not a strong bisimulation. We set out to devise a new term calculus in which reshuffling can be formulated as a strong bisimulation *without changing* the PPN semantics. The relation we obtain, \simeq , is indeed a strong bisimulation over canonical forms, i.e. \simeq -equivalent canonical terms have exactly the same redexes. This is the result we present in this section. It relies crucially on our decomposition of replacement \rightarrow_R (cf. Sec. 4) into *linear* and *non-linear* replacements, the former having no computational content (i.e. structurally equivalent PPNs modulo multiplicative cuts), and thus included in our \simeq -equivalence, while the latter corresponding to exponential cut elimination steps, and thus considered as part of our meaningful reduction.

Before stating the bisimulation result, we mention some important technical lemmas:

► **Lemma 23.** *Let $o \in \mathcal{O}(\Lambda M)$. If $o \simeq o'$, then $\mathfrak{C}(\mathbb{0}\langle o \rangle) \simeq \mathfrak{C}(\mathbb{0}\langle o' \rangle)$.*

► **Lemma 24.** *Let $u, s, o \in \mathcal{O}(\Lambda M)$ be in \mathfrak{C} -nf. Assume $p \simeq p'$ and $v \simeq v'$ and $q \simeq q'$. Then,*

■ $\mathfrak{C}(p\{x\backslash u\}) \simeq \mathfrak{C}(p'\{x\backslash u\})$ and $\mathfrak{C}(o\{x\backslash v\}) \simeq \mathfrak{C}(o\{x\backslash v'\})$.

■ $\mathfrak{C}(p\{\{\gamma\backslash_{\gamma'} s\}\}) \simeq \mathfrak{C}(p'\{\{\gamma\backslash_{\gamma'} s\}\})$ and $\mathfrak{C}(o\{\{\gamma\backslash_{\gamma'} q\}\}) \simeq \mathfrak{C}(o\{\{\gamma\backslash_{\gamma'} q'\}\})$.

Proof. Uses Lem. 23. Details in [7]. ◀

We are now able to state the promised result, namely, the fact that \simeq is a strong \rightsquigarrow -bisimulation.

► **Theorem 25 (Strong Bisimulation).** *Let $o \in \mathcal{O}(\Lambda M)$. If $o \simeq p$ and $o \rightsquigarrow o'$, then $\exists p'$ s.t. $p \rightsquigarrow p'$ and $o' \simeq p'$.*

Proof. Uses all the previous lemmas of this section. See [7] for full details. ◀

10 Conclusion

This paper refines the $\lambda\mu$ -calculus by splitting its rules into multiplicative and exponential fragments. This new presentation of $\lambda\mu$ allows to reformulate σ -equivalence on $\lambda\mu$ -terms as a strong bisimulation relation \simeq on the extended term language ΛM . In addition, \simeq is conservative w.r.t. σ -equivalence, and \simeq -equivalent terms share the same PPN representation.

Besides [23], which inspired this paper and has been discussed at length, we briefly mention further related work. In [1], polarized MELL are represented by proof-nets without boxes, by using the polarity information to transform explicit !-boxes into more compact structures. In [19], the $\lambda\mu$ -calculus is refined to a calculus $\lambda\mu\mathbf{x}$ with explicit operators, together with a small-step substitution/replacement operational semantics *at a distance*. At first sight $\lambda\mu\mathbf{x}$ seems to be more atomic than ΛM . However, $\lambda\mu\mathbf{x}$ forces the explicit replacements to be evaluated from left to right, as there is no mechanism of composition, and thus only replacements on named locations can be performed. Other refinements of the $\lambda\mu$ -calculus were defined in [6, 26, 28]. A further related reference is [16]. A precise correspondence is established between PPN and a typed version of the asynchronous π -calculus. Moreover, they show that Laurent's \simeq_σ corresponds exactly to structural equivalence of π -calculus processes (Prop. 1 in op.cit). In [24] Laurent and Regnier show that there is a precise correspondence between CPS translations from classical calculi (such as $\lambda\mu$) into intuitionistic ones on the one hand, and translations between LLP and LL on the other.

Besides confluence, studied in Sec. 2, it would be interesting to analyse other rewriting properties of our term language such as preservation of $\lambda\mu$ -strong normalization of the reduction relations $\rightarrow_{\Lambda M}$ and \rightsquigarrow , or confluence of \rightsquigarrow . Moreover, a reformulation of ΛM in

terms of two different syntactical operators, one for renaming replacement, and another one for stack replacements, would probably enlighten the intuitions on PPNs that have been used in this work. We have however chosen a unified syntax for explicit replacements in order to shorten the inductive cases of many of our proofs.

Another further topic would be to explore how our notion of strong bisimulation behaves on different calculi for Classical Logic, such as for example $\lambda\mu\tilde{\mu}$ [9]. Moreover, following the computational interpretation of deep inference provided by the intuitionistic atomic lambda-calculus [14], it would be interesting to investigate a classical extension and its corresponding notion of strong bisimulation. It is also natural to wonder what would be an ideal syntax for Classical Logic, that is able to capture strong bisimulation by reducing the syntactical axioms to a small and simple set of equations.

We believe the relation \rightsquigarrow is well-suited for devising a residual theory for $\lambda\mu$. That is, treating \rightsquigarrow as an orthogonal system, from a diagrammatic point of view [4], in spite of the critical pairs introduced by ρ and θ . This could, in turn, shed light on call-by-need for $\lambda\mu$ via the standard notion of neededness defined using residuals.

Finally, our notion of \simeq -equivalence could facilitate proofs of correctness between abstract machines and $\lambda\mu$ (like [3] for lambda-calculus) and help establish whether abstract machines for $\lambda\mu$ are “reasonable” [3].

References

- 1 Beniamino Accattoli. Compressing Polarized Boxes. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 428–437. IEEE Computer Society, 2013. URL: <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6570844>.
- 2 Beniamino Accattoli. Proof-Nets and the Linear Substitution Calculus. In Bernd Fischer and Tarmo Uustalu, editors, *Theoretical Aspects of Computing - ICTAC 2018 - 15th International Colloquium, Stellenbosch, South Africa, October 16-19, 2018, Proceedings*, volume 11187 of *Lecture Notes in Computer Science*, pages 37–61. Springer, 2018. doi:10.1007/978-3-030-02508-3_3.
- 3 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. Distilling abstract machines. In Johan Jeuring and Manuel M. T. Chakravarty, editors, *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming, Gothenburg, Sweden, September 1-3, 2014*, pages 363–376. ACM, 2014. doi:10.1145/2628136.2628154.
- 4 Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. A nonstandard standardization theorem. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 659–670. ACM, 2014. doi:10.1145/2535838.2535886.
- 5 Beniamino Accattoli and Delia Kesner. The Permutative λ -Calculus. In Nikolaj Bjørner and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings*, volume 7180 of *Lecture Notes in Computer Science*, pages 23–36. Springer, 2012. doi:10.1007/978-3-642-28717-6_5.
- 6 Philippe Audebaud. Explicit Substitutions for the Lambda-Mu-Calculus. Technical report, LIP, ENS Lyon, 1994.
- 7 Eduardo Bonelli, Delia Kesner, and Andrés Viso. Strong Bisimulation for Control Operators. *CoRR*, abs/1906.09370, 2019. arXiv:1906.09370.
- 8 Roberto Di Cosmo, Delia Kesner, and Emmanuel Polonowski. Proof-Nets and Explicit Substitutions. In Jerzy Tiuryn, editor, *Foundations of Software Science and Computation Structures, Third International Conference, FOSSACS 2000, Held as Part of the Joint European*

- Conferences on Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings*, volume 1784 of *Lecture Notes in Computer Science*, pages 63–81. Springer, 2000. doi:10.1007/3-540-46432-8_5.
- 9 Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In Martin Odersky and Philip Wadler, editors, *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000.*, pages 233–243. ACM, 2000. doi:10.1145/351240.351262.
 - 10 Vincent Danos and Laurent Regnier. Proof-nets and the Hilbert space. In *Workshop on Advances in Linear Logic, New York, NY, USA*, page 307–328. Cambridge University Press, 1995.
 - 11 Philippe de Groote. On the Relation between the Lambda-Mu-Calculus and the Syntactic Theory of Sequential Control. In Frank Pfenning, editor, *Logic Programming and Automated Reasoning, 5th International Conference, LPAR'94, Kiev, Ukraine, July 16-22, 1994, Proceedings*, volume 822 of *Lecture Notes in Computer Science*, pages 31–43. Springer, 1994. doi:10.1007/3-540-58216-9_27.
 - 12 Jean-Yves Girard. Linear Logic. *Theoretical Computer Science*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.
 - 13 Timothy Griffin. A Formulae-as-Types Notion of Control. 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 47–58. ACM Press, 1990. doi:10.1145/96709.96714.
 - 14 Tom Gundersen, Willem Heijltjes, and Michel Parigot. Atomic Lambda Calculus: A Typed Lambda-Calculus with Explicit Sharing. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 311–320. IEEE Computer Society, 2013.
 - 15 Thérèse Hardin. Confluence results for the strong pure categorical logic CCL; λ -calculi as subsystems of CCL. *tcs*, 65, 1989.
 - 16 Kohei Honda and Olivier Laurent. An exact correspondence between a typed pi-calculus and polarised proof-nets. *Theoretical Computer Science*, 411(22-24):2223–2238, 2010.
 - 17 Delia Kesner. A Theory of Explicit Substitutions with Safe and Full Composition. *Logical Methods in Computer Science*, 5(3), 2009. URL: <http://arxiv.org/abs/0905.2539>.
 - 18 Delia Kesner and Stéphane Lengrand. Extending the Explicit Substitution Paradigm. In Jürgen Giesl, editor, *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*, volume 3467 of *Lecture Notes in Computer Science*, pages 407–422. Springer, 2005. doi:10.1007/978-3-540-32033-3_30.
 - 19 Delia Kesner and Pierre Vial. Types as Resources for Classical Natural Deduction. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, September 3-9, 2017, Oxford, UK*, volume 84 of *LIPICs*, pages 24:1–24:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPICs.FSCD.2017.24.
 - 20 Delia Kesner and Pierre Vial. Non-idempotent types for classical calculi in natural deduction style. *Logical Methods in Computer Science*, 2019. To appear.
 - 21 Olivier Laurent. Krivine’s abstract machine and the lambda-mu calculus. URL: <https://perso.ens-lyon.fr/olivier.laurent/lmkamen.pdf>.
 - 22 Olivier Laurent. *A study of polarization in logic*. Theses, Université de la Méditerranée - Aix-Marseille II, March 2002. URL: <https://tel.archives-ouvertes.fr/tel-00007884>.
 - 23 Olivier Laurent. Polarized proof-nets and lambda- μ -calculus. *Theoretical Computer Science*, 290(1):161–188, 2003.
 - 24 Olivier Laurent and Laurent Regnier. About Translations of Classical Logic into Polarized Linear Logic. In *18th IEEE Symposium on Logic in Computer Science (LICS 2003), 22-25 June 2003, Ottawa, Canada, Proceedings*, pages 11–20. IEEE Computer Society, 2003. doi:10.1109/LICS.2003.1210040.

- 25 Michel Parigot. Classical Proofs as Programs. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Computational Logic and Proof Theory, Third Kurt Gödel Colloquium, KGC'93, Brno, Czech Republic, August 24-27, 1993, Proceedings*, volume 713 of *Lecture Notes in Computer Science*, pages 263–276. Springer, 1993. doi:10.1007/BFb0022575.
- 26 Emmanuel Polonovski. *Substitutions explicites, logique et normalisation. (Explicit substitutions, logic and normalization)*. PhD thesis, Paris Diderot University, France, 2004. URL: <https://tel.archives-ouvertes.fr/tel-00007962>.
- 27 Laurent Regnier. Une équivalence sur les lambda-termes. *Theoretical Computer Science*, 2(126):281–292, 1994.
- 28 Steffen van Bakel and Maria Grazia Vigliotti. A fully-abstract semantics of lambda-mu in the pi-calculus. In Paulo Oliva, editor, *Proceedings Fifth International Workshop on Classical Logic and Computation, CL&C 2014, Vienna, Austria, July 13, 2014.*, volume 164 of *EPTCS*, pages 33–47, 2014. doi:10.4204/EPTCS.164.3.

From Classical Proof Theory to P versus NP: a Guide to Bounded Theories

Iddo Tzameret

Royal Holloway, University of London, UK
<http://www.cs.rhul.ac.uk/home/tzameret>
Iddo.Tzameret@rhul.ac.uk

Abstract

This talk explores the question of what does logic and specifically proof theory can tell us about the fundamental hardness questions in computational complexity. We start with a brief description of the main concepts behind *bounded arithmetic* which is a family of weak formal theories of arithmetic that mirror in a precise manner the world of propositional proofs: if a statement of a given form is provable in a given bounded arithmetic theory then the same statement is suitably translated to a family of propositional formulas with short (polynomial-size) proofs in a corresponding propositional proof system.

We will proceed to describe the motivations behind the study of bounded arithmetic theories, their corresponding propositional proof systems, and how they relate to the fundamental complexity class separations and circuit lower bounds questions in computational complexity. We provide a collage of results and recent developments showing how bounded arithmetic and propositional proof complexity form a cohesive framework in which both concrete combinatorial questions about complexity as well as meta-mathematical questions about provability of statements of complexity theory itself, are studied.

Specific topics we shall mention are: (i) *The bounded reverse mathematics program* [2]: studying the weakest possible axiomatic assumptions that can prove important results in mathematics and computing (cf. [8, 4]), and the correspondence between circuit classes and theories. (ii) *The meta-mathematics of computational complexity*: what kind of reasoning power do we need in order to prove major results in complexity theory itself, and applications to complexity lower bounds (cf. [6, 7]). (iii) *Proof complexity*: the systematic treatment of propositional proofs as combinatorial and algebraic objects and their algorithmic applications (cf. [1, 5, 3]).

2012 ACM Subject Classification Theory of computation → Proof complexity; Theory of computation → Complexity theory and logic; Theory of computation → Circuit complexity; Theory of computation → Proof theory; Theory of computation → Algebraic complexity theory

Keywords and phrases Bounded arithmetic, complexity class separations, circuit complexity, proof complexity, weak theories of arithmetic, feasible mathematics

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.5

Category Invited Talk

References

- 1 Samuel Buss. Towards NP-P via Proof Complexity and Search. *Annals of Pure and Applied Logic*, 163(7):906–917, 2012.
- 2 Stephen Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. ASL Perspectives in Logic. Cambridge University Press, 2010.
- 3 Noah Fleming, Pravesh Kothari, and Toniann Pitassi. Semialgebraic Proofs and Efficient Algorithm Design. *Electronic Colloquium on Computational Complexity (ECCC)*, 26:106, 2019. URL: <https://eccc.weizmann.ac.il/report/2019/106>.
- 4 Pavel Hrubeš and Iddo Tzameret. Short Proofs for the Determinant Identities. *SIAM J. Comput.*, 44(2):340–383, 2015. (A preliminary version appeared in Proceedings of the 44th Annual ACM Symposium on the Theory of Computing (STOC'12)). doi:10.1137/130917788.



© Iddo Tzameret;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 5; pp. 5:1–5:2

Leibniz International Proceedings in Informatics



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


5:2 Guide to Bounded Theories

- 5 Tonnian Pitassi and Iddo Tzameret. Algebraic Proof Complexity: Progress, Frontiers and Challenges. *ACM SIGLOG News*, 3(3), 2016.
- 6 Alexander A. Razborov. Unprovability of lower bounds on circuit size in certain fragments of bounded arithmetic. *Izv. Ross. Akad. Nauk Ser. Mat.*, 59(1):201–224, 1995.
- 7 Rahul Santhanam and Jan Pich. Why are proof complexity lower bounds hard? In *60th Annual IEEE Symposium on Foundations of Computer Science FOCS 2019, November 9-12, 2019, Baltimore, Maryland USA*, 2019.
- 8 Iddo Tzameret and Stephen A. Cook. Uniform, integral and efficient proofs for the determinant identities. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005099.

Generalized Connectives for Multiplicative Linear Logic

Matteo Acclavio 

Samovar, UMR 5157 Télécom SudParis and CNRS, 9 rue Charles Fourier, 91011 Évry, France
<http://matteoacclavio.com/Math>

Roberto Maieli 

Mathematics & Physics Department, Roma Tre University, L.go S. L. Murialdo 1, 00146 Rome, Italy
<http://logica.uniroma3.it/maieli>
maieli@mat.uniroma3.it

Abstract

In this paper we investigate the notion of generalized connective for multiplicative linear logic. We introduce a notion of orthogonality for partitions of a finite set and we study the family of connectives which can be described by two orthogonal sets of partitions.

We prove that there is a special class of connectives that can never be decomposed by means of the multiplicative conjunction \otimes and disjunction \wp , providing an infinite family of non-decomposable connectives, called *Girard connectives*. We show that each Girard connective can be naturally described by a type (a set of partitions equal to its double-orthogonal) and its orthogonal type. In addition, one of these two types is the union of the types associated to a family of MLL-formulas in disjunctive normal form, and these formulas only differ for the cyclic permutations of their atoms.

2012 ACM Subject Classification Theory of computation \rightarrow Linear logic

Keywords and phrases Linear Logic, Partitions Sets, Proof Nets, Sequent Calculus

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.6

1 Introduction

In his seminal paper [7], Girard introduced the notion of generalized multiplicative connective for linear logic [6] expressed in terms of permutations over finite sets. This work was then improved by Danos and Regnier in [5] where permutations were replaced by the weaker structure of partitions of finite sets. In particular, the original orthogonality condition for permutations proposed by Girard is replaced by the following:

two partitions on the same finite domain are orthogonal iff the (bipartite) multigraph with vertices the blocks of the two partitions and edges between blocks sharing an element is connected and acyclic (ACC for short).

This orthogonality relation is extended to sets of partitions: two sets of partitions P and Q are orthogonal (denoted $P \perp Q$) if their elements are pairwise orthogonal (see Figure 1).

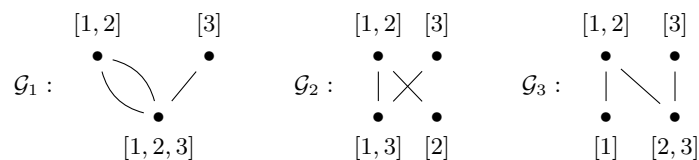


Figure 1 The two partitions $\langle [1, 2], [3] \rangle$ and $\langle [1, 2, 3] \rangle$ are not orthogonal since \mathcal{G}_1 contains a cycle. The two sets of partitions $P = \{ \langle [1, 2], [3] \rangle \}$ and $Q = \{ \langle [1, 3], [2] \rangle, \langle [1], [2, 3] \rangle \}$ are orthogonal.



© Matteo Acclavio and Roberto Maieli;
licensed under Creative Commons License CC-BY

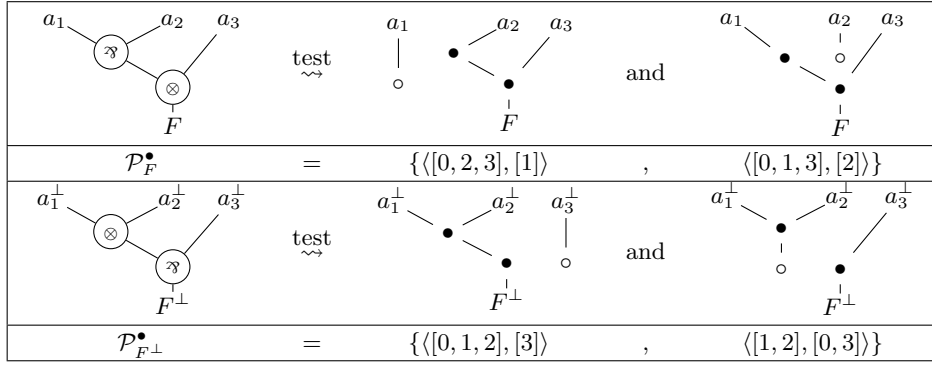
28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 6; pp. 6:1–6:16

Leibniz International Proceedings in Informatics



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



■ **Figure 2** The pretype of the formulas $F = (a_1 \wp a_2) \otimes a_3$ and $F^\perp = (a_1^\perp \otimes a_2^\perp) \wp a_3^\perp$.

Multiplicative linear logic has two well-known proof systems: sequent calculus and proof nets. Thus, we are able to associate sets of partitions to multiplicative formulas $F = F(a_1, \dots, a_n)$ by means these two syntaxes.

In the *sequential syntax*, a partition keeps the information about how the literals a_1, \dots, a_n occurring in F are gathered between its m premise sequents. In this way, we can see a (non-logical) derivation of F from a_1, \dots, a_n as a generalized m -ary rule of the sequent calculus and this rule is completely characterized by the *organization* of its premises – i.e. how premise atoms are split into sequents. This is possible because multiplicative rules are *linear*, that is conservative with respect to literals, and *unconditional*, that is context-free. By means of example, consider the following (non-logical) derivation of $F(a_1, a_2, a_3) = (a_1 \wp a_2) \otimes a_3$ and its associated generalized rule ρ :

$$\frac{\frac{a_1, a_2}{a_1 \wp a_2} \wp \quad a_3}{(a_1 \wp a_2) \otimes a_3} \otimes \rightsquigarrow \frac{a_1, a_2 \quad a_3}{F(a_1, a_2, a_3)} \rho$$

Then, the organization of F is the same of its unique associated generalized rule ρ , that is $\mathcal{O}_F = \{\langle [1, 2], [3] \rangle\} = \mathcal{O}_\rho$. However, if we consider its dual formula $F^\perp(a_1^\perp, a_2^\perp, a_3^\perp) = (a_1^\perp \otimes a_2^\perp) \wp a_3^\perp$ we observe two possible derivations associated to two possible generalized rules ρ_1 and ρ_2 and that $\mathcal{O}_{F^\perp} = \{\langle [1, 3], [2] \rangle, \langle [2, 3], [1] \rangle\} = \mathcal{O}_{\rho_1} \cup \mathcal{O}_{\rho_2}$ since $\mathcal{O}_{\rho_1} = \{\langle [1, 3], [2] \rangle\}$ and $\mathcal{O}_{\rho_2} = \{\langle [2, 3], [1] \rangle\}$. Moreover $\mathcal{O}_F \perp \mathcal{O}_{F^\perp}$.

$$\frac{\frac{a_1^\perp, a_3^\perp}{a_1^\perp \otimes a_3^\perp} \otimes \quad a_2^\perp}{(a_1^\perp \otimes a_3^\perp) \wp a_2^\perp} \wp \rightsquigarrow \frac{a_1^\perp, a_3^\perp \quad a_2^\perp}{F^\perp(a_1^\perp, a_2^\perp, a_3^\perp)} \rho_1 \quad \frac{a_2^\perp, a_3^\perp \quad a_1^\perp}{a_1^\perp \otimes a_2^\perp, a_3^\perp} \otimes \rightsquigarrow \frac{a_2^\perp, a_3^\perp \quad a_1^\perp}{F^\perp(a_1^\perp, a_2^\perp, a_3^\perp)} \rho_2$$

In the *graphical syntax* (i.e. *proof structures*), a partition keeps the information about how the premises are gathered by a *Danos-Regnier switching* [5] in the correction graph of the proof structure with premises a_1, \dots, a_n and the conclusion of a MLL-formula F (i.e. the formula tree of F). However, as already observed in [12], this construction gives another key information: not all blocks of a partition have the same statute. In fact, only one of its block is *principal*, that is, it is connected with the conclusion. To keep this information, we consider the set of *pointed partitions*, i.e. partitions over $\{0, 1, \dots, n\}$ where 0 is a marking for principal blocks of a formula $F(a_1, \dots, a_n)$: we call this set, the *pretype* of F , denoted by \mathcal{P}_F^\bullet (see Figure 2). Once we define a *forgetting function* $[-]$ erasing the occurrence of 0 in a pointed partition, we observe that $[\mathcal{P}_F^\bullet] \perp [\mathcal{P}_{F^\perp}^\bullet]$.

The sequential and the graphical way to associate a set of partitions to a MLL-formula $F(a_1, \dots, a_n)$ are in some sense orthogonal since we can show that $\mathcal{O}_F = [\mathcal{P}_F^\bullet]^\perp$.

This construction suggests a natural generalization for sequent calculus: given two sets of partitions P and Q over $\{1, \dots, n\}$ such that $P \perp Q$ and $Q = P^\perp$ (or $P = Q^\perp$), we define a pair of generalized multiplicative connectives $C = C_{(P,Q)}$ and C^\perp for which we assume given a set of sequent rules. Each rule introducing C or C^\perp has as organization a partition in P or respectively Q . Moreover, the orthogonality of P and Q assures the existence of a cut-elimination procedure.

Analogously, in proof structures syntax, given two sets of pointed partitions P^\bullet and Q^\bullet over $\{0, 1, \dots, n\}$ such that $[P^\bullet] \perp [Q^\bullet]$ and $[P^\bullet]^\perp \perp [Q^\bullet]^\perp$, we define a pair of dual multiplicative connectives satisfying cut-elimination. The information given by the pointed partitions allows us to define the Danos-Regnier switches for these connectives, since it gives us not only the information on how to gather the incoming edges of a node into blocks, but also which one of them is connected with the outgoing edge. This gives an extension of the correctness criterion for proof structures containing such connectives. Furthermore, the orthogonality of $[P^\bullet]^\perp$ and $[Q^\bullet]^\perp$ is mandatory for cut-elimination.

One natural question arises about the *decomposability* by means of \wp and \otimes :

given a pair of partitions (P, Q) describing a multiplicative connective $C_{(P,Q)}$, is it always possible to find a MLL-formula F such that $\mathcal{O}_F = P$ and $\mathcal{O}_{F^\perp} = Q$?

A preliminary negative answer to this question is given by the non-decomposable connective G_4 defined in [7] in terms of permutations, and here reported as reformulated in [5]:

$$G_4 = C_{(P,Q)} \text{ with } P = \{\langle [1, 2], [3, 4] \rangle, \langle [2, 3], [4, 1] \rangle\} \text{ and } Q = \{\langle [1, 3], [2], [4] \rangle, \langle [2, 4], [1], [3] \rangle\}$$

In [11] the second author defines an infinite family of non-decomposable connectives generalizing G_4 . Each (sequential) connective of this family is given by a set of two partitions P , called *entangled pair*¹, together with its *orthogonal* set of partitions $P^\perp = \{q \mid q \perp p \text{ for all } p \in P\}$.

In this paper we make a step further with respect to [11] by providing an infinite class of sets of partitions $\mathfrak{S}_{\langle u,v \rangle}$ enabling us to define an infinite class of non-decomposable connectives strictly including the entangled ones. We show that this set of partitions can be expressed as the union of the types of a family of formulas obtained by all the possible cyclic permutations of the literals of a formula $F(a_1, \dots, a_n) = (a_{1,1} \otimes \dots \otimes a_{1,n_1}) \wp \dots \wp (a_{k,1} \otimes \dots \otimes a_{k,n_k})$, i.e. a MLL disjunctive normal form. Besides the combinatorial nature of this property, this allows to prove that if $\mathfrak{S}_{\langle u,v \rangle} \subset P$, then any generalized connective $C_{(P,Q)}$ cannot be decomposable.

Non-decomposable connectives represent a new challenging research subject in linear logic: a denotational semantics and the geometry of interaction for the extension of MLL with these connectives are still missing. Moreover, we foresee an extension of Andreoli's paradigm of *modular proof construction*, using such connectives as additional modules [3, 10].

Structure of the paper. In Section 2 we give some backgrounds on graphs and partitions of finite sets. In particular, we provide a family of partitions satisfying a property of closure with respect to a notion of *orthogonality*. Furthermore we recall some multiplicative linear logic definitions and results in Section 3. In Section 4 we explain the correspondence between partitions sets and generalized multiplicative connectives and in Section 5 we redefine the notions of decomposable connectives in graphical and sequential syntax. Finally in Section 6 we give the family of non-decomposable connectives called *Girard connectives*.

¹ A pair of partitions, p and q , is *entangled* iff p and q have the same number of blocks and each block contains at most 2 elements of the support $\{1, \dots, n\}$.

2 Graphs and Partitions

A (direct) multigraph $\mathcal{G} = (V, E)$ is given by a set of vertices V and a multiset of edges $E = \{(u, v) | u, v \in V\}$. We denote $u \sim v$ iff there is a $(u, v) \in E$ and $u \not\sim v$ iff there is no (u, v) in E . A multigraph is *undirected* if the set of edges is reflexive, i.e. $(u, v) \in E$ iff $(v, u) \in E$. Let $u, v \in V$, then a *path from u to v* is a sequence of vertices $v_0, \dots, v_n \in V$ such that $v_i \sim v_{i+1}$ for all $i \in 0, \dots, n-1$. A multigraph is *connected* if for all $u, v \in V$ there is a path from u to v . A connected component of a graph is a maximal subset of connected vertices $V' \subset V$. A path is a *cycle* if $v_0 = v_n$. A cycle is *primitive* if $v_i \not\sim v_j$ for all $j \neq i+1$ with $i \neq 0$ and $j \neq n$. A multigraph is *acyclic* if it contains no cycles. A *graph* is a multigraph such that E is a set of edges, i.e. there is at most one edge (u, v) for each pair of vertices $u, v \in V$.

► **Theorem 1** (Euler-Poincaré invariance). *Let $\mathcal{G} = (V, E)$ be a multigraph. If $|\text{Cy}|$ and $|\text{CC}|$ are respectively the number of primitive cycles and the number of connected components of \mathcal{G} , then $|V| - |E| + |\text{Cy}| - |\text{CC}| = 0$.*

A *partition* $p = \langle \gamma_1, \dots, \gamma_u \rangle$ of a finite set $X = \{1, \dots, n\}$ is a set of subsets of X (an element of $\mathcal{P}(X)$) such that $X = \bigcup_i \gamma_i$ and if $i \neq j$ then $\gamma_i \cap \gamma_j = \emptyset$. We denote by \mathbb{P}_X the set of partitions of a finite set X and $\mathbb{P}_n = \mathbb{P}_{\{1, \dots, n\}}$. We call X the *support* of p and γ_i a *block* of p . To simplify reading, we differentiate parenthesis for partitions and blocks as follows $p = \langle [a_{1,1}, \dots, a_{1,k_1}], \dots, [a_{u,1}, \dots, a_{u,k_u}] \rangle$.

► **Definition 2** (Orthogonality). *Let $p, q \in \mathbb{P}_n$. The (undirected) graph of incidence of p and q , denoted $\mathcal{G}(p, q)$, is the multigraph with vertices the blocks of p and q such that there is an edge $v_{\gamma_1} \sim v_{\gamma_2}$ for each element in $\gamma_1 \cap \gamma_2 \neq \emptyset$. We say that p and q are orthogonal, denoted $p \perp q$, iff the induced multigraph $\mathcal{G}(p, q)$ is connected and acyclic (ACC for short).*

The notion of orthogonality extends to set of partitions: if $P, Q \subset \mathbb{P}_n$, we say that P and Q are orthogonal ($P \perp Q$) iff they are pointwise orthogonal, that is $p \perp q$ for all $p \in P$ and $q \in Q$. If $P \subset \mathbb{P}_n$, we denote $P^\perp = \{q \in \mathbb{P}_n \mid p \perp q \text{ for all } p \in P\}$ the orthogonal of P . For an example refer to Figure 1.

From Theorem 1 we deduce the following

► **Corollary 3.** *If $p, q \in P \in \mathbb{P}_n$ and $|p| \neq |q|$ then $P^\perp = \emptyset$.*

► **Definition 4** (Type). *A set of partitions $P \subset \mathbb{P}_n$ is a type iff $P = P^{\perp\perp}$.*

We here recall some results from [12] which are useful to compute the orthogonal of a set of partitions and to decide whenever a set of partitions is a type.

► **Proposition 5** (Partitions and Orthogonality). *Let $A, B \subset \mathbb{P}_n$, then the following facts hold:*

1. $A^\perp = A^{\perp\perp\perp}$. This means that A^\perp is a type;
2. $A \perp B$ iff $A \subseteq B^\perp$ and $A \perp B$ iff $B \subseteq A^\perp$;
3. $A \subseteq B$ implies $B^\perp \subseteq A^\perp$;
4. if A is a type, then there is B such that $A = B^\perp$;
5. $(\bigcup_i A_i)^\perp = \bigcap_i A_i^\perp$;
6. $(\bigcap_i A_i)^\perp \supseteq \bigcup_i A_i^\perp$;
7. if A admits a set B such that $A \perp B$ then all partitions in A have the same cardinality.

In particular, the intersection of types is always a type, while the union is not.

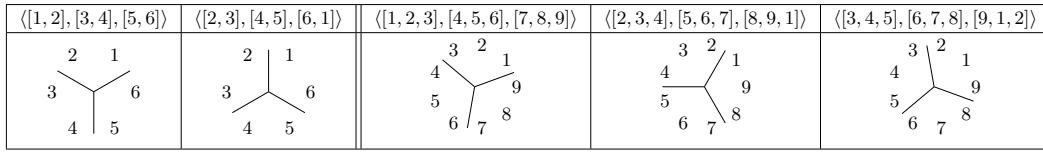


Figure 3 Examples of basic partitions and their corresponding subdivision of the cycle in n parts.

Example 6. Let $P, Q \subset \mathbb{P}_4$ be defined as

$$P = \{p_1 = \langle [1, 3], [2, 4] \rangle, p_2 = \langle [1, 4], [2, 3] \rangle\} \text{ and } Q = \{q_1 = \langle [1, 3, 4], [2] \rangle, q_2 = \langle [2, 3, 4], [1] \rangle\}$$

Then $P^\perp = \{p_1\}^\perp \cap \{p_2\}^\perp = \{\langle [3, 4], [1], [2] \rangle, \langle [1, 2], [3], [4] \rangle\}$ and $Q^\perp = \{q_1\}^\perp \cap \{q_2\}^\perp = \{\langle [1, 2], [3], [4] \rangle\}$. That is P is a type and Q is not.

Theorem 7 (No sub-type). If $T \subset \mathbb{P}_n$ is a type, then there is no type $T' \neq T$ such that $T' \subset \mathbb{P}_n$ and $T' \subset T$.

Proof. By Proposition 5.3 if $P \subset T$ then $T^\perp \subseteq P^\perp$. In particular, $T \perp P^\perp$. By Proposition 5.2 we have $P \subseteq T^{\perp\perp}$. ◀

Definition 8 (Entangled pairs of partitions [11]). A pair of partitions $P = \{p, q\} \subset \mathbb{P}_n$ with $p \neq q$ is an entangled pair if $|p| = |q|$ and $1 \leq |\gamma| \leq 2$ for each $\gamma \in p \cup q$.

By means of example, the set P and P^\perp given in Example 6 are both entangled pairs.

Theorem 9 (Entangled types [11]). Every entangled pair of partitions $P \subset \mathbb{P}_n$ is a type.

2.1 Basic Partitions

For the rest of this paper we assume $n \in \mathbb{N}$ such that $n = uv$ for some $u, v > 1$.

A basic partition of n is a partition $p \in \mathbb{P}_n$ with u blocks of v elements such that each block is of the form $[i, i + 1, \dots, i + v - 1]$, if $i + v - 1 \leq n$, or $[i, \dots, n, 1, 2, \dots, i + v - 1 - n]$ otherwise. Intuitively, if we place the elements in $\{1, \dots, n\}$ over a circle in an increasing order, a basic partition can be viewed as a subdivision of that circle into u intervals containing v elements as shown in Figure 3.

Definition 10 (Space of basic partitions). We call the space of basic partitions of rank $\langle u, v \rangle$, denoted $\mathfrak{S}_{\langle u, v \rangle}$, the set of all possible basic partitions of n made of u blocks of v elements. That is, $\mathfrak{S}_{\langle u, v \rangle} \subset \mathbb{P}_n$ is the following set

$$\left\{ \begin{array}{l} p_1 : \langle [1, \dots, v], [v + 1, \dots, 2v], \dots, [v(u - 1) + 1, \dots, n] \rangle \\ p_2 : \langle [2, \dots, v + 1], [v + 2, \dots, 2v + 1], \dots, [v(u - 1) + 2, \dots, n, 1] \rangle \\ \vdots \\ p_i : \langle [i, \dots, v + (i - 1)], [v + i, \dots, 2v + (i - 1)], \dots, [v(u - 1) + i, \dots, n, 1, \dots, i - 1] \rangle \\ \vdots \\ p_v : \langle [v, \dots, 2v - 1], [2v, \dots, 3v - 1], \dots, [n, 1, \dots, v - 1] \rangle \end{array} \right\}$$

Some examples of spaces with different rank are given in Figure 4.

Lemma 11 (Cardinality of $\mathfrak{S}_{\langle u, v \rangle}$). If $\mathfrak{S}_{\langle u, v \rangle}$ is a space of rank $\langle u, v \rangle$, then $|\mathfrak{S}_{\langle u, v \rangle}| = v$.

Proof. For each $1 \leq i \leq v$ there is a unique $p \in \mathfrak{S}_{\langle u, v \rangle}$ such that $[i, \dots, i + v - 1] \in p$. ◀

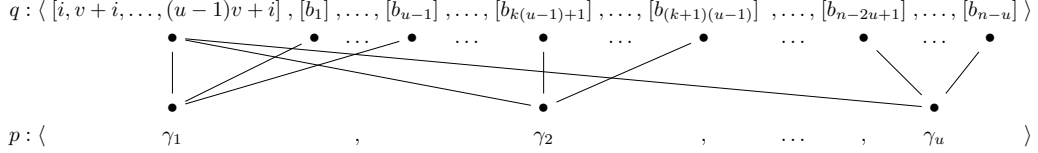
6:6 Generalized Connectives for Multiplicative Linear Logic

$$\mathfrak{S}_{\langle 3,2 \rangle} = \{ \langle [1, 2], [3, 4], [5, 6] \rangle, \langle [2, 3], [4, 5], [6, 1] \rangle \}$$

$$\mathfrak{S}_{\langle 2,3 \rangle} = \{ \langle [1, 2, 3], [4, 5, 6] \rangle, \langle [2, 3, 4], [5, 6, 1] \rangle, \langle [3, 4, 5], [6, 1, 2] \rangle \}$$

$$\mathfrak{S}_{\langle 3,3 \rangle} = \{ \langle [1, 2, 3], [4, 5, 6], [7, 8, 9] \rangle, \langle [2, 3, 4], [5, 6, 7], [8, 9, 1] \rangle, \langle [3, 4, 5], [6, 7, 8], [9, 1, 2] \rangle \}$$

■ **Figure 4** Some examples of spaces of rank $\langle u, v \rangle$.



■ **Figure 5** If $p \in \mathfrak{S}_{\langle u, v \rangle}$ and $b_i \in \{1, \dots, n\} \setminus \{i, v + 1, \dots, (u-1)v + i\}$ then $p \perp q$ for $q, p \in \mathbb{P}_n$.

► **Definition 12** (Distance). Given $1 \leq i, j \leq n$, we define the distance of i and j modulo n

$$\delta_n(i, j) = \begin{cases} \min\{j - i, i - j + n\} & \text{if } j \geq i \\ \min\{i - j, j - i + n\} & \text{if } j < i \end{cases} \quad (1)$$

E.g the distance of 1 and 9 modulo $n = 9$ is 1 that is, $\delta_9(1, 9) = \min\{8, 1\}$.

► **Lemma 13** (Distance). Let $1 \leq i, j \leq n = uv$.

- $\delta_n(i, j) < v$ iff there is $p \in \mathfrak{S}_{\langle u, v \rangle}$ containing a block γ such that $i, j \in \gamma$;
- $\delta_n(i, j) \geq v$ iff for all $p \in \mathfrak{S}_{\langle u, v \rangle}$ there are $\gamma_1 \neq \gamma_2 \in p$ such that $i \in \gamma_1, j \in \gamma_2$.

Proof. ■ Since $\delta_n(i, j) = \delta_n(j, i)$, we assume without losing generality that $i < j$. Hence, it suffices to remark that $\mathfrak{S}_{\langle u, v \rangle}$ always contains a partition including block $[i, \dots, i + v - 1]$ if $i + v - 1 \leq n$, or including block $[i, i + 1, \dots, n, 1, 2, \dots, i + v - 1 - n]$ if $i + v - 1 > n$;
 ■ By similar reasoning. ◀

► **Lemma 14.** If $\mathfrak{S}_{\langle u, v \rangle}$ is a space of basic partitions then its orthogonal $\mathfrak{S}_{\langle u, v \rangle}^\perp$ is not empty.

Proof. Let q be the partition consisting of $n - u + 1$ blocks including a block $[i = a_1, \dots, a_u]$ such that $\delta_n(a_i, a_j) = hv$ with $h \in \mathbb{N}$ for $1 < j \leq u$ (called i^{th} -block of congruence modulo v), and $n - u$ singleton blocks over $\{1, \dots, n\} \setminus \{a_1, \dots, a_u\}$.

After Lemma 13 the multigraph $\mathcal{G}(p, q)$ is acyclic, that is $|\text{Cy}| = 0$. Hence, by Theorem 1, $p \perp q$ for all $p \in \mathfrak{S}_{\langle u, v \rangle}$ (see Figure 5 for an intuition). ◀

► **Corollary 15.** All partitions of $\mathfrak{S}_{\langle u, v \rangle}^\perp$ have size $1 + n - u = 1 + u(v - 1)$.

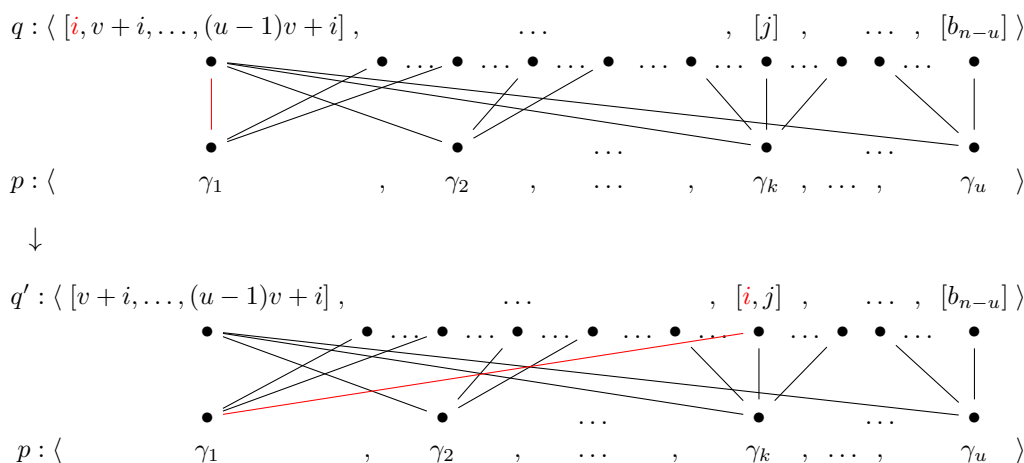
Proof. It follows Lemma 14. ◀

Moreover, by simple arithmetic argument we have the following results:

► **Lemma 16.** If $p \in \mathfrak{S}_{\langle u, v \rangle}$ and $1 \leq i, j \leq n = uv$ with $\delta_n(i, j) > v$, then there is $1 \leq k \leq n$ such that $\delta_n(i, j) = hv$ for a $h \in \mathbb{N}$ and $\delta_n(j, k) < v$. That is, for each i, j there is a k at distance a multiple of v from i which belongs to the same block of j in the partition p .

► **Proposition 17.** If $\mathfrak{S}_{\langle u, v \rangle}$ is a space of rank $\langle u, v \rangle$, then:

1. if $1 \leq i \leq n$, then there exists a partition $q \in \mathfrak{S}_{\langle u, v \rangle}^\perp$ such that $[i] \in q$;



■ **Figure 6** The permutation q including the i^{th} -block of congruence modulo v and q' are both orthogonal to $p \in \mathfrak{S}_{\langle u, v \rangle}$.

2. if $1 \leq i, j \leq n$ such that $\delta_n(i, j) \geq v$, then there is a partition $q \in \mathfrak{S}_{\langle u, v \rangle}^\perp$ containing a block γ such that $i, j \in \gamma$.

Proof. 1. By Lemma 14, given $1 \leq i, j \leq n$ such that $\delta(i, j) > v$ and $\delta(i, j) = hv$ for $h \in \mathbb{N}$, there is a partition q containing the j^{th} -block of congruence modulo v and all singleton blocks is in $\mathfrak{S}_{\langle u, v \rangle}^\perp$. Hence, in q there is the singleton block $[i]$.

2. if $\delta_n(i, j) = hv$ for a $h \in \mathbb{N}$, then we consider the partition q made of the i^{th} -block of congruence modulo v and singleton blocks.

If $\delta_n(i, j) > v$ and $\delta_n(i, j) = hv$ for a $h \in \mathbb{N}$ we define a partition q' from q by removing i from the i^{th} -block of congruence modulo v and adding i to the singleton $[j]$ as shown in Figure 6. To prove that $q' \in \mathfrak{S}_{\langle u, v \rangle}^\perp$ it suffices to use Lemma 16. In fact, we can assume that i belongs to $\gamma_i \in p \in \mathfrak{S}_{\langle u, v \rangle}$. Then there is k such that $i \in \gamma_k$ for any $p \in \mathfrak{S}_{\langle u, v \rangle}$. Since $j \neq i + hv$ with $h \in \mathbb{N}$, then $\gamma_k \neq \gamma_1$. By Theorem 1, $\mathcal{G}(q', p)$ is acyclic and connected, hence $q' \perp p$. ◀

► **Theorem 18.** Every space of basic partitions $\mathfrak{S}_{\langle u, v \rangle}$ is a type.

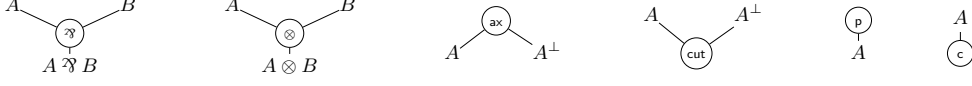
Proof. Assume by contradiction that $\mathfrak{S}_{\langle u, v \rangle}$ is not a type, i.e. assume there exists $p' \in (\mathfrak{S}_{\langle u, v \rangle}^\perp)^\perp$ such that $p' \notin \mathfrak{S}_{\langle u, v \rangle}$. By Proposition 17.1, p' cannot contain any singleton block $[i]$. Moreover, by Proposition 17.2, p' cannot contain any block γ such that $i, j \in \gamma$ and $\delta_n(i, j) \geq v$. This means that p' consists only of blocks containing elements at distance strictly smaller than v , hence $|p'| > u$. This contradicts Proposition 5.7. ◀

3 Multiplicative Linear Logic Backgrounds

We consider the class \mathcal{F} of *multiplicative linear logic formulas* (denoted by A, B, \dots) in negation normal form, generated by a countable set $\mathcal{A} = \{a, b, \dots\}$ of *propositional variables* by the grammar $A, B ::= a \mid A^\perp \mid A \wp B \mid A \otimes B$ modulo the involution of $(\cdot)^\perp$ and the *de Morgan* laws: $A^{\perp\perp} = A$, $(A \otimes B)^\perp = A^\perp \wp B^\perp$ and $(A \wp B)^\perp = A^\perp \otimes B^\perp$. A *sequent* is a set of occurrences of formulas. If $a \in \mathcal{A}$, we say that a and a^\perp are *atoms* or *atomic formulas*. The sequent system for MLL is given by the rules in Figure 7. If ρ is a sequent system rule, we call *active* a formula in a premise of a rule which is not in its conclusion and *principal* the formula introduced by the rule in the conclusion.

$$\frac{}{A, A^\perp} \text{ax} \quad \frac{\Gamma, A \quad \Delta, A^\perp}{\Gamma, \Delta} \text{cut} \quad \frac{\Gamma, A \quad \Delta, B}{\Gamma, \Delta, A \otimes B} \otimes \quad \frac{\Gamma, A, B}{\Gamma, A \wp B} \wp$$

■ **Figure 7** Standard MLL Sequent Calculus.



■ **Figure 8** Labels conditions for vertices and edges of a proof structures (also known as *links*).

► **Definition 19** (Proof Structure). A proof structure \mathfrak{P} is a direct graph with edges labeled by MLL-formulas and vertices labeled by $\{\text{ax}, \text{cut}, \otimes, \wp, \text{p}, \text{c}\}$ according to conditions of Figure 8. We call premises (conclusions) of a proof structure the nodes labeled by p (c). Moreover, abusing notation, we identify these nodes with the formula labeling the outgoing (respectively incoming) edge of these nodes. Similarly, we call premises (conclusion) of a node its incoming (outgoing) edges labels.

To each derivation \mathfrak{d} with conclusion Γ in MLL we associate the proof structure $\mathfrak{P}_{\mathfrak{d}}$ with conclusions Γ defined as follows:

- for all inference rule ρ in \mathfrak{d} there is a corresponding node in $\mathfrak{P}_{\mathfrak{d}}$ labeled by ρ having as premises the active formulas of ρ and as conclusion the principal formula of ρ ;
- for each formula in the conclusion of \mathfrak{d} there is a node in $\mathfrak{P}_{\mathfrak{d}}$ labeled by c .

► **Definition 20.** A proof structure π is a proof net if there is a derivation \mathfrak{d} in MLL such that $\pi = \mathfrak{P}_{\mathfrak{d}}$.

We characterize proof nets by means of correctness conditions on proof structures.

► **Definition 21** (Switching). A switching σ of a MLL proof structure \mathfrak{P} is a function associating to each \wp -node in \mathfrak{P} a switch, i.e. a block of the partition $\langle [1], [2] \rangle$. For each switching, we define $\sigma(\mathfrak{P})$ as the undirected correction graph (also called test) obtained by forgetting the orientation of edges and by removing, for each \wp -node with conclusion $A \wp B$, the edge labeled by B if its switch is $[1]$ or the edge labeled by A if the switch is $[2]$.

► **Theorem 22** (Danos-Regnier sequentialization [5]). For each switching σ of π , the graph $\sigma(\pi)$ is ACC iff there is a derivation \mathfrak{d} such that $\mathfrak{P} = \mathfrak{P}_{\mathfrak{d}}$.

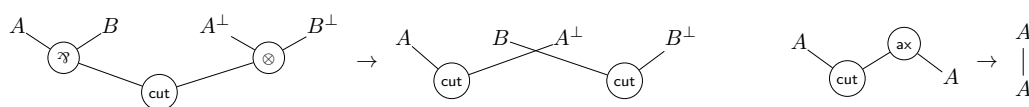
The interest of proof nets lies on the fact that they allow of identify derivations which are equivalent modulo rules permutations. This simplifies the proof of cut-elimination theorem for MLL by eliminating the bureaucracy of rules permutations during cut-elimination procedure. The *rewriting rules* for proof structures cut-elimination are given in Figure 9.

► **Theorem 23** (Danos-Regnier cut-elimination [5]). *Cut-elimination procedure for proof structures is convergent and preserves connectedness and acyclicity.*

4 Generalized multiplicative connectives and partitions sets

An *n-ary connective* is a syntactic symbol C we use to construct a new formula $\text{C}(A_1, \dots, A_n)$ from the formulas A_1, \dots, A_n in a formal grammar. By means of example, in MLL we have only the (binary) connectives \wp and \otimes . We remark that in a complete sequent calculus, each *n-ary connective* C admits at least one rule ρ with $k \leq n$ premise sequents with active formulas A_1, \dots, A_n and principal formula $\text{C}(A_1, \dots, A_n)$.

In [5] the authors define a *generalized multiplicative rule* as a sequent rule which is:



■ **Figure 9** Proof nets cut-elimination rewriting rules.

$$\frac{\vdash \Gamma_1, A_{i_1}, \dots, A_{i_k} \quad \dots \quad \vdash \Gamma_m, A_{i_h}, \dots, A_{i_n}}{\vdash \Gamma_1, \dots, \Gamma_m, C(A_1, \dots, A_n)} \rho_C \quad \mathcal{O}_\rho = \langle [i_1, \dots, i_k], \dots, [i_h, \dots, i_n] \rangle$$

■ **Figure 10** A sequential rule ρ introducing the connective C and its associate partition \mathcal{O}_ρ .

- *conservative* with respect of the atoms (or *linear*), i.e. the premises of the rule have exactly the same atoms as the conclusion;
- *unconditional*, i.e. the rule does not require information about the contexts.

As remarked in [7] and [5], these conditions allow us to associate set of partitions to multiplicative connectives of linear logic. In fact, in sequent calculus we can associate to each connective C the set of partitions describing how all the sequential rules introducing C gather the principal subformula between its premise sequents.

Similarly, by the Danos-Regnier correctness criterion, each switching of a MLL proof structure determines a partition corresponding to the premises belonging to the same connected component. However, some of the premises can never be connected to the root of a single-conclusion test of a proof net. For this reason, we prove in this paper that a set of partitions is not enough to describe a graphical connective, since each connective has to be given together with its possible switches. This additional information is provided by considering a special symbol to mark the *principal block*, i.e. the unique block selected by the switch to be connected to the conclusion.

4.1 Partitions and generalized sequential connectives

We can associate to a multiplicative rule (i.e. linear and context-free) of the sequent calculus with n active formulas a partition in \mathbb{P}_n . That is, a multiplicative m -ary rule ρ_C for a generalized n -ary connective C is completely characterized by the *organization* of its principal subformulas A_1, \dots, A_n (see Figure 10).

► **Definition 24** (Organization of a rule). *Let ρ be an m -ary rule (i.e. a rule with m premise sequents) with n active formulas A_1, \dots, A_n and principal formula $C(A_1, \dots, A_n)$. The partition $\mathcal{O}_\rho \in \mathbb{P}_n$ associated to ρ is made of m blocks defined as follows: i, j belong to a same block iff the formulas A_i and A_j belong to the same premise of ρ . We call \mathcal{O}_ρ the organization of the rules ρ .*

► **Example 25.** The organizations of the \wp -rule and the \otimes -rule are respectively $\{\langle [1, 2] \rangle\}$ and $\{\langle [1], [2] \rangle\}$. Moreover, $\{\langle [1, 2] \rangle\} \perp \{\langle [1], [2] \rangle\}$.

This allows to describe an n -ary connective by means of a set of partitions.

► **Definition 26** (Generalized sequential connective). *We says that a pair (P, Q) of non-empty sets of partitions in \mathbb{P}_n is a description of (or it describes) a sequential n -ary connective if $P \perp Q$ and if $Q = P^\perp$ or $P = Q^\perp$.*

If (P, Q) is a description of a n -ary sequential connective, we denote by $C_{(P, Q)}$ a sequential n -ary connective described by (P, Q) and by $C_{(P, Q)}^\perp = C_{(Q, P)}$ its dual connective – described by (Q, P) . We call $\mathcal{O}(C_{(P, Q)}) = P$ the organization of $C_{(P, Q)}$.

6:10 Generalized Connectives for Multiplicative Linear Logic

The organization a sequential connective $C = C_{(P,Q)}$ can be interpreted as the set of the organizations of the rules introducing C . That is, if C is a sequential connective described by (P, Q) , we can think to $\mathcal{O}(C)$ as the organizations of some rules in a two-sided calculus introducing C in the right-hand side and the set $\mathcal{O}(C^\perp)$ as the organizations of all the rules introducing C on the left-hand side (because $\mathcal{O}_C^\perp = \mathcal{O}_{C^\perp}$ is a type).

► **Remark 27.** If $C_{(P,Q)}$ is a generalized sequential connective, since $Q \neq \emptyset$, by Corollary 3 all its sequential rules have the same arity $m = |p|$ for any $p \in P$.

Let $\mathcal{C} = \{C_1, \dots, C_n\}$ be a set of multiplicative connectives, we define the generalized \mathcal{C} -multiplicative formulas $\mathcal{F}_{\mathcal{C}}$ extending \mathcal{F} with the generalized connectives in \mathcal{C} , that is, for all $C = C_{(P,Q)} \in \mathcal{C}$ with $P, Q \in \mathbb{P}_n$ we extend the grammar of MLL-formulas with $C(A_1, \dots, A_{n_i})$ and $C^\perp(A_1, \dots, A_{n_i})$. Thus, for each $p \in P$, we define a sequential rule ρ_C^p introducing the connective $C_{(P,Q)}$ such that $\mathcal{O}_{\rho_C^p} = p$ (see Figure 10). We denote $\text{MLL}(\mathcal{C})$ the extension of MLL with the sequent rules $\bigcup_{C \in \mathcal{C}} \bigcup_{p \in P} \{\rho_C^p\}$.

► **Theorem 28.** *The sequent system $\text{MLL}(\mathcal{C})$ is cut-free, that is a sequent Γ in $\mathcal{F}_{\mathcal{C}}$ is derivable in $\text{MLL}(\mathcal{C}) \cup \{\text{cut}\}$ iff it is in $\text{MLL}(\mathcal{C})$.*

Proof. The proof is given in [5]. It suffices to remark that the partitions sets describing C and its dual C^\perp describe the introduction rules for these connectives. Hence a cut-elimination step consist of replacing the cut-rule and the two rules ρ and ρ' introducing the cut-formula by cut-rules between the active formulas of ρ and ρ' . ◀

► **Example 29.** If we consider the partitions sets $P = \{\{[1, 2], [3]\}\}$ and $Q = \{\{[1, 2, 3]\}\}$, we have $P \not\perp Q$. If ρ_P and ρ'_Q , are the corresponding sequent rules, we can not define a cut-elimination step as shown below.

$$\frac{\frac{\vdash \Gamma, A_1, B_2 \quad \vdash \Delta, C_3}{\vdash \Gamma, \Delta, \rho(A_1, B_2, C_3)} \rho \quad \frac{\vdash \Gamma, A_1, B_2, C_3}{\vdash \Gamma, \rho'(A_1, B_2, C_3)} \rho'}{\frac{\frac{\vdash \Gamma, A_1, B_2 \quad \vdash \Delta, C_3}{\vdash \Gamma, \Delta, \rho(A_1, B_2, C_3)} \rho \quad \frac{\vdash \Sigma, A_1, B_2, C_3}{\vdash \Sigma, \rho'(A_1, B_2, C_3)} \rho'}{\vdash \Gamma, \Delta, \Sigma} \text{cut}}$$

4.2 Partitions and generalized graphical connectives

We associate to each correction graph of an MLL-proof structure with n premises and one single conclusion a partition in \mathbb{P}_n where each block contains the indices of connected premises. Hence, we are able to associate to each proof net a set of partitions corresponding to all its possible correction graphs where axiom nodes are replaced by pairs of premise nodes.

► **Definition 30** (Pointed partition). *A pointed partition² p^\bullet is defined as a partition of the set $\{0, k+1, \dots, n\}$ with $k \in \mathbb{N}$ such that $[0]$ is not an allowed block of p^\bullet . We denote by \mathbb{P}_n^\bullet the set of pointed partitions over the set $\{0, 1, \dots, n\}$. We define a forgetful map*

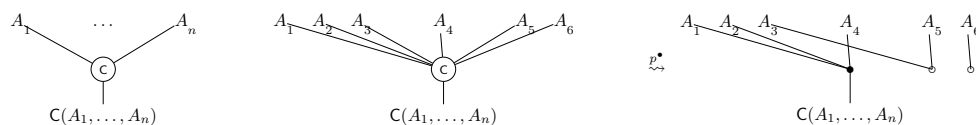
$$\lfloor - \rfloor : \mathbb{P}_{\{0, k+1, \dots, n\}} \rightarrow \mathbb{P}_{\{k+1, \dots, n\}}$$

which associates to each pointed partition p^\bullet a partition $\lfloor p^\bullet \rfloor = p$ called underlying partition of p^\bullet given by removing the element 0 from its the non-singleton block in which occurs. Similarly if p^\bullet is a set of pointed partitions we denote by $P = \lfloor p^\bullet \rfloor$ the set $\{p = \lfloor p^\bullet \rfloor \mid p^\bullet \in p^\bullet\}$.

Intuitively, we use the element 0 to mark the *principal block*, i.e. the block containing the indices of the premises which are connected to the conclusion in a test.

With this definition, we define the analogous of Definition 26 for graphical connectives.

² The name “pointed partition” is inspired by *pointed spaces* of topology, which are spaces where a specific point plays a special role.



■ **Figure 11** **On the left:** Labels conditions for generalized connectives. **On the right:** A node labeled by the $C_{(P^\bullet, Q^\bullet)}$ with $\langle [0, 1, 2, 4], [3, 5], [6] \rangle = p^\bullet \in P^\bullet$ and how this node is modified during test computation when p^\bullet is its selected switch.

► **Definition 31** (Generalized Graphical Connectives). *We say that the pair (P^\bullet, Q^\bullet) of non-empty sets of pointed partitions in \mathbb{P}_n^\bullet such that for all $0 < i \leq n$ there is a block γ such that $\{0, i\} \subset \gamma \in p^\bullet \in P^\bullet$ (respectively $\{0, i\} \subset \gamma \in q^\bullet \in Q^\bullet$) is a description of (or it describes) a graphical n -ary connective if $[P^\bullet] \perp [Q^\bullet]$ and if $[P^\bullet]^\perp \perp [Q^\bullet]^\perp$.*

We denote by $C_{(P^\bullet, Q^\bullet)}$ a graphical n -ary connective described by (P^\bullet, Q^\bullet) and by $C_{(P^\bullet, Q^\bullet)}^\perp = C_{(Q^\bullet, P^\bullet)}$ its dual connective – described by (Q^\bullet, P^\bullet) .

► **Example 32.** If $P^\bullet = \{\langle [1, 0], [2] \rangle, \langle [1], [0, 2] \rangle\}$ and $Q^\bullet = \{\langle [0, 1, 2] \rangle\}$, then \wp and \otimes are respectively described by (P^\bullet, Q^\bullet) and (Q^\bullet, P^\bullet) .

► **Definition 33** (Generalized Proof Structure). *Let $\mathcal{C} = \{C_{(P_1^\bullet, Q_1^\bullet)}, \dots, C_{(P_k^\bullet, Q_k^\bullet)}\}$ be a set of graphical n -ary connectives. An $\text{MLL}(\mathcal{C})$ proof structure is a direct graph \wp with edges labeled by $\text{MLL}(\mathcal{C})$ -formulas and vertices labeled by $\{\text{ax}, \text{cut}, \otimes, \wp, \text{p}, \text{c}\} \cup \{C, C^\perp\}_{C \in \mathcal{C}}$ satisfying conditions in Figures 8 and 11.*

As for MLL proof structure, in order to define a correctness criterion, we extend the notion of switching to graphical n -ary connectives.

► **Definition 34** (Switching). *Let \mathcal{C} be a set of generalized graphical connectives. A switching σ of a $\text{MLL}(\mathcal{C})$ proof structure \wp is a function associating to each $C_{(P^\bullet, Q^\bullet)}$ -node (i.e. a node labeled by $C_{(P^\bullet, Q^\bullet)} \in \mathcal{C}$) a switch, i.e. a pointed partition $p^\bullet \in P^\bullet$.*

Each switching σ defines an undirected graph $\sigma(\wp)$ (called correction graph or test) obtained by forgetting edge orientations and modifying each node v labeled by $C_{(P^\bullet, Q^\bullet)}$ with switch $p^\bullet \in P^\bullet$ as follows: for each block $\gamma \in p^\bullet$ with $0 \notin \gamma$, disconnect the corresponding edges targeting v and we re-link them to a fresh target node v_γ for each γ (see Figure 11).

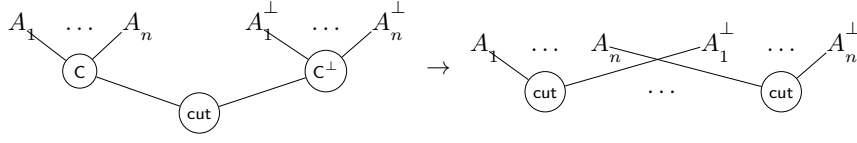
► **Definition 35** (Pretype). *Let F be a $\text{MLL}(\mathcal{C})$ formula over the atoms a_1, \dots, a_n and \wp_F be the unique proof structure with premise a_1, \dots, a_n and conclusion F , i.e. \wp_F is the formula tree of F . For each switching σ of \wp_F we define a pointed partition $p_\sigma^\bullet \in \mathbb{P}_n^\bullet$ as follows:*

- *i and j belong to the same block in p_σ^\bullet iff a_i and a_j belongs to the same connected component of $\sigma(\wp)$;*
- *i belongs in the same block 0 in p_σ^\bullet iff a_i is connected to the conclusion of $\sigma(\wp)$.*

The pretype of F is the set $\mathcal{P}_F^\bullet = \{p_\sigma^\bullet \in \mathbb{P}_n^\bullet \mid \sigma \text{ is a switching of } \wp_F\}$. We call $[P_F^\bullet]$ the Danos-Regnier pretype (or DR-pretype for short) of F . The type of F is the bi-orthogonal of DR-pretype, i.e. $\mathcal{T}_F = [P_F^\bullet]^{\perp\perp}$.

► **Definition 36** (Generalized Proof Net). *A $\text{MLL}(\mathcal{C})$ -proof structure \wp is a $\text{MLL}(\mathcal{C})$ -proof net iff for each switching σ of \wp the graph $\sigma(\wp)$ is ACC.*

The computational meaning of generalized connectives is guaranteed by the fact that the elimination of a cut-vertex linking two vertices labeled by C and C^\perp preserves the correctness criterion [5]. This follows from Definition 31 of a graphical connective: the condition $P \perp Q$ is necessary for ACC of proof structures, while the condition $P^\perp \perp Q^\perp$ is mandatory to ensure the stability of ACC under cut-elimination (see Figure 12).



■ **Figure 12** Generalized proof nets cut-elimination rewriting rule.

5 Decomposable connectives

In this section we study a notion of decomposability by means of \otimes and \wp for generalized connectives in both sequential and graphical sense. In particular, we provide a new definition of decomposability for graphical connectives which has to replace the one given in [5].

5.1 Sequential connectives

► **Definition 37** (Organization of a formula). *If $F = F(a_1, \dots, a_n)$ is a MLL-formula, we define the organization of F as the set of all partitions $p \in \mathbb{P}_n$ with $p = \langle \gamma_1, \dots, \gamma_k \rangle$ such that there is a MLL-derivation of F from the premise sequents $\{a_i\}_{i \in \gamma_1}, \dots, \{a_i\}_{i \in \gamma_k}$.*

► **Definition 38** (Decomposable sequential connectives). *A sequential connective $C_{(P,Q)}$ is s-decomposable if there is a MLL-formula F such that $P = \mathcal{O}_F$ (and $Q = \mathcal{O}_{F^\perp}$).*

► **Example 39.** Let $P = \{\langle [1, 3, 4], [2] \rangle, \langle [2, 3, 4], [1] \rangle, \langle [1, 3], [2, 4] \rangle, \langle [1, 4], [2, 3] \rangle\}$ and $Q = \{\langle [1, 2], [3], [4] \rangle\}$. Then $C_{(P,Q)}$ is s-decomposable. In fact $P = \mathcal{O}_F$ for $F = (a_1 \otimes a_2) \wp a_3 \wp a_4$.

We show in Subsection 5.3 (Corollary 48) that if C is a s-decomposable sequential connective, then \mathcal{O}_C is a type.

5.2 Graphical connectives

As for the sequential case, we define a notion of decomposability for graphical connectives.

► **Definition 40** (Decomposable graphical connectives). *A graphical n -ary connective $C_{(P^\bullet, Q^\bullet)}$ is g-decomposable iff there is a MLL formula $F(a_1, \dots, a_n)$ such that $P^\bullet = \mathcal{P}_F^\bullet$ and $Q^\bullet = \mathcal{P}_{F^\perp}^\bullet$. It is DR-decomposable if $\lfloor P^\bullet \rfloor = \lfloor \mathcal{P}_F^\bullet \rfloor$ and $\lfloor Q^\bullet \rfloor = \lfloor \mathcal{P}_{F^\perp}^\bullet \rfloor$.*

► **Lemma 41.** *If a graphical connective is not DR-decomposable then it is not g-decomposable.*

Proof. By absurd, let $C_{(P^\bullet, Q^\bullet)}$ be a g-decomposable graphical connective which is not DR-decomposable. Thus, there is a MLL formula F such that $P^\bullet = \mathcal{P}_F^\bullet$ and $Q^\bullet = \mathcal{P}_{F^\perp}^\bullet$. Then $\lfloor P^\bullet \rfloor = \lfloor \mathcal{P}_F^\bullet \rfloor$ and $\lfloor Q^\bullet \rfloor = \lfloor \mathcal{P}_{F^\perp}^\bullet \rfloor$. ◀

► **Example 42.** Let $F = (((a_1 \wp a_2) \otimes a_3) \otimes a_4) \wp a_5$ and

$$\begin{aligned} \mathcal{P}_F^\bullet &= P_1^\bullet = \{\langle [0, 1, 3, 4], [2], [5] \rangle, \langle [1, 3, 4], [2], [0, 5] \rangle, \langle [0, 2, 3, 4], [1], [5] \rangle, \langle [2, 3, 4], [1], [0, 5] \rangle\} \\ \mathcal{P}_{F^\perp}^\bullet &= Q^\bullet = \{\langle [0, 1, 2, 5], [3], [4] \rangle, \langle [0, 3, 5], [1, 2], [4] \rangle, \langle [0, 4, 5], [1, 2], [3] \rangle\}. \end{aligned}$$

Let $P_2^\bullet = P_1^\bullet \cup \{\langle [1, 3, 4], [0, 2], [5] \rangle\}$ and $C_1 = C_{(P_1^\bullet, Q^\bullet)}$ and $C_2 = C_{(P_2^\bullet, Q^\bullet)}$. Then C_1 and C_2 are both DR-decomposable, since $\lfloor P_1^\bullet \rfloor = \lfloor P_2^\bullet \rfloor = \lfloor \mathcal{P}_F^\bullet \rfloor$ and $\lfloor Q^\bullet \rfloor = \lfloor \mathcal{P}_{F^\perp}^\bullet \rfloor$, while $C_1 = C_{(P_1^\bullet, Q^\bullet)}$ is g-decomposable and $C_2 = C_{(P_2^\bullet, Q^\bullet)}$ is not g-decomposable.

► **Proposition 43** (Switchings composition). *Let $F = F(a_1, \dots, a_n)$ be a MLL-formula, then:*

1. *If $F = F_1(a_1, \dots, a_k) \wp F_2(a_{k+1}, \dots, a_n)$ and σ is a switching of \wp_F , there are $p_1^\bullet \in \mathbb{P}_{\{0,1,\dots,m\}}$ and a $p_2^\bullet \in \mathbb{P}_{\{0,m+1,\dots,n\}}$ pointed partitions associated respectively to a test of \wp_{F_1} and a test of \wp_{F_2} such that $p_\sigma^\bullet \in \mathbb{P}_n^\bullet$ is $p_\sigma^\bullet = [p_1^\bullet] \cup p_2^\bullet$ or $p_\sigma^\bullet = p_1^\bullet \cup [p_2^\bullet]$.*
2. *If $F = F_1(a_1, \dots, a_k) \otimes F_2(a_{k+1}, \dots, a_n)$ and σ is a switching of \wp_F , then there are $p_1^\bullet \in \mathbb{P}_{\{0,1,\dots,m\}}$ and a $p_2^\bullet \in \mathbb{P}_{\{0,m+1,\dots,n\}}$ pointed partitions associated respectively to a test of \wp_{F_1} and a test of \wp_{F_2} such that $p_\sigma^\bullet \in \mathbb{P}_n^\bullet$ is the pointed partition*

$$p_\sigma^\bullet = (p_1^\bullet \setminus \{\gamma_1^\bullet\}) \cup (p_2^\bullet \setminus \{\gamma_2^\bullet\}) \cup \{\gamma_1^\bullet \cup \gamma_2^\bullet\}$$

with γ_1^\bullet and γ_2^\bullet respectively the blocks of p_1^\bullet and p_2^\bullet containing 0.

3. *For all $i \in \{1, \dots, n\}$ there is a $p^\bullet \in \mathcal{P}_F^\bullet$ with $\gamma \in p^\bullet$ such that $\{0, i\} \subset \gamma$.*

Proof. 1. Since $F = F_1 \wp F_2$, then every switching σ on \wp_F is given by a switching σ_1 on \wp_{F_1} , a switching σ_2 on \wp_{F_2} and a switch for the principal \wp node. If $i, j > 0$, then $i, j \in \gamma \in p_\sigma^\bullet$ iff their corresponding premise are connected in $\sigma(\wp_F)$. Thus i, j belong to a same block iff there is a block in $p_{\sigma_1}^\bullet$ or in $p_{\sigma_2}^\bullet$ which contains both i and j .

For $j = 0$, since only one block may contain 0 accordingly with the switch of the principal \wp , i and 0 belong in the same block iff they are either in the same block in $p_{\sigma_1}^\bullet$ or in $p_{\sigma_2}^\bullet$.

2. Similarly to the previous case. It suffices to remark that if $i, j \in \gamma \in p_\sigma^\bullet$ then either i and j belong to the same block in $p_{\sigma_1}^\bullet$ or in $p_{\sigma_2}^\bullet$, or i and 0 belong to the same block in $p_{\sigma_1}^\bullet$ and j and 0 belong to the same block in $p_{\sigma_2}^\bullet$.
3. By induction over F . If $F = a$ is an atomic formula then $\mathcal{P}_F^\bullet = \{\langle [1] \rangle\}$, while if $F = F_1 \otimes F_2$ or $F = F_1 \wp F_2$ then i and 0 belong to the same block of a pointed partition in \mathcal{P}_F^\bullet iff i and 0 belong to a same block of a partition in $\mathcal{P}_{F_1}^\bullet \cup \mathcal{P}_{F_2}^\bullet$. ◀

► **Proposition 44** (Pretypes composition). *Let F be a MLL-formula.*

1. *If $F = F_1 \wp F_2$, then $p^\bullet \in \mathcal{P}_F^\bullet$ iff $p^\bullet = [p_1^\bullet] \cup p_2^\bullet$ or $p^\bullet = p_1^\bullet \cup [p_2^\bullet]$ with $p_1^\bullet \in \mathcal{P}_{F_1}^\bullet$ and $p_2^\bullet \in \mathcal{P}_{F_2}^\bullet$.*
2. *If $F = F_1 \otimes F_2$, then $p^\bullet \in \mathcal{P}_F^\bullet$ iff $p^\bullet = (p_1^\bullet \setminus \{\gamma_1^\bullet\}) \cup (p_2^\bullet \setminus \{\gamma_2^\bullet\}) \cup \{\gamma_1^\bullet \cup \gamma_2^\bullet\}$ with $p_1^\bullet \in \mathcal{P}_{F_1}^\bullet$ and $0 \in \gamma_1^\bullet \in p_1^\bullet$, and $p_2^\bullet \in \mathcal{P}_{F_2}^\bullet$ and $0 \in \gamma_2^\bullet \in p_2^\bullet$.*

Proof. It follows the constructions given in the proof of Proposition 44. ◀

► **Lemma 45.** *If $F = F_1 \wp F_2$ is a MLL formula then $|\mathcal{P}_F^\bullet| = |\mathcal{P}_{F_1}^\bullet| \cdot |\mathcal{P}_{F_2}^\bullet|$.*

Proof. Since $[p_1^\bullet \cup p_2^\bullet] = [[p_1^\bullet] \cup p_2^\bullet] = [p_1^\bullet] \cup [p_2^\bullet]$, we conclude by Proposition 44. ◀

5.3 Correspondence between sequential and graphical connectives

There is a strong link between s-decomposable sequential and g-decomposable graphical connectives as exemplified by \otimes and \wp :

$$\begin{aligned} \lfloor \{\langle [0, 1, 2] \rangle\} \rfloor^\perp &= \{\langle [1, 2] \rangle\}^\perp = \{\langle [1], [2] \rangle\} = \mathcal{O}(\otimes) \\ \lfloor \{\langle [1, 0], [2] \rangle, \langle [1], [0, 2] \rangle\} \rfloor^\perp &= \{\langle [1], [2] \rangle\}^\perp = \{\langle [1, 2] \rangle\} = \mathcal{O}(\wp) \end{aligned}$$

In fact, the two syntaxes are orthogonal views of a same decomposable connective:

► **Proposition 46** ([5]). *If $C_{(\mathcal{P}^\bullet, \mathcal{Q}^\bullet)}$ is a g-decomposable graphical connective, then there is a MLL formula F such that $\mathcal{O}(F) = \lfloor \mathcal{P}_{C_{(\mathcal{P}^\bullet, \mathcal{Q}^\bullet)}}^\bullet \rfloor^\perp$.*

► **Example 47.** If we consider the connective C given in the Example 39, $\lfloor \mathcal{P}_C^\bullet \rfloor = \lfloor \mathcal{P}_F^\bullet \rfloor = \{\langle [1, 2], [3], [4] \rangle, \langle [1, 3], [2, 4] \rangle, \langle [1, 4], [2, 3] \rangle\}$ with $F = ((a_1 \otimes a_2) \wp a_3) \wp a_4$ and $\lfloor \mathcal{P}_F^\bullet \rfloor^\perp = \mathcal{O}_F = \{\langle [1, 3, 4], [2] \rangle, \langle [2, 3, 4], [1] \rangle, \langle [1, 3], [2, 4] \rangle, \langle [1, 4], [2, 3] \rangle\}$.

► **Corollary 48.** *If \mathcal{C} is a s -decomposable sequential connective, $\mathcal{O}(\mathcal{C})$ and $\mathcal{O}(\mathcal{C}^\perp)$ are types.*

Proof. It is consequence of Propositions 5.4 and 46. ◀

6 Non-decomposable connectives

In this section we show that not all connectives are decomposable. We start by the following connective given in [7], then reformulated in [5]:

$$\mathbf{G}_4 = \mathbf{C}_{(P,Q)} \text{ with } P = \{\langle [1, 2], [3, 4] \rangle, \langle [2, 3], [4, 1] \rangle\} \text{ and } Q = \{\langle [1, 3], [2], [4] \rangle, \langle [2, 4], [1], [3] \rangle\}$$

Following [11], \mathbf{G}_4 belongs to a class of non-decomposable connectives, called *entangled*, given by two sets of partitions P and Q such that one of them is an entangled type (Definition 8).

We now define a more general class of non-decomposable connectives $\mathbf{C}_{(P,Q)}$ where $P = \mathfrak{S}_{\langle u,v \rangle}$ is a basic set of partitions. We then call such connectives *Girard connectives*. We prove that these connectives are not decomposable and that whenever $\mathfrak{S}_{\langle u,v \rangle}$ is contained in a set of partitions P then the connective $\mathbf{C}_{(P,Q)}$ is non-decomposable (for any Q).

Moreover, if \mathbf{G} is a Girard connective, the sequent $\mathbf{G}(a_1, \dots, a_n), \mathbf{G}^\perp(a_1^\perp, \dots, a_n^\perp)$ admits no η -expaded proof in $\text{MLL}(\mathcal{C})$ (this problem is known as “packaging problem”). In fact, since Girard connectives are not decomposable, this sequent is not stepwise derivable in $\text{MLL}(\mathcal{C})$. In other words, for any \mathcal{C} containing at least one non-decomposable connective, any sequent system for $\text{MLL}(\mathcal{C})$ can not be an *initial-coherent system* [13].

► **Definition 49** (Girard connectives). *If $\mathfrak{S}_{\langle u,v \rangle}$ is a space of basic partitions with u and v prime numbers, we call the sequential connective $\mathbf{C}_{\langle u,v \rangle}$ described by $(\mathfrak{S}_{\langle u,v \rangle}, \mathfrak{S}_{\langle u,v \rangle}^\perp)$ a sequential Girard connective. Moreover, we call the graphical connective $\mathbf{C}_{\langle u,v \rangle}$ described by (P^\bullet, Q^\bullet) a graphical Girard connective iff $\lfloor P^\bullet \rfloor = \mathfrak{S}_{\langle u,v \rangle}$ and $\lfloor Q^\bullet \rfloor = \mathfrak{S}_{\langle u,v \rangle}^\perp$.*

► **Theorem 50.** *Every Girard graphical connective is not DR-decomposable.*

Proof. Let $\mathbf{C}_{(P^\bullet, Q^\bullet)}$ be a Girard graphical connective. By definition this means that $\lfloor P^\bullet \rfloor = \mathfrak{S}_{\langle u,v \rangle}$ and $\lfloor Q^\bullet \rfloor = \mathfrak{S}_{\langle u,v \rangle}^\perp$. By absurd, if $\mathbf{C}_{(P^\bullet, Q^\bullet)}$ is DR-decomposable, then there is a MLL-formula F such that $\lfloor \mathcal{P}_F^\bullet \rfloor = \mathfrak{S}_{\langle u,v \rangle}$ and $\lfloor \mathcal{P}_{F^\perp}^\bullet \rfloor = \mathfrak{S}_{\langle u,v \rangle}^\perp$. Depending on F , we have three cases:

- if F is an atomic formula, then $\lfloor \mathcal{P}_F^\bullet \rfloor = \{\langle [1] \rangle\} \neq \mathfrak{S}_{\langle u,v \rangle}$ for any $u, v \in \mathbb{N}$;
- if $F = F_1 \wp F_2$, by Lemma 45, $v = |\mathfrak{S}_{\langle u,v \rangle}| = \lfloor \mathcal{P}_F^\bullet \rfloor = \lfloor \mathcal{P}_{F_1}^\bullet \rfloor \cdot \lfloor \mathcal{P}_{F_2}^\bullet \rfloor$. Since v is prime, we can assume without loss of generality that $\lfloor \mathcal{P}_{F_1}^\bullet \rfloor = \{p_1\}$, thus there is at least a block $\gamma \in p_1$ such that $\gamma \in p$ for all $p \in \lfloor \mathcal{P}_F^\bullet \rfloor$;
- if $F(a_1, \dots, a_n) = F_1 \otimes F_2$, we can assume without loss of generality that $F_1 = F_1(a_1, \dots, a_k)$ and $F_2 = F_2(a_{k+1}, \dots, a_n)$ with $k+1 > v$. Thus, by Proposition 43.3, there is a $\gamma_1 \in p_1^\bullet \in \mathcal{P}_{F_1}^\bullet$ such that $0, 1 \in \gamma_1$. Since $k+1 > v$ and $n = uv$, then there is $j \geq k+1$ such that $\delta_n(i, j) \leq v$. Moreover, by Proposition 43.3, there is a $\gamma_2 \in p_2^\bullet \in \mathcal{P}_{F_2}^\bullet$ such that $0, j \in \gamma_2$. By Proposition 43.2 we conclude that there is $\gamma \in p^\bullet \in \mathcal{P}_F^\bullet$ such that $j, i \in \gamma$, which is absurdum after Lemma 13. ◀

► **Corollary 51.** *Every graphical Girard connective is not g -decomposable.*

Proof. By Theorem 50 and Lemma 41. ◀

► **Corollary 52.** *Every Girard connective is not s -decomposable.*

► **Theorem 53** (Danos-Regnier). *Let $P = \lfloor P^\bullet \rfloor$ and $Q = \lfloor Q^\bullet \rfloor$ s.t. $P = Q^\perp$ and $Q = P^\perp$. Then a graphical connective $\mathbf{C}_{(P^\bullet, Q^\bullet)}$ is DR-decomposable iff $\mathbf{C}_{(P,Q)}$ is s -decomposable.*

G_4	G_4^\perp
$\{ \langle [0, 1, 2], [3, 4] \rangle, \langle [0, 3, 4], [1, 2] \rangle \}$ \cup $\{ \langle [0, 1, 4], [2, 3] \rangle, \langle [0, 2, 3], [1, 4] \rangle \}$	$\{ \langle [0, 1, 3], [2], [4] \rangle, \langle [0, 2, 4], [1], [3] \rangle, \langle [0, 1, 4], [2], [3] \rangle, \langle [0, 2, 3], [1], [4] \rangle \}$ \cap $\{ \langle [0, 1, 3], [2], [4] \rangle, \langle [0, 2, 4], [1], [3] \rangle, \langle [0, 1, 2], [3], [4] \rangle, \langle [0, 3, 4], [1], [2] \rangle \}$

Figure 13 The connectives $G_4 = C_{(2,2)}$ and its dual connective G_4^\perp seen respectively as the union of DNF formulas pretypes and the intersection of CNF formula pretypes.

In [11] it is showed that $P = Q^\perp$ and $Q = P^\perp$ for every sequential connective $C_{(P,Q)}$.

► **Corollary 54** (Completion of a sequential Girard connective). *Let $n = uv$ with u, v prime numbers, and P and Q non empty subsets of \mathbb{P}_n . If $C_{(P,Q)}$ is a s-decomposable sequential, then $C_{\langle u,v \rangle} \not\subseteq P$ and $C_{\langle u,v \rangle}^\perp \not\subseteq P$.*

Proof. By Theorem 18, both $\mathfrak{S}_{\langle u,v \rangle}$ and $\mathfrak{S}_{\langle u,v \rangle}^\perp$ are types. Moreover, by Proposition 46, if $C_{(P,Q)}$ is decomposable then P is a type. Then, by Theorem 7, none of $C_{\langle u,v \rangle}$ and $C_{\langle u,v \rangle}^\perp$ can be subsets of P . ◀

7 Conclusions and future works

In this paper we studied the generalized multiplicative connectives which can be described by two sets of pairwise orthogonal partitions. The orthogonality condition guarantees the definition of dual connectives for which cut-elimination is satisfied. Thus, multiplicative linear logic can be extended with these connectives preserving a computational interpretation.

We defined a notion of decomposability by means of \mathfrak{A} and \otimes for generalized connectives, with respect to both sequent calculus and proof structures syntax. We then showed the existence of connectives which are not decomposable in both senses. In particular, we exhibited the existence of an infinite family of non-decomposable connectives called Girard connectives. For such non-decomposable generalized connectives, we gave an interpretation as superposition of special decomposable generalized connectives which are connectives associated to a family of MLL disjunctive normal forms.

The class of Girard connectives strictly includes the class of non-decomposable entangled connectives, thus extending the previous work of the second author on the same subject [11]. Although the definition of a Girard connective appears to be highly combinatorial, it admits the following simple geometrical interpretation. Every Girard connective in graphical syntax can be interpreted either as the union of the pretypes of a family of DNF formulas or as the intersection of the pretypes of a family of CNF formulas having the same formula tree but differing for the cyclic permutation of their atoms/leaves (see Figure 13). Observe that cyclic permutations can help to visualize the partition associated to those connectives (see Figure 3). This interpretation is not trivial since, by Proposition 5.6, the union of pretypes is not

necessarily a type. However, these connectives have no relation with the cyclic fragment of multiplicative linear logic [1]: neither the order among blocks nor the order among the elements of each block take role in the definition.

The existence of non-decomposable multiplicative connectives which do not admit any sequentialization via the \otimes and \wp , suggests future investigations on their geometry of interaction [8], their connection to syntaxes for *concurrency* such as the π -calculus [14] and their denotational semantics [4] expanding the ideas given in [9] for syntectic connectives. Moreover, from the view point of logical programming with proof nets [3], non-decomposable graphical connectives provide additional *modules*. We foresee the use of the Girard connectives which may be interpreted as superposition of DNF for the definition of modules representing the superpositions of *bipoles* [2].

References

- 1 V. Michele Abrusci and Roberto Maieli. Proof nets for multiplicative cyclic linear logic and Lambek calculus. *Mathematical Structures in Computer Science*, 29(6):733–762, 2019. doi:10.1017/S0960129518000300.
- 2 Jean-Marc Andreoli. Focussing and proof construction. *Annals of Pure and Applied Logic*, 107(1):131–163, 2001. doi:10.1016/S0168-0072(00)00032-4.
- 3 Jean-Marc Andreoli and Laurent Mazaré. Concurrent construction of proof-nets. In *International Workshop on Computer Science Logic*, pages 29–42. Springer, 2003. doi:10.1007/978-3-540-45220-1_3.
- 4 Antonio Bucciarelli and Thomas Ehrhard. On phase semantics and denotational semantics in multiplicative–additive linear logic. *Annals of Pure and Applied Logic*, 102(3):247–282, 2000. doi:10.1016/S0168-0072(00)00056-7.
- 5 Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 28(3):181–203, 1989. doi:10.1007/BF01622878.
- 6 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987. doi:10.1016/0304-3975(87)90045-4.
- 7 Jean-Yves Girard. Multiplicatives. *Rendiconti del Seminario Matematico*, pages 11–34, 1987.
- 8 Jean-Yves Girard. On Geometry of Interaction. In Helmut Schwichtenberg, editor, *Proof and Computation*, pages 145–191, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.
- 9 Jean-Yves Girard. On the meaning of logical rules II: multiplicatives and additives. *NATO ASI Series F Computer and Systems Sciences*, 175:183–212, 2000.
- 10 Roberto Maieli. Construction of Retractable Proof Structures. In Gilles Dowek, editor, *Rewriting and Typed Lambda Calculi*, pages 319–333. Springer International Publishing, 2014. doi:10.1007/978-3-319-08918-8_22.
- 11 Roberto Maieli. Non decomposable connectives of linear logic. *Annals of Pure and Applied Logic*, 170(11):102709, 2019. doi:10.1016/j.apal.2019.05.006.
- 12 Roberto Maieli and Quintijn Puite. Modularity of proof-nets. *Archive for Mathematical Logic*, 44(2):167–193, 2005. doi:10.1007/s00153-004-0242-2.
- 13 Dale Miller and Elaine Pimentel. A formal framework for specifying sequent calculus proof systems. *Theoretical Computer Science*, 474:98–116, 2013.
- 14 Robin Milner. *Communicating and mobile systems: the Pi-calculus*. Cambridge university press, 1999.

On Free Completely Iterative Algebras

Jiří Adámek

Czech Technical University Prague, Czech Republic
J.Adamek@tu-bs.de

Abstract

For every finitary set functor F we demonstrate that free algebras carry a canonical partial order. In case F is bicontinuous, we prove that the cpo obtained as the conservative completion of the free algebra is the free completely iterative algebra. Moreover, the algebra structure of the latter is the unique continuous extension of the algebra structure of the free algebra.

For general finitary functors the free algebra and the free completely iterative algebra are proved to be posets sharing the same conservative completion. And for every recursive equation in the free completely iterative algebra the solution is obtained as the join of an ω -chain of approximate solutions in the free algebra.

2012 ACM Subject Classification Theory of computation \rightarrow Categorical semantics

Keywords and phrases free algebra, completely iterative algebra, terminal coalgebra, initial algebra, finitary functor

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.7

Related Version A full version of the paper is available at <https://arxiv.org/abs/1906.11166>.

Funding Jiří Adámek: Supported by the Grant Agency of the Czech Republic under the grant 19-00902S.

1 Introduction

Recursion and iteration belong to the crucial concepts of theoretical computer science. An algebraic treatment was suggested by Elgot who introduced iterative algebraic theories in [9]. The corresponding concept for algebras over a given endofunctor F was defined by Milius [10]: an algebra is called completely iterative if every recursive equation has a unique solution in it. We recall this in Section 5. The free completely iterative theory of Elgot is then precisely the algebraic theory corresponding to the free completely iterative algebras. Milius also described the free completely iterative algebra on a given object X : it is precisely the terminal coalgebra for the endofunctor $F(-) + X$. This corresponds nicely to the fact that the free algebra on X is precisely the initial algebra for $F(-) + X$.

In the present paper we study iterative algebras for a finitary set functor F (i.e., one preserving filtered colimits). We first show that given a choice of an element of $F\emptyset$, we obtain a canonical partial order on the initial algebra μF and on the terminal coalgebra νF . To illustrate this, consider the polynomial functor H_Σ for a finitary signature Σ : here νH_Σ is the algebra of all Σ -trees and μH_Σ the subalgebra of all finite Σ -trees. The ordering of νH_Σ is “by cutting”: for two Σ -trees s and s' we put $s < s'$ if s is obtained from s' by cutting, for a certain height, all nodes of larger heights away. This makes νH_Σ a cpo which is the conservative completion of the subposet μH_Σ . (The basic reason is that for every infinite Σ -tree its cuttings $\partial_n s$ at level $n \in \mathbb{N}$ form an ω -chain with $s = \sqcup \partial_n s$.) Now every finitary set functor can be presented as a quotient of a polynomial functor, see Section 4, and both μF and νF inherit their orders from the order of Σ -trees by cutting. We prove that

- (a) if F is bicontinuous, i.e., it also preserves limits of ω^{op} -sequences, then νF is a cpo which is the conservative completion (see Remark 8) of μF , and
- (b) for finitary set functors in general νF and μF share the same conservative completion.



© Jiří Adámek;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 7; pp. 7:1–7:21

Leibniz International Proceedings in Informatics



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

7:2 On Free Completely Iterative Algebras

Moreover, the coalgebra structure of νF is the unique continuous extension of the inverted algebra structure of μF . And for every coalgebra A the unique homomorphism into νF is a join of an ω -chain of approximate homomorphisms $h_n: A \rightarrow \mu F$. All this depends on the choice of an element in $F\emptyset$.

We then apply this to a new description of the free completely iterative algebra on an arbitrary set $X \neq \emptyset$. We choose a variable in X and obtain an order on ΦX , the free algebra for F on X , and one on ΨX , the free completely iterative algebra on X . We prove that the conservative completion of ΦX and ΨX coincide. And that in case that F is bicontinuous, ΨX is the conservative completion of ΦX . In both cases, the algebra structure of ΨX is the unique continuous extension of that of ΦX . Moreover, solutions of recursive equations in ΨX can be obtained as joins of ω -chains of so-called approximate solutions in ΦX obtained in a canonical manner.

Related Work. We can work with complete metrics in place of complete partial orders. Barr proved that given a bicontinuous set functor F with $F\emptyset \neq \emptyset$, there is a canonical complete metric on νF which is the Cauchy completion of μF , see [8]. This was extended in [2] to finitary set functors with $F\emptyset \neq \emptyset$: νF and μF have the same Cauchy completion, and the coalgebra structure of νF is the unique continuous extension of the inverted algebra structure of μF .

In the bicontinuous case a cpo structure of νF was presented in [4]. But the definition was quite technical; we recall this in Section 3. One of the main results of the present paper that the order of νF by cutting (inherited from Σ -trees) coincides with that of op. cit.

2 Polynomial Functors

We first illustrate our method on the special case: the *polynomial functor* H_Σ associated with a signature $\Sigma = (\Sigma_n)_{n \in \mathbb{N}}$. This is a set functor given by

$$H_\Sigma X = \prod_{n \in \mathbb{N}} \Sigma_n \times X^n,$$

and we represent the elements of the above set as “flat” terms $\sigma(x_1, \dots, x_n)$ where $\sigma \in \Sigma_n$ and $(x_i) \in X^n$.

► Remark 1.

(1) A free algebra $\Phi_\Sigma X$ on a set is the algebra of all terms with variables in X . This can be represented by finite trees as follows. A Σ -tree is an ordered tree labelled in Σ so that every node labelled in Σ_n has precisely n successors. We consider Σ -trees up to isomorphism. Now given a set X we form a new signature

$$\Sigma_X = \Sigma + X$$

in which elements of X have arity 0. A Σ_X -tree is called a Σ -tree over X ; its leaves are labelled by nullary symbols or variables from X . Then we get

$$\Phi_\Sigma X = \text{all finite } \Sigma\text{-trees over } X.$$

The algebra structure

$$\varphi: H_\Sigma(\Phi_\Sigma X) \rightarrow \Phi_\Sigma X$$

assigns to each member $\sigma(t_1, \dots, t_n)$ (where t_i are finite Σ_X -trees) the Σ_X -tree with root labelled by σ and with n maximum proper subtrees t_1, \dots, t_n . Thus φ^{-1} is tree tupling.

- (2) The terminal coalgebra νH_Σ can analogously be described as the coalgebra of *all* Σ -trees, the coalgebra operation is tree-tupling. For every set X we denote by Ψ_X the terminal coalgebra of H_{Σ_X} ($= H_\Sigma(-) + X$):

$$\Psi_\Sigma X = \nu H_{\Sigma_X} = \nu(H_\Sigma + X).$$

It consists of all Σ -trees over X . The coalgebra structure

$$\tau: \Psi_\Sigma X \rightarrow H_\Sigma(\Psi_\Sigma X)$$

assigns to a tree $t \in \Psi_\Sigma X$ either $x \in X$, if t is a root-only tree labelled in X , or $\sigma(t_1, \dots, t_n)$, if the root of t is labelled by $\sigma \in \Sigma_n$ and its successor subtrees are t_1, \dots, t_n . This is a free completely iterative algebra for H_Σ , see Section 5.

► **Example 2.**

- (1) If Σ consists of a set A of unary operation symbols, we have $H_\Sigma X = A \times X$. A tree in $\Psi_\Sigma X$ is either a finite unary tree over X corresponding to an element of $A^* \times X$ (a leaf labelled in X , the other nodes labelled in A) or an infinite unary tree corresponding to a word in A^ω :

$$\Psi_\Sigma X = A^* \times X + A^\omega.$$

- (2) Let Σ be a signature of one n -ary symbol for every $n \in \mathbb{N}$. Thus $H_\Sigma X = X^*$. A tree in $\Psi_\Sigma X$ does not need labels for inner nodes, and for leaves we either have a label in X or we consider the leaf unlabelled:

$$\Psi_\Sigma X = \text{all finitely branching trees with leaves partially labelled in } X.$$

► **Notation 3.** Let us choose an element $p \in X \cup \Sigma_0$. Then every tree t in $\Psi_\Sigma X$ yields a tree $\partial_n t$ of height at most n by cutting all nodes of larger heights away and relabelling all leaves of height n by p .

► **Definition 4.** We consider $\Psi_\Sigma X$ as a poset where for distinct trees s, s' we put

$$s < s' \quad \text{iff } s \text{ is a cutting of } s'.$$

That is, $s = \partial_n s'$ for some $n \in \mathbb{N}$.

► **Example 5.**

- (1) For $H_\Sigma X = A \times X$ the subset A^ω of $\Psi_\Sigma X$ is discretely ordered. Given (u, x) and (v, y) in $A^* \times X$ then

$$(u, x) < (v, y) \quad \text{iff } u \text{ is a proper prefix of } v \text{ and } x = y.$$

Finally $(u, x) < w$, for $w \in A^\omega$, iff u is a finite prefix of w and $x = p$.

- (2) For $H_\Sigma X = X^*$ the set $\Psi_\Sigma X$ is ordered by cutting.

► **Remark 6.**

- (a) Every tree s in $\Psi_\Sigma X$ is a join of its cuttings:

$$s = \bigsqcup_{n \in \mathbb{N}} \partial_n s.$$

7:4 On Free Completely Iterative Algebras

- (b) Every strictly increasing sequence $(s_n)_{n \in \mathbb{N}}$ in $\Psi_\Sigma X$ lies in $\Phi_\Sigma X$, i.e., each s_n is finite. And this sequence has a unique upper bound. Indeed, define $s \in \Phi_\Sigma X$ as follows: for every $k \in \mathbb{N}$ there exists $n \in \mathbb{N}$ such that all the trees $s_n, s_{n+1}, s_{n+2}, \dots$ agree up to height k . Then this is how s is defined up to height k .

It is easy to verify that s is a well-defined Σ -tree over X . This is obviously an upper bound: to verify $s_m < s$ for every m , one shows, for the height k of the finite tree s_m , that s_m and s agree at that height, hence $s_m = \partial_k s$. Every other upper bound s' agrees with s on heights $0, 1, 2, \dots$ – thus, $s = s'$.

- (c) Given a directed set $A \subseteq \Psi_\Sigma X$, all strictly increasing ω -chains in A have the same upper bound. Indeed, let (s_n) and (s'_n) be strictly increasing sequences in A , then since A is directed, we can find a strictly increasing sequence (s''_n) in A such that each s''_n is an upper bound of s_n and s'_n for every n . The unique upper bound of that sequence is also an upper bound for (s_n) and (s'_n) .

► **Corollary 7.** $\Psi_\Sigma X$ is a cpo, i.e., it has directed joins.

Indeed, if a directed set $A \subseteq \Psi_\Sigma X$ has a largest element, then this is $\sqcup A$. Assuming the contrary, we can find a strictly increasing sequence $s_n \in A$. If s is its upper bound, then $s = \sqcup A$. In fact, given $x \in A$, we can find a strictly increasing sequence $s'_n \geq s_n$ in A with $x \leq s'_0$ (since A is directed). Since $\sqcup s'_n$ is an upper bound of (s_n) , it follows that $\sqcup s'_n = s$. Thus, s is an upper bound of A , and it is clearly the smallest one.

► **Remark 8.**

- (1) A monotone function between posets is called *continuous* if it preserves all existing directed joins.
- (2) Recall that a *conservative completion* of a poset P is a cpo \bar{P} containing P as a subposet closed under existing directed joins with the following universal property:
For every continuous function $f: P \rightarrow Q$, where Q is a cpo, there exists a unique continuous extension $\bar{f}: \bar{P} \rightarrow Q$.
See [7], Corollary 2, for the proof that \bar{P} exists.
- (3) $\Psi_\Sigma X$ is a conservative completion of $\Phi_\Sigma X$. Indeed, given a continuous function $f: \Phi_\Sigma X \rightarrow Q$, define $\bar{f}: \Psi_\Sigma X \rightarrow Q$ by $\bar{f}(s) = \sqcup_{n \in \mathbb{N}} f(\partial_n s)$ for every tree s in $\Psi_\Sigma X$. This extends f , and the proof of Corollary 7 demonstrates that \bar{f} is continuous. It is unique: from $s = \sqcup \partial_n s$ the formula for \bar{f} follows via continuity.

3 The limit $F^\omega 1$ as a cpo

In this section F denotes a finitary set functor with $F\emptyset \neq \emptyset$. If we choose an element $p: 1 \rightarrow F\emptyset$, then the limit $F^\omega = \lim_{n \in \mathbb{N}} F^n 1$ of the terminal-coalgebra chain carries a structure of a cpo (a poset with directed joins). This cpo was presented in [4], we recall this structure here and show in the next section a more intuitive description of that cpo ordering.

► **Notation 9.**

- (1) The initial algebra is denoted by μF with the algebra structure $\varphi: F(\mu F) \rightarrow \mu F$. The terminal coalgebra is denoted by νF with the structure $\tau: \nu F \rightarrow F(\nu F)$.
- (2) For the initial object 0 (empty set) the unique morphism $i: 0 \rightarrow F0$ yields an ω -sequence of objects $F^n 0$ ($n \in \mathbb{N}$) and connecting morphisms $F^n i$ called the *initial-algebra ω -chain*. Its colimit is denoted by $F^\omega 0$ with the colimit cocone $i_n: F^n 0 \rightarrow F^\omega 0$. Since F is

finitary, $F^\omega 0$ is an initial algebra. The algebra structure $\varphi: F(F^\omega 0) \rightarrow F^\omega 0$ is the unique morphism with $\varphi \cdot F i_n = i_{n+1}$ for $n \in \mathbb{N}$. See [3].

- (3) Dually, the unique morphism $t: F1 \rightarrow 1$ yields an ω^{op} -sequence of objects $F^n 1$ ($n \in \mathbb{N}$) and connecting morphisms $F^n t$, called the *terminal coalgebra ω -chain*. Its limit is denoted by $F^\omega 1$ with the limit cone $t_n: F^\omega 1 \rightarrow F^n 1$.
- (4) The unique morphism $u: 0 \rightarrow 1$ defines morphisms $F^n u: F^n 0 \rightarrow F^n 1$. There exists a unique monomorphism $\bar{u}: F^\omega 0 \rightarrow F^\omega 1$ with $t_n \cdot \bar{u} \cdot i_n = F^n u$ ($n \in \mathbb{N}$), see [4, Lemma 2.4].
- (5) Since $p: 1 \rightarrow F0$ has been chosen, we get morphisms

$$e_n = \bar{u} \cdot i_{n+1} \cdot F^n p: F^n 1 \rightarrow F^\omega 0,$$

and we define

$$r_n = e_n \cdot t_n: F^\omega 1 \rightarrow F^\omega 0.$$

The following theorem is Theorem 3.3 in [4]. The assumption, made in that paper, that F is bicontinuous, was not used in the proof. Observe that the statement concerns the limit $F^\omega 1$ of which we do *not* claim it is νF .

► **Theorem 10.** $F^\omega 1$ is a cpo w.r.t. the following ordering

$$x \sqsubseteq y \quad \text{iff} \quad x = y \quad \text{or} \quad x = r_n(y) \quad \text{for some } n \in \mathbb{N}.$$

Every strictly increasing ω -chain has a unique upper bound in $F^\omega 1$.

► **Example 11.**

- (1) For $F = H_\Sigma$ we have $F^\omega 1 = \nu H_\Sigma$, all Σ -trees. Recall our choice of $p \in F0 = \Sigma_0$. The ordering \sqsubseteq above is precisely that by cutting, see Definition 4.

Indeed, $\bar{u}: \mu H_\Sigma \rightarrow \nu H_\Sigma$ is just the inclusion map. If we put $1 = \{p\}$, then $H_\Sigma 1$ consists of Σ -trees $\sigma(p, \dots, p)$ or $\sigma \in \Sigma_0$ of height at most 1 with leaves labelled by p . More generally, $H_\Sigma^n 1$ consists of Σ -trees of height at most n with leaves of height n labelled by p . The function $e_n: H_\Sigma^n 1 \rightarrow \mu H_\Sigma$ is the inclusion map, hence, r_n is the cutting function ∂_n of Section 2.

- (2) For the finite power-set functor \mathcal{P}_f we have $\mathcal{P}_f 0 = \{\emptyset\}$, thus the chosen element is $p = \emptyset$. Recall that a non-ordered tree is called *extensional* if for every node all maximum subtrees are pairwise distinct (i.e., non-isomorphic). Every tree has an *extensional quotient* obtained by recursively identifying equal maximum subtrees of every node.

In the initial-algebra chain, $\mathcal{P}_f^n 0$ can be described as the set of all extensional trees of height at most n (and $\mathcal{P}_f^n i$ are the inclusion maps). Hence $\mathcal{P}_f^\omega 0 = \bigcup_{n \in \mathbb{N}} \mathcal{P}_f^n 0$ is the set of all finite extensional trees.

Worrell proved that $\mathcal{P}_f^\omega 1$ can be described as the set of all compactly branching *strongly extensional* trees, see [11]. (Given a tree s , a relation R on its nodes is called a *tree bisimulation* if (a) it only relates nodes of the same height and (b) given xRy , then for every successor x' of x there is a successor y' of y with $x'Ry'$, and vice versa. A tree is called strongly extensional if every tree bisimulation is contained in the diagonal relation.)

7:6 On Free Completely Iterative Algebras

► Remark 12. Observe that each r_n factorizes through μF : we have morphisms

$$\partial_n: \nu F \rightarrow \mu F \quad \text{with} \quad r_n = \bar{u} \cdot \partial_n.$$

Indeed, put $\partial_n = i_{n+1} \cdot F^n p \cdot t_n$.

► **Notation 13** (See [3]). The *initial-algebra chain* for F beyond the above finitary iterations is the following chain indexed by all ordinals n : on objects define $F^n 0$ by $F^0 0 = 0$, $F^{n+1} 0 = F(F^n 0)$ and $F^k 0 = \text{colim}_{n < k} F^n 0$ for limit ordinals k . The connecting morphisms are denoted by $i_{n,k}: F^n 0 \rightarrow F^k 0$ ($n \leq k$). We have $i_{0,1}: 0 \rightarrow F 0$ unique, $i_{n+1,k+1} = F i_{n,k}$, and for limit ordinals k the cocone $(i_{n,k})_{n < k}$ is a colimit cocone.

Dually, the *terminal-coalgebra chain* indexed by Ord^{op} has objects $F^n 1$ with $F^0 1 = 1$, $F^{n+1} 1 = F(F^n 1)$ and $F^k 1 = \text{lim}_{k > n} F^n 1$. And it has connecting morphisms $t_{n,k}$ with $t_{1,0}$ unique, $t_{n+1,k+1} = F t_{n,k}$ and $(t_{n,k})_{k > n}$ the limit cone if k is a limit ordinal. In our notation above we thus have $t = t_{1,0}$, $F t = t_{2,1}$, etc.

► **Lemma 14.** Every natural transformation $\varepsilon: H \rightarrow F$ between endofunctors induces

(1) a unique natural transformation $\hat{\varepsilon}_n: H^n 1 \rightarrow F^n 1$ ($n \in \text{Ord}$) between their terminal-coalgebra chains satisfying

$$\hat{\varepsilon}_{n+1} \equiv H(H^n 1) \xrightarrow{\varepsilon_{H^n 1}} F(H^n 1) \xrightarrow{F \hat{\varepsilon}_n} F(F^n 1),$$

and

(2) a unique natural transformation $\tilde{\varepsilon}_n: H^n 0 \rightarrow F^n 0$ ($n \in \text{Ord}$) between their initial-algebra chains satisfying

$$\tilde{\varepsilon}_{n+1} \equiv H(H^n 0) \xrightarrow{\varepsilon_{H^n 0}} F(H^n 0) \xrightarrow{F \tilde{\varepsilon}_n} F(F^n 0).$$

Proof. We present the proof of (1), that of (2) is completely analogous.

Denote by $t_{n,k}$ and $t'_{n,k}$ the connecting morphisms of the terminal-coalgebra chains for F and H , resp.

We have $\hat{\varepsilon}_0: 1 \rightarrow 1$ unique, and $\hat{\varepsilon}_1 = \varepsilon_1: H 1 \rightarrow F 1$ is also unique. The first naturality square

$$\begin{array}{ccc} H 1 & \xrightarrow{t'_{1,0}} & 1 \\ \hat{\varepsilon}_1 \downarrow & & \downarrow \hat{\varepsilon}_0 \\ F 1 & \xrightarrow{t_{1,0}} & 1 \end{array}$$

trivially commutes.

Given $\hat{\varepsilon}_n$, then $\hat{\varepsilon}_{n+1}$ is uniquely determined by the above formula. And every naturality square for n

$$\begin{array}{ccc} H^n 1 & \xrightarrow{t'_{n,m}} & H^m 1 \\ \hat{\varepsilon}_n \downarrow & & \downarrow \hat{\varepsilon}_m \\ F^n 1 & \xrightarrow{t_{n,m}} & F^m 1 \end{array} \quad (m \leq n)$$

yields the following naturality square for $n + 1$:

$$\begin{array}{ccc}
 H^{n+1}1 & \xrightarrow{Ht'_{n,m}} & H^{m+1}1 \\
 \downarrow \hat{\varepsilon}_{n+1} & \searrow \varepsilon_{H^{n+1}} & \swarrow \varepsilon_{H^{m+1}} \\
 & F(H^n 1) & \\
 & \xrightarrow{Ft'_{n,m}} & \\
 & \swarrow F\hat{\varepsilon}_n & \searrow F\hat{\varepsilon}_m \\
 F^{n+1}1 & \xrightarrow{Ft_{n,m}} & F^{m+1}1 \\
 & \downarrow \hat{\varepsilon}_{m+1} & \\
 & &
 \end{array}$$

Indeed, the upper part commutes since $\varepsilon: H \rightarrow F$ is natural, and for the lower one apply F to the square above.

Thus, all we need proving is that given a limit ordinal k for which all the above squares with $m \leq n < k$ commute, there is a unique $\hat{\varepsilon}_k: H^k 1 \rightarrow F^k 1$ making the following squares

$$\begin{array}{ccc}
 H^k 1 & \xrightarrow{t'_{k,n}} & H^n 1 \\
 \downarrow \hat{\varepsilon}_k & & \downarrow \hat{\varepsilon}_n \\
 F^k 1 & \xrightarrow{t_{k,n}} & F^n 1
 \end{array} \quad (n < k)$$

commutative. The morphism $\hat{\varepsilon}_n \cdot t'_{k,n}$ for all $n < k$ form a cone of the k -chain with limit $F^k 1$, i.e., we have, for each $n > m$, the following commutative triangle

$$\begin{array}{ccc}
 & H^k 1 & \\
 & \swarrow t'_{k,n} & \searrow t'_{k,m} \\
 & H^n 1 & \text{---} H^m 1 \\
 & \swarrow \hat{\varepsilon}_n & \searrow \hat{\varepsilon}_m \\
 F^n 1 & \xrightarrow{t_{n,m}} & F^m 1
 \end{array}$$

Thus, $\hat{\varepsilon}_k$ is uniquely determined by the above commutative squares. ◀

► **Remark 15.** $\hat{\varepsilon}_\omega: H^\omega 1 \rightarrow F^\omega 1$ is the unique morphism satisfying $\hat{\varepsilon}_n \cdot t'_n = t_n \cdot \hat{\varepsilon}_\omega$ for every $n \in \mathbb{N}$. Indeed, this follows from the above proof since $t_n = t_{\omega,n}$ and $t'_n = t'_{\omega,n}$. Analogously, $\tilde{\varepsilon}_\omega: H^\omega 0 \rightarrow P^\omega 0$ is the unique morphism satisfying $\tilde{\varepsilon} \cdot i'_n = i_n \cdot \tilde{\varepsilon}_n$ for every $n \in \mathbb{N}$.

► **Remark 16.** Recall the description of the terminal coalgebra of a finitary set functor F due to Worrell [11]:

- (a) All connecting morphisms $t_{n,\omega}$ with $n \geq \omega$ are monic, thus, $F^{\omega+\omega} 1 = \bigcap_{n \in \mathbb{N}} F^{\omega+n} 1$;
- (b) $F^{\omega+\omega} 1$ is the terminal coalgebra whose coalgebra structure is inverse to $t_{\omega+\omega+1,\omega+\omega}$.

► **Example 17.** For \mathcal{P}_f (see 11(2)) the subset $\mathcal{P}^{\omega+n} 1$ of $\mathcal{P}^\omega 1$ consists of all strongly extensional compactly branching trees which are finitely branching at all levels up to $n - 1$. Thus, $\bigcap_{n \in \mathbb{N}} \mathcal{P}^{\omega+n} 1$ is the set $\nu \mathcal{P}_f$ of all finitely branching strongly extensional trees in $\mathcal{P}_f^\omega 1$. This was proved in [11].

7:8 On Free Completely Iterative Algebras

► **Remark 18.** Since μF can be viewed as a coalgebra for F (via φ^{-1}), we have a unique coalgebra homomorphism

$$m: \mu F \rightarrow \nu F \quad \text{with} \quad \tau \cdot m = Fm \cdot \varphi^{-1}.$$

This is monic for every finitary set functor, see [2, Proposition 5.1].

We thus can consider μF as a subset of νF and m as the inclusion map.

Since both H_Σ and F are finitary functors, we have the morphism $\tilde{\varepsilon}_\omega: \mu H_\Sigma \rightarrow \mu F$ of Lemma 14.

► **Lemma 19.** $\tilde{\varepsilon}_\omega: (\mu H_\Sigma, \varphi') \rightarrow (\mu F, \varphi \cdot \varepsilon_{\mu F})$ is a homomorphism of algebras for H_Σ . Consequently, $\tilde{\varepsilon}_\omega$ is a restriction of \hat{k} , i.e., we have $\hat{k} \cdot m' = m \cdot \tilde{\varepsilon}_\omega: \mu H_\Sigma \rightarrow \nu F$.

Proof.

(1) To verify that $\tilde{\varepsilon}_\omega$ is a homomorphism, i.e., $\tilde{\varepsilon}_\omega \cdot \varphi' = \varphi \cdot \varepsilon_{\mu F} \cdot H_\Sigma \tilde{\varepsilon}_\omega$, we use the fact that the colimit cocone $(i'_n)_{n \in \mathbb{N}}$ yields a colimit cocone $(H_\Sigma i'_n)_{n \in \mathbb{N}}$. And each $H_\Sigma i'_n$ merges the two sides of our equation:

$$\begin{aligned} \tilde{\varepsilon}_\omega \cdot \varphi' \cdot H_\Sigma i'_n &= \tilde{\varepsilon}_\omega \cdot i'_{n+1} && \text{(definition of } \varphi') \\ &= i_{n+1} \cdot \tilde{\varepsilon}_{n+1} && \text{(definition of } \tilde{\varepsilon}_\omega) \\ &= \varphi \cdot F i_n \cdot \tilde{\varepsilon}_{n+1} && \text{(definition of } \varphi) \\ &= \varphi \cdot F(i_n \cdot \tilde{\varepsilon}_n) \cdot \varepsilon_{F^n 0} && \text{(definition of } \tilde{\varepsilon}_{n+1}) \\ &= \varphi \cdot \varepsilon_{\mu F} \cdot H_\Sigma(i_n \cdot \tilde{\varepsilon}_n) && \text{(} \varepsilon \text{ natural)} \\ &= \varphi \cdot \varepsilon_{\mu F} \cdot H_\Sigma \tilde{\varepsilon}_\omega \cdot H_\Sigma i'_n && \text{(definition of } \tilde{\varepsilon}_\omega). \end{aligned}$$

(2) We observe that m and m' are homomorphisms of algebras for H_Σ . Indeed, $\tau \cdot m = Fm \cdot \varphi^{-1}$ in Remark 18 yields

$$m \cdot (\varphi \cdot \varepsilon_{\mu F}) = \tau^{-1} \cdot Fm \cdot \varepsilon_{\mu F} = (\tau^{-1} \cdot \varepsilon_{\nu F}) \cdot H_\Sigma m,$$

analogously for m' . Due to (1) this shows that $m \cdot \tilde{\varepsilon}_\omega: (\mu H_\Sigma, \varphi') \rightarrow (\nu F, \tau^{-1} \cdot \varepsilon_{\nu F})$ is a homomorphism for H_Σ . So is $\hat{k} \cdot m'$, thus the initiality of μF yields $\hat{k} \cdot m' = m \cdot \tilde{\varepsilon}_\omega$. ◀

4 The Order by Cutting

We have seen in Section 2 that for polynomial functors the terminal coalgebra νH_Σ is a cpo when ordered by cutting of the Σ -trees. In the present section we represent an arbitrary finitary set functor F as a quotient of some H_Σ . This will enable us to introduce an order by cutting on νF and μF . We then prove the following, whenever $F\emptyset \neq \emptyset$:

(a) if F is *bicontinuous*, i.e., preserves also limits of ω^{op} -chains, then νF is a cpo which is the conservative completion of μF ,

and

(b) for F in general νF and μF share the same conservative completion.

► **Definition 20.** By a presentation of a set functor F is meant a finitary signature Σ and a natural transformation $\varepsilon: H_\Sigma \rightarrow F$ with epic components.

► **Proposition 21** (See [6]). A set functor has a presentation iff it is finitary. The category of algebras for F is then equivalent to a variety of Σ -algebras.

► **Remark 22.** The proof is not difficult: a possible signature for F is $\Sigma_n = Fn$ for $n \in \mathbb{N}$. Yoneda Lemma yields a natural transformation from $\Sigma_n \times \mathbf{Set}(n, -)$ to F for every $n \in \mathbb{N}$, and this defines $\varepsilon: H_\Sigma \rightarrow F$ which is epic iff F is finitary.

Moreover, if elements of $H_\Sigma X = \coprod_{n \in \mathbb{N}} \Sigma_n \times X^n$ are represented as flat terms $\sigma(x_1, \dots, x_n)$, then we define ε -equations as equations of the following form:

$$\sigma(x_1, \dots, x_n) = \tau(y_1, \dots, y_m)$$

such that $\sigma \in \Sigma_n$, $\tau \in \Sigma_m$, and ε_X merges the given elements of $H_\Sigma X$. (Here $X = \{x_1, \dots, x_n, y_1, \dots, y_m\}$.) The variety of Σ -algebras presented by all ε -equations is equivalent to the category of F -algebras. This equivalence takes an algebra $\alpha: FA \rightarrow A$ to the Σ -algebra $\alpha \cdot \varepsilon_A: H_\Sigma A \rightarrow A$.

► **Corollary 23.** *The initial algebra μF is the quotient of the algebra μH_Σ of finite Σ -trees modulo the congruence \sim merging trees s and s' iff s can be obtained from s' by a (finite) application of ε -equations.*

► **Example 24.** The finite power-set functor \mathcal{P}_f has a presentation by the signature Σ with a unique n -ary operation for every $n \in \mathbb{N}$. Thus, $H_\Sigma X = X^*$. And we consider the natural transformation $\varepsilon_X: X^* \rightarrow \mathcal{P}_f X$ given by $(x_1 \dots x_n) \mapsto \{x_1, \dots, x_n\}$.

μH_Σ can be described as the algebra of all (unlabelled) finite trees. And two trees are congruent iff they have the same extensional quotient, see Example 11. Consequently, $\mu \mathcal{P}_f = \mu H_\Sigma / \sim$ is the set of all finite unordered extensional trees.

► **Remark 25.** Analogously to $\mu F = \mu H_\Sigma / \sim$ above, we can describe the terminal coalgebra νF as a quotient of νH_Σ , whenever a nullary symbol $p \in \Sigma_0$ is chosen, as follows. In [5, 3.13], the congruence \sim^* on νH_Σ of a possibly infinite application of ε -equations was defined as follows:

$$s \sim^* s' \quad \text{iff} \quad \partial_n s \sim \partial_n s' \quad (n \in \mathbb{N}).$$

► **Theorem 26** ([5, 3.15]). *The quotient coalgebra $\nu H_\Sigma / \sim^*$ is, when considered as an F -coalgebra, the terminal coalgebra. Shortly,*

$$\nu F = \nu H_\Sigma / \sim^* .$$

► **Remark 27.** Let $\tau': \nu H_\Sigma \rightarrow H_\Sigma(\nu H_\Sigma)$ and $\tau: \nu F \rightarrow F(\nu F)$ denote the respective coalgebra structures. The quotient map $\hat{k}: \nu H_\Sigma \rightarrow \nu F$ is a homomorphism of coalgebras for F , i.e., the following square

$$\begin{array}{ccc} \nu H_\Sigma & \xrightarrow{\tau'} & H_\Sigma(\nu H_\Sigma) \xrightarrow{\varepsilon_{\nu H_\Sigma}} F(\nu H_\Sigma) \\ \hat{k} \downarrow & & \downarrow F\hat{k} \\ \nu F & \xrightarrow{\tau} & F(\nu F) \end{array}$$

commutes. This was proved in [5], see the proof of Theorem 3.15 there (where \hat{k} was denoted by $\hat{\varepsilon}$).

► **Lemma 28.** *The morphism $\hat{k}: \nu H_\Sigma \rightarrow \nu F$ is a split epimorphism.*

7:10 On Free Completely Iterative Algebras

Proof. Choose $b: F(\nu F) \rightarrow H_\Sigma(\nu F)$ with $\varepsilon_{\nu F} \cdot b = \text{id}$. For the coalgebra $b \cdot \tau: \nu F \rightarrow H_\Sigma(\nu F)$ we have a unique homomorphism $k^*: \nu F \rightarrow \nu H_\Sigma$ with $\tau' \cdot k^* = H_\Sigma k^* \cdot (b \cdot \tau)$. We prove $\hat{k} \cdot k^* = \text{id}$ by verifying that $\hat{k} \cdot k^*$ is an endomorphism of the terminal coalgebra νF , i.e., $\tau \cdot (\hat{k} \cdot k^*) = F(\hat{k} \cdot k^*) \cdot \tau$:

$$\begin{aligned}
\tau \cdot \hat{k} \cdot k^* &= F\hat{k} \cdot \varepsilon_{\nu H_\Sigma} \cdot \tau' \cdot k^* && (\hat{k} \text{ a homomorphism}) \\
&= Fk \cdot \varepsilon_{\nu H_\Sigma} \cdot H_\Sigma k^* \cdot b \cdot \tau && (k^* \text{ a homomorphism}) \\
&= F(k \cdot k^*) \cdot \varepsilon_{\nu F} \cdot b \cdot \tau && (\varepsilon \text{ natural}) \\
&= F(k \cdot k^*) \cdot \tau && (\varepsilon_{\nu F} \cdot b = \text{id}) \quad \blacktriangleleft
\end{aligned}$$

► **Definition 29.** The following relation \leq on νF is called *order by cutting*: given distinct congruence classes $[s]$ and $[s']$ of \sim^* , put

$$[s] < [s'] \quad \text{iff} \quad s \sim \partial_n s' \quad \text{for some} \quad n \in \mathbb{N}.$$

We obtain posets νF and μF (as a subposet via \bar{u} see Remark 18).

► **Example 30.** For the presentation of \mathcal{P}_f of Example 24 we know that νH_Σ is the algebra of all finitely branching trees. We have $s \sim^* s'$ iff the extensional quotients of $\partial_n s$ and $\partial_n s'$ coincide for all $n \in \mathbb{N}$. This way Barr described $\nu \mathcal{P}_f$ in [8].

Consequently, for extensional trees we have $s < s'$ iff s is the extensional quotient of some cutting of s' .

► **Notation 31.** In the rest of the present section we assume that F is a finitary set functor with $F\emptyset \neq \emptyset$, and that a presentation ε is given. Since $\varepsilon_\emptyset: \Sigma_0 \rightarrow F\emptyset$ is epic, we can choose a nullary symbol p' in Σ_0 . This yields a choice of $p = \varepsilon_\emptyset(p')$ in $F\emptyset$.

We use the notation τ, φ, r_n etc. for F as in Section 3, and the corresponding notation τ', φ', r'_n etc. for H_Σ . Recall $\hat{\varepsilon}_\omega: \nu H_\Sigma \rightarrow F^\omega 1$ from Lemma 14.

► **Remark 32.**

- (1) The homomorphism $\hat{k}: \nu H_\Sigma \rightarrow \nu F$ of Remark 27 is clearly monotone and preserves the least elements. Indeed, if $p' \in \Sigma_0$ is the chosen element, then the least element of νH_Σ is the singleton tree labelled by p' . And the least element of νF is $[p'] = \hat{k}(p')$.
- (2) Since $\hat{\varepsilon}_\omega$ is a domain-codomain restriction of \hat{k} , see Lemma 19, it also is monotone and preserves the least element.

► **Proposition 33.** The morphisms $r_n: F^\omega 1 \rightarrow F^\omega 1$ and $r'_n: H_\Sigma^\omega 1 \rightarrow H_\Sigma^\omega 1$ are related by $r_n \cdot \hat{\varepsilon}_\omega = \hat{\varepsilon}_\omega \cdot r'_n$ ($n \in \mathbb{N}$).

Proof.

- (1) We prove $F^n u \cdot \tilde{\varepsilon}_n = \hat{\varepsilon}_n \cdot H_\Sigma^n u$ by induction on $n \in \mathbb{N}$. The first step is trivial. The induction step is computed as follows:

$$\begin{aligned}
F^{n+1} u \cdot \tilde{\varepsilon}_{n+1} &= F(F^n u \cdot \tilde{\varepsilon}_n) \cdot \varepsilon_{H_\Sigma^n 0} && (\text{definition of } \tilde{\varepsilon}_n) \\
&= F(\hat{\varepsilon}_n \cdot H_\Sigma^n u) \cdot \varepsilon_{H_\Sigma^n 0} && (\text{induction hypothesis}) \\
&= F\hat{\varepsilon}_n \cdot \varepsilon_{H_\Sigma^n 1} \cdot H_\Sigma^{n+1} u && (\varepsilon \text{ natural}) \\
&= \hat{\varepsilon}_{n+1} \cdot H_\Sigma^{n+1} u && (\text{definition of } \hat{\varepsilon}_n).
\end{aligned}$$

- (2) We next verify $\bar{u} \cdot \tilde{\varepsilon}_\omega = \hat{\varepsilon}_\omega \cdot \bar{u}'$. For that it is sufficient to prove, for all $n \in \mathbb{N}$, that $\bar{u} \cdot \tilde{\varepsilon}_\omega \cdot i'_n = \hat{\varepsilon}_\omega \cdot \bar{u}' \cdot i'_n$. Indeed, (i'_n) is a collectively epic cocone. Thus, we only need to verify, by induction on $k \in \mathbb{N}$, that t_{n+k} merges the two sides of that equation: $t_{n+k} \cdot (\bar{u} \cdot \tilde{\varepsilon}_\omega \cdot i'_n) = t_{n+k} \cdot (\hat{\varepsilon}_\omega \cdot \bar{u}' \cdot i'_n)$. (Here we use the fact that $(t_{n+k})_{k \in \mathbb{N}}$ is a collectively monic cone for every n .)

This follows for $k = 0$ from the following computation:

$$\begin{aligned}
t_n \cdot \bar{u} \cdot \tilde{\varepsilon}_\omega \cdot i'_n &= t_n \cdot \bar{u} \cdot i_n \cdot \tilde{\varepsilon}_n && \text{see Remark 15} \\
&= F^n u \cdot \tilde{\varepsilon}_n && \text{(definition of } \bar{u} \text{)} \\
&= \hat{\varepsilon}_n \cdot H_\Sigma^n u && \text{see (1)} \\
&= \hat{\varepsilon}_n \cdot t'_n \cdot \bar{u}' \cdot i'_n && \text{(definition of } \bar{u}' \text{)} \\
&= t_n \cdot \hat{\varepsilon}_\omega \cdot \bar{u}' \cdot i'_n && \text{see Remark 15.}
\end{aligned}$$

And if the above equation holds for k , then we can write $t_{n+(k+1)}$ as $t_{(n+1)+k}$ and apply the above equation to k and $n + 1$. From that we obtain the induction step:

$$\begin{aligned}
t_{n+(k+1)} \cdot \bar{u} \cdot \tilde{\varepsilon}_\omega \cdot i'_n &= t_{(n+1)+k} \cdot \bar{u} \cdot \tilde{\varepsilon}_\omega \cdot i'_{n+1} \cdot H_\Sigma^n i && (i'_n \text{ compatible}) \\
&= t_{(n+1)+k} \cdot \hat{\varepsilon}_\omega \cdot \bar{u}' \cdot i'_{n+1} \cdot H_\Sigma^n i && \text{(induction hypothesis)} \\
&= t_{n+(k+1)} \cdot \hat{\varepsilon}_\omega \cdot \bar{u}' \cdot i'_n && (i'_n \text{ compatible}).
\end{aligned}$$

- (3) Now we prove for the given point $p = \varepsilon_\emptyset \cdot p' : 1 \rightarrow F0$ that $F^n p \cdot \hat{\varepsilon}_n = \tilde{\varepsilon}_{n+1} \cdot H_\Sigma^n p'$. This is trivial for $n = 0$, and the induction step is as follows:

$$\begin{aligned}
F^{n+1} p \cdot \hat{\varepsilon}_{n+1} &= F^{n+1} p \cdot F \hat{\varepsilon}_n \cdot \varepsilon_{H^{n+1}} && \text{(definition of } \hat{\varepsilon}_n \text{)} \\
&= F(\tilde{\varepsilon}_{n+1} \cdot H_\Sigma^n p') \cdot \varepsilon_{H^{n+1}} && \text{(induction hypothesis)} \\
&= F \tilde{\varepsilon}_{n+1} \cdot \varepsilon_{H_\Sigma^{n+1}} \cdot H_\Sigma^{n+1} p' && (\varepsilon \text{ natural)} \\
&= \tilde{\varepsilon}_{n+2} \cdot H_\Sigma^{n+1} p' && \text{(definition of } \tilde{\varepsilon}_n \text{)}.
\end{aligned}$$

- (4) The proof of our proposition follows. Recall that r_n is defined by

$$r_n = e_n \cdot t_n = \bar{u} \cdot i_{n+1} \cdot F^n p \cdot t_n$$

and analogously r'_n . Thus

$$\begin{aligned}
r_n \cdot \hat{\varepsilon}_\omega &= \bar{u} \cdot i_{n+1} \cdot F^n p \cdot t_n \cdot \hat{\varepsilon}_\omega \\
&= \bar{u} \cdot i_{n+1} \cdot F^n p \cdot \hat{\varepsilon}_n \cdot t'_n && \text{see Remark 15} \\
&= \bar{u} \cdot i_{n+1} \cdot \tilde{\varepsilon}_{n+1} \cdot H_\Sigma^n p' \cdot t'_n && \text{see (3)} \\
&= \bar{u} \cdot \tilde{\varepsilon}_\omega \cdot i'_{n+1} \cdot H_\Sigma^n p' \cdot t'_n && \text{see Remark 15} \\
&= \hat{\varepsilon}_\omega \cdot \bar{u}' \cdot i'_{n+1} \cdot H_\Sigma^n p' \cdot t'_n && \text{see (2)} \\
&= \hat{\varepsilon}_\omega \cdot r'_n. && \blacktriangleleft
\end{aligned}$$

In the following theorem μF is considered as a subset of νF via the monomorphism m , see Remark 18. Thus $(\mu F)^A$, ordered component-wise, is a subset of $(\nu F)^A$. Moreover, $F(\mu F)$ is considered as a poset via the bijection φ , and analogously for $F(\nu F)$.

► **Theorem 34.** *Let F be a finitary set functor with $F\emptyset \neq \emptyset$. The order of νF by cutting coincides with that of Theorem 10. And the poset νF has the same conservative completion as its subset μF . The coalgebra structure τ is the unique continuous extension of φ^{-1} .*

7:12 On Free Completely Iterative Algebras

Proof.

(1) Recall that $t_n = t_{\omega,n}$ and $Ft_n = t_{\omega+1,n+1}$, thus

$$t_{n+1} \cdot t_{\omega+\omega,\omega} = t_{\omega+\omega,n+1} = Ft_n \cdot t_{\omega+\omega,\omega+1}.$$

Moreover, observe that since $\tau^{-1} = t_{\omega+\omega+1,\omega+\omega}$, we have $t_{\omega+\omega,\omega+1} \cdot \tau^{-1} = Ft_{\omega+\omega,\omega}$.

(2) We prove that the homomorphism $\hat{k}: \nu H_\Sigma \rightarrow \nu F$ of Remark 27 fulfils

$$\hat{\varepsilon}_\omega = t_{\omega+\omega,\omega} \cdot \hat{k}: \nu H_\Sigma \rightarrow F^\omega 1.$$

Following Remark 15 we need to prove the following equalities

$$t_n \cdot (t_{\omega+\omega,\omega} \cdot \hat{k}) = \hat{\varepsilon}_n \cdot t'_n \quad (n \in \mathbb{N}).$$

The case $n = 0$ is trivial. The induction step is as follows:

$$\begin{aligned} \hat{\varepsilon}_{n+1} \cdot t'_{n+1} &= F\hat{\varepsilon}_n \cdot \varepsilon_{H_\Sigma^1} \cdot t'_{n+1} && \text{(definition of } \hat{\varepsilon}_n) \\ &= F\hat{\varepsilon}_n \cdot \varepsilon_{H_\Sigma^1} \cdot H_\Sigma t'_n \cdot \tau' && (\tau' = (t'_{\omega+1,\omega})^{-1} \\ &&& \text{and } t'_n = t'_{\omega,n}) \\ &= F(\hat{\varepsilon}_n \cdot t'_n) \cdot \varepsilon_{\nu H_\Sigma} \cdot \tau' && (\varepsilon \text{ natural}) \\ &= Ft_n \cdot Ft_{\omega+\omega,\omega} \cdot F\hat{k} \cdot \varepsilon_{\nu H_\Sigma} \cdot \tau' && \text{(induction hypothesis)} \\ &= Ft_n \cdot t_{\omega+\omega,\omega+1} \cdot \tau^{-1} \cdot F\hat{k} \cdot \varepsilon_{\nu H_\Sigma} \cdot \tau' && \text{by (1)} \\ &= Ft_n \cdot t_{\omega+\omega,\omega+1} \cdot \hat{k} && (\hat{k} \text{ a homomorphism)} \\ &= t_{n+1} \cdot t_{\omega+\omega,\omega} \cdot \hat{k} && \text{by (1)}. \end{aligned}$$

(3) The congruence \sim^* is the kernel equivalence of \hat{k} , see Remark 27. Since $t_{\omega+\omega,\omega}$ is monic, it follows from (1) that this is also the kernel equivalence of $\hat{\varepsilon}_\omega$.

(4) The ordering of $F^\omega 1$ defined in Theorem 10 coincides, when restricted to νF (via the embedding $t_{\omega+\omega,\omega}$), with the ordering by cutting. To prove this, we verify that, given elements $x = [t]$ and $y = [s]$ of νF , the following equivalence holds for every $n \in \mathbb{N}$:

$$t \sim^* \partial_n s \quad \text{iff} \quad t_{\omega+\omega,\omega}(x) = r_n \cdot t_{\omega+\omega,\omega}(y).$$

That is, we are to prove for all $n \in \mathbb{N}$ that

$$\hat{\varepsilon}_\omega(t) = \hat{\varepsilon}_\omega \cdot r'_n(s) \quad \text{iff} \quad t_{\omega+\omega,\omega}(x) = r_n \cdot t_{\omega+\omega,\omega}(y).$$

Due to (2), this translates to the following equivalence

$$\hat{\varepsilon}_\omega(t) = \hat{\varepsilon}_\omega \cdot r'_n(s) \quad \text{iff} \quad \hat{\varepsilon}_\omega(t) = r_n \cdot \hat{\varepsilon}_\omega(s),$$

which follows from Proposition 33.

(5) For the morphism $\bar{u}: \mu F \rightarrow F^\omega 1$ of 3.1(4) we prove that

$$\bar{u} = t_{\omega+1,\omega} \cdot F\bar{u} \cdot \varphi^{-1}.$$

It is sufficient to prove that the equality holds when precomposed by $i_{n+1}: F^{n+1}0 \rightarrow \mu F$ for every $n \in \mathbb{N}$. Since $i_{n+1} = i_{n+1,\omega}$ and $\varphi^{-1} = i_{\omega,\omega+1}$, we have $\varphi^{-1} \cdot i_n = i_{n+1,\omega+1} = Fi_n$. Thus we want to verify

$$\bar{u} \cdot i_{n+1} = t_{\omega+1,\omega} \cdot F(\bar{u} \cdot i_n).$$

For that, we postcompose by t_{n+1+k} for all $k \in \mathbb{N}$ (and use that given any n this cone is collectively monic):

$$t_{n+1+k} \cdot \bar{u} \cdot i_{n+1} = t_{\omega+1, n+1+k} \cdot F(\bar{u} \cdot i_n).$$

This equation holds for $k = 0$ since the left-hand side is $F^{n+1}u$, see 3.1(4), and the right-hand one is

$$t_{\omega+1, n+1} \cdot F(\bar{u} \cdot i_n) = Ft_n \cdot F\bar{u} \cdot Fi_n = F(F^n u).$$

The induction step from k to $k + 1$ (for n arbitrary) is easy: just re-write $n + 1 + k + 1$ as $n + 2 + k$ and use the induction hypothesis on $n + 1$ in place of n .

- (6) For every element $x \in F^\omega 1$, all elements $r_n(x)$ are compact. That is, given a directed set $D \subseteq F^\omega 1$, then $r_n(x) \sqsubseteq \bigsqcup D$ implies $r_n(x) \sqsubseteq y$ for some $y \in D$. This clearly holds for $F = H_\Sigma$. Due to Proposition 33 and (4) above, it also follows for F .

- (7) $F^\omega 1$ is the conservative completion of μF . More precisely, we prove that the embedding $\bar{u}: \mu F \rightarrow F^\omega 1$ has the universal property w.r.t. to continuous maps from μF to cpo's. (Observe that μF is trivially closed under existing directed joins due to Theorem 10.) First, observe that the image of each r_n is a subset of the image of \bar{u} , see Remark 12. Every element $x \in F^\omega 1$ yields a sequence $r_n(x)$ in μF , and for the order of Theorem 10 we clearly have $x = \bigsqcup_{n \in \mathbb{N}} r_n(x)$. Given a monotone function $f: \mu F \rightarrow B$ where B is a cpo, we define $\bar{f}: F^\omega 1 \rightarrow B$ by $\bar{f}(x) = \bigsqcup_{n \in \mathbb{N}} f(r_n(x))$. This is a continuous function. Indeed, given a directed set $D \subseteq F^\omega 1$ we know from Theorem 10 that $x = \bigsqcup D$ exists. Then D is mutually cofinal with $\{r_n(x); n \in \mathbb{N}\}$. This is clear if $x \in D$. Otherwise, (6) implies that each $r_n(x)$ is, due to $r_n(x) \sqsubseteq x$, under some element of D . And for each $y \in D$ the fact that $y \sqsubseteq x$ implies that we have n with $y = r_n(x)$. Consequently, $f[D]$ is mutually cofinal with $\{f(r_n(x))\}$ in B , thus, $f(\bigsqcup D) = f(x) = \bigsqcup f[D]$.

- (8) We prove that \bar{u} factorizes through $t_{\omega+\omega, \omega} = \bigcap_{n \in \mathbb{N}} t_{\omega+n, \omega}$, see Remark 16. We verify by induction a factorization through $t_{\omega+n, \omega}$. For $n = 1$, see (5). For $n = 2$ we apply (5) twice: since $Ft_{\omega+1, \omega} = t_{\omega+2, \omega+1}$, we get

$$\begin{aligned} \bar{u} &= t_{\omega+1, \omega} \cdot F(t_{\omega+1, \omega} \cdot F\bar{u} \cdot \varphi^{-1}) \\ &= t_{\omega+2, \omega} \cdot F(F\bar{u} \cdot \varphi^{-1}). \end{aligned}$$

Analogously for $n = 3, 4, \dots$

- (9) The proof of the theorem follows. First, $F^\omega 1$ is the conservative completion of νF , the argument is as in (7). It follows that $\varphi^{-1}: \mu F \rightarrow F(\mu F)$, which is a poset isomorphism (by our definition of the order of $F(\mu F)$) has at most one continuous extension to νF . And τ is continuous (indeed, a poset isomorphism, too). Thus, we just need proving that τ extends φ^{-1} . In other words, the inclusion map m of Remark 18 fulfils $\tau \cdot m = Fm \cdot \varphi^{-1}$, and Fm is also the inclusion map.

The latter is clear in case F preserves inclusion maps. Next let F be arbitrary. By Theorem III.4.5 in [6] there exists a set functor \bar{F} preserving inclusion which agrees with F on all nonempty sets and functions and fulfils $\bar{F}\emptyset \neq \emptyset$ (since $F\emptyset \neq \emptyset$). Then the categories of algebras for F and \bar{F} also coincide, thus $\mu F = \mu \bar{F}$. And the categories of nonempty coalgebras for F and \bar{F} also coincide, hence, $\nu F = \nu \bar{F}$. Since the theorem holds for \bar{F} , it also holds for F . ◀

7:14 On Free Completely Iterative Algebras

► **Corollary 35.** *A bicontinuous set functor with $F\emptyset \neq \emptyset$ has a terminal coalgebra which is the conservative completion of its initial algebra. Its coalgebra structure is the unique continuous extension of the inverted algebra structure of μF .*

This follows from the above proof: we have seen that the conservative completion of μF is $F^\omega 1$ which, for F bicontinuous, is νF .

► **Remark 36.** In the proof of the above theorem we have seen that \hat{k} is a domain restriction of $\hat{\varepsilon}_\omega$: we have $\hat{\varepsilon}_\omega = t_{\omega+\omega, \omega} \cdot \hat{k}$. And the homomorphism $\tilde{\varepsilon}_\omega$ is a domain-codomain restriction of \hat{k} , see Lemma 19. Consequently, Proposition 33 yields

$$\partial_n \cdot \hat{k} = \tilde{\varepsilon}_\omega \cdot \partial'_n : \nu H_\Sigma \rightarrow \mu F \quad (n \in \mathbb{N}).$$

Here ∂'_n is the domain-restriction of r'_n and ∂_n that of r_n , see Remark 12.

5 Free Iterative Algebras

► **Assumption 37.** Throughout this section F is a finitary set functor with a given presentation $\varepsilon : H_\Sigma \twoheadrightarrow F$, see Definition 20.

► **Remark 38.** Let X be a nonempty set.

- (1) The initial algebra of $F(-) + X$ is precisely the free algebra for F on X : notation $\Phi X = \mu F(-) + X$. Indeed, the components of the algebra structure $\varphi : F(\Phi X) + X \rightarrow \Phi X$ yield an algebra ΦX for F and a morphism $\eta : X \rightarrow \Phi X$, respectively. That F -algebra clearly has the universal property w.r.t. η .
- (2) Let us choose an element $p' \in \Sigma_0 + X$. The finitary functor $F(-) + X$ has the following presentation: the signature is Σ_X of Remark 1. And the natural transformation can, since $H_{\Sigma_X} = H_\Sigma(-) + X$, be chosen to be

$$\varepsilon + \text{id}_X : H_{\Sigma_X} \twoheadrightarrow F(-) + X.$$

This yields an element $p \in F\emptyset + X$ which is $\varepsilon_\emptyset(p')$ in case $p' \in \Sigma_0$, else $p' = p$.

► **Notation 39.** ΦX denotes the poset forming the free algebra on X for F ordered by cutting w.r.t $\varepsilon + \text{id}_X$. And \sim is the congruence on $\Phi_\Sigma X$ (the algebra of finite Σ -trees on X) of applying ε -equations, see Corollary 23.

► **Remark 40.** We do not speak about $(\varepsilon + \text{id}_X)$ -equations, since we do not have to: the function

$$\varepsilon_Z + \text{id}_X : H_\Sigma Z + X \rightarrow FZ + X$$

does not merge flat terms with variables from X , hence, every $(\varepsilon + \text{id}_X)$ -equation is simply an ε -equation.

► **Corollary 41.** *Free algebras for F are free Σ -algebras modulo ε -equations: $\Phi X = \Phi_\Sigma X / \sim$.*

► **Examples 42.**

- (1) For $F = \text{Id}$ we choose $p \in X$ and obtain $\Phi X = \mathbb{N} \times X$ ordered as follows:

$$(n, x) < (m, y) \quad \text{iff} \quad n < m \quad \text{and} \quad x = p.$$

(2) The functor $F X = A \times X$ yields

$$\Phi X = A^* \times X$$

ordered by

$$(u, x) < (v, y) \quad \text{iff} \quad u \text{ is a prefix of } v \text{ and } x = y.$$

(3) The functor $F X = X^I \times \{0, 1\}$ (corresponding to deterministic automata with a finite input set I) is naturally equivalent to H_Σ , where Σ consists of two operations a, b of arity $n = \text{card } I$. Thus ΦX is the algebra of all finite n -ary trees with inner nodes labelled by $\{a, b\}$ and leaves labelled in X . The order is by tree cutting.

(4) Let \mathcal{P}_k denote the subfunctor of the power-set functor given by all subsets of at most k elements. We can describe ΦX as the algebra of all non-ordered, finite extensional k -branching trees (i.e. every node has at most k children) with leaves labelled in $X + \{p\}$. Here we use a signature Σ having, for every $n \leq k$, precisely one n -ary operation; the nullary one is called p . Then $\Phi_\Sigma X$ is the algebra of all k -branching finite trees with leaves labelled in $X + \{p\}$. It is ordered by tree cutting. And given k -branching trees s and s' we have $s \sim s'$ iff they have the same extensional quotient, see 11(2). This yields the above description of ΦX .

To describe the order of ΦX , let us call the extensional quotient of $\partial_n s$ (cutting s at height n) the n -th *extensional cutting*. Then for distinct s, s' in ΦX we have $s < s'$ iff s is an extensional cutting of s' .

► Remark 43. Whereas the initial algebra for $F(-) + X$ is the free algebra for F , the terminal coalgebra

$$\Psi X = \nu F(-) + X$$

is the free completely iterative algebra for F , as we recall below. The concept of a *recursive equation* in an algebra $\alpha: F A \rightarrow A$ is given by a set X of recursive variables and a morphism $e: X \rightarrow F X + A$.

► Definition 44. A solution of recursive equation $e: X \rightarrow F X + A$ in an algebra (A, α) is a morphism $e^\dagger: X \rightarrow A$ making the following square

$$\begin{array}{ccc} X & \xrightarrow{e^\dagger} & A \\ e \downarrow & & \uparrow [\alpha, \text{id}] \\ F X + A & \xrightarrow{F e^\dagger + \text{id}} & F A + A \end{array}$$

commutative. The algebra (A, α) is called *completely iterative* if every recursive equation has a unique solution.

► Example 45. If $F = H_\Sigma$, we can think of e as a system of recursive equations of the form

$$x = \sigma(x_1, \dots, x_n) \quad \text{or} \quad x = a \quad (a \in A),$$

one for every variable $x \in X$ (depending on $e(x)$ lying in the left-hand or right-hand summand of $H_\Sigma X + A$). And then the solution e^\dagger makes an assignment of elements of A to variables from X satisfying those recursive equations: from $x = \sigma(x_1, \dots, x_n)$ we get $e^\dagger(x) = \sigma_A(e^\dagger(x_1), \dots, e^\dagger(x_n))$, and from $x = a$ we get $e^\dagger(x) = a$.

7:16 On Free Completely Iterative Algebras

The algebra νH_Σ of Σ -trees (with the algebra structure τ^{-1} of tree-tupling) is completely iterative. For every recursive equation $e: X \rightarrow H_\Sigma X + \nu H_\Sigma$ the solution $e^\dagger: X \rightarrow \nu H_\Sigma$ can be defined as follows: given $n \in \mathbb{N}$ we describe the cut trees $\partial'_n e^\dagger(x)$ for all variables $x \in X$ simultaneously by induction on $n \in \mathbb{N}$:

- (1) $\partial'_0 e^\dagger(x)$ is the singleton tree labelled by p .
- (2) Given $\partial'_n e^\dagger(x)$ for all $x \in X$, then for every $x \in X$ with $e(x) = \sigma(x_1, \dots, x_n)$ in the left-hand summand $H_\Sigma X$ we define $\partial'_{n+1} e^\dagger(x)$ to be the tree with root labelled by σ and with n subtrees $\partial'_n e^\dagger(x_i)$, $i = 1, \dots, n$. Whereas if $e(x) = s \in \nu H_\Sigma$, then $\partial'_{n+1} e^\dagger(x) = \partial'_{n+1} s$.

► **Theorem 46** (See [10]). *Let $\tau_X: \Psi X \rightarrow F(\Psi X) + X$ be the terminal coalgebra for $F(-) + X$. The components of τ_X^{-1} make ΨX an F -algebra with a morphism $\eta: X \rightarrow \Psi X$. This is the free completely iterative algebra for F w.r.t. the universal morphism η .*

In particular, $(\nu F, \tau^{-1})$ is the initial completely iterative algebra.

► **Notation 47.** ΨX denotes the poset forming the free completely iterative algebra on X for F ordered by cutting w.r.t. $\varepsilon + \text{id}_X$. And \sim^* is the congruence on $\Psi_\Sigma X$ (the algebra of Σ trees over X) of a possibly infinite application of ε -equations, see Remark 25.

► **Corollary 48.** *Let F be a bicontinuous set functor. The free completely iterative algebra ΨX on a set $X \neq \emptyset$ is a cpo which is the conservative completion of the free algebra ΦX .*

The algebra structure of ΨX is the unique continuous extension of the algebra structure of ΦX .

This is an application of Corollary 35 to $F(-) + X$.

► **Example 49.**

- (1) For $F = \text{Id}$ the conservative completion of $\Phi X = \mathbb{N} \times X$ adds just one maximum element as $\bigsqcup_{n \in \mathbb{N}} (n, p)$. Thus $\Psi X = \mathbb{N} \times X + 1$.
- (2) For $F X = A \times X$ the conservative completion of $A^* \times X$ adds joins to all sequences $(u_0, p) < (u_1, p) < (u_2, p) < \dots$ where each u_n is a prefix of u_{n+1} ($n \in \mathbb{N}$). That join is expressed by the infinite word in A^ω whose prefixes are all u_n . We thus get

$$\Psi X = A^* \times X + A^\omega.$$

- (3) For the bicontinuous functor $F = \mathcal{P}_k$ we can describe ΨX as the algebra of all extensional k -branching trees with leaves labelled in $X + \{p\}$.

Indeed, this algebra with the order by extensional cutting (see Example 42), is the completion of its subalgebra ΦX of finite trees. To see this, observe that every strictly increasing sequence $s_0 < s_1 < s_2 < \dots$ in ΨX has a unique upper bound: the tree s defined level by level so that, given n , its extensional cutting at n is the same as that of s_k for all but finitely many $k \in \mathbb{N}$. Therefore, given a continuous function $f: \Phi X \rightarrow B$ where B is a cpo, the unique continuous extension $\bar{f}: \Psi X \rightarrow B$ is given by $\bar{f}(s) = \bigsqcup_{n \in \mathbb{N}} f(s_n)$

where s_n is the extensional cutting of s at level n .

- (4) For the functor $F X = X^I \times \{0, 1\}$ we have $\Psi X = n$ -ary trees with leaves labelled in X and inner nodes labelled in $\{a, b\}$. The order is by cutting.
- (5) Aczel and Mendler introduced in [1] the functor $(-)_2^3$ defined by

$$X_2^3 = \{(x_1, x_2, x_3) \in X^3; x_i = x_j \text{ for some } i \neq j\}.$$

This is a bicontinuous functor with a presentation using $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$, all operations binary, and the following ε -equations

$$\sigma_1(x, x) = \sigma_2(x, x) = \sigma_3(x, x).$$

Here $\varepsilon: H_\Sigma \rightarrow (-)_2^3$ is given by $\sigma_1(x, y) \mapsto (x, x, y)$, $\sigma_2(x, y) \mapsto (x, y, y)$ and $\sigma_3(x, y) \mapsto (x, y, x)$.

The free algebra $\Phi X = \Phi_\Sigma X / \sim$ is described as follows: $\Phi_\Sigma X$ consists of finite binary trees with leaves labelled in X and inner nodes labelled in Σ . And $s \sim s'$ means that we can obtain s from s' by relabelling arbitrarily inner nodes whose left and right child yield the same tree. The order is by cutting.

The free completely iterative algebra is $\Psi X = \Psi_\Sigma X / \sim^*$, where $\Psi_\Sigma X$ are binary trees with leaves labelled in X and inner nodes labelled in Σ . And \sim^* allows infinite relabelling of the type above. ΨX is a cpo which is the conservative completion of ΦX .

► **Corollary 50.** *For every finitary set functor the free algebra on a set $X \neq \emptyset$ has the same conservative completion as the iterative algebra on X . The algebra structure of ΨX is, again, the unique continuous extension of the algebra structure of ΦX .*

This is an application of Theorem 34 to $F(-) + X$.

► **Example 51.** For the finite power-set functor \mathcal{P}_f the algebra ΨX can be described as the quotient $\Psi_\Sigma X / \sim^*$, where $\Psi_\Sigma X$ are the finitely branching trees with leaves labelled in $X + \{p\}$. And $s \sim^* s'$ means that the extensional cuttings of s and s' are the same for every level n .

A better description: ΨX is the set of all finitely branching strongly extensional trees (see Example 11(2)), with leaves labelled in $X + \{p\}$. The proof is completely analogous to that for $\nu\mathcal{P}_f$ in Worrell's paper [11].

ΨX is ordered by extensional cutting. This is not a cpo. To see this, consider an arbitrary strongly extensional tree s which is not finitely branching. Thus, $s \notin \Psi X$. Each extensional cutting is finite (since for every n we only have a finite number of extensional trees of height n) and this yields an increasing ω -sequence in ΦX that has no join in ΨX .

The common conservative completion of ΦX and ΨX is the algebra of all compactly branching strongly extensional trees with leaves labelled in $X + \{p\}$. The proof is, again, analogous to that for $\nu\mathcal{P}_f$ in [11].

6 Approximate Solutions

In this section we prove that solutions of iterative equations in free iterative algebras are obtainable as joins of ω -chains of approximate solutions. This is true for every finitary set functor F and every nonempty set of recursion variables. We first prove the corresponding result for the terminal coalgebra considered as an algebra $\tau^{-1}: F(\nu F) \rightarrow \nu F$.

Throughout this section a presentation $\varepsilon: H_\Sigma \rightarrow F$ is assumed and a choice of an element $p \in F\emptyset + X$ where X is a fixed set of “recursion” variables. In particular at the beginning we set $X = \emptyset$ and choose $p \in F\emptyset$, i.e., we work with a finitary functor with $F\emptyset \neq \emptyset$.

We continue to use $\tau, \varphi, \partial_n, \dots$ for F and $\tau', \varphi', \partial'_n, \dots$ for H_Σ (as in Section 4). We know that νF is the initial completely iterative algebra. We are going to describe solutions $e^\dagger: X \rightarrow \nu F$ of recursive equations $e: X \rightarrow FX + \nu F$ as joins of ω -chains

$$e_0^\dagger \sqsubseteq e_1^\dagger \sqsubseteq e_2^\dagger \dots : X \rightarrow \mu F$$

7:18 On Free Completely Iterative Algebras

of approximate solutions in the initial algebra. Here we work with the poset $(\nu F)^X$ ordered pointwise and its subposet $(\mu F)^X$.

Recall ∂_n from Remark 12.

► **Definition 52.** *The k -th approximate solution $e_k^\dagger: X \rightarrow \nu F$ of a recursive equation $e: X \rightarrow FX + \nu F$ is defined by induction on $k \in \mathbb{N}$ as follows:*

$e_0^\dagger: X \rightarrow \mu F$ is the least element of the poset $(\mu F)^X$,

and given e_k^\dagger , then the following square defines e_{k+1}^\dagger :

$$\begin{array}{ccc}
 X & \xrightarrow{e_{k+1}^\dagger} & \mu F \\
 \downarrow e & & \uparrow [\varphi, \text{id}] \\
 FX + \nu F & & F(\mu F) + \mu F \\
 \downarrow \text{id} + \partial_k & & \uparrow Fe_k^\dagger + \text{id} \\
 FX + \mu F & \xrightarrow{Fe_k^\dagger + \text{id}} & F(\mu F) + \mu F
 \end{array}$$

We are going to prove that the unique solution e^\dagger of e in νF is the join of the ω -chain e_k^\dagger considered in $(\nu F)^X$. Or, more precisely, for the inclusion $m: \mu F \rightarrow \nu F$ of Remark 18 we have

$$e^\dagger = \bigsqcup_{k \in \mathbb{N}} m \cdot e_k^\dagger.$$

► **Example 53.** If $F = H_\Sigma$ then e_n^\dagger is precisely the cutting $\partial'_n e^\dagger$ of Example 45. This is obvious for $n = 0$, and the induction step is easy.

► **Theorem 54.** *Let F be a finitary set functor with $F\emptyset \neq \emptyset$. For every recursive equation $e: X \rightarrow FX + \nu F$ the unique solution in νF is the join of the ω -chain of approximate solutions e_n^\dagger ($n \in \mathbb{N}$) in the poset $(\nu F)^X$.*

Proof. We know from Example 45 that the theorem holds for H_Σ . We apply this to the following recursive equation w.r.t. H_Σ :

$$e' \equiv X \xrightarrow{e} FX + \nu F \xrightarrow{b+k^*} H_\Sigma X + \nu H_\Sigma$$

where b is a splitting of ε_X and k^* splits \hat{k} , see Lemma 28. Thus, $e = (\varepsilon_X + \hat{k}) \cdot e'$. We know that $(e')^\dagger$ is the join $(e')^\dagger = \bigsqcup_{n \in \mathbb{N}} m' \cdot (e')_n^\dagger$ for the inclusion $m': \mu H_\Sigma \rightarrow \nu H_\Sigma$. From that we

derive $e^\dagger = \bigsqcup_{n \in \mathbb{N}} m \cdot e_n^\dagger$ by proving that (1) $e^\dagger = \hat{k} \cdot (e')^\dagger$ and (2) $e_n^\dagger = \tilde{\varepsilon}_\omega \cdot (e')_n^\dagger$ for $n \in \mathbb{N}$ (see

Remark 15). Indeed, we then have

$$e^\dagger = \hat{k} \cdot \bigsqcup_{n \in \mathbb{N}} m' \cdot (e')_n^\dagger = \bigsqcup_{n \in \mathbb{N}} \hat{k} \cdot m' \cdot (e')_n^\dagger$$

since post-composition with \hat{k} preserves the order and all joins that exist in $(\mu H_\Sigma)^X$: recall from Remark 32 that $\hat{k}: \nu H_\Sigma \rightarrow (\nu H_\Sigma)/\sim^*$ is the quotient map inducing the order by cutting on νF . We get from Lemma 19

$$e^\dagger = \bigsqcup_{n \in \mathbb{N}} m \cdot \tilde{\varepsilon}_\omega \cdot (e')_n^\dagger = \bigsqcup_{n \in \mathbb{N}} m \cdot e_n^\dagger$$

as required.

- (1) Proof of $e^\dagger = \hat{k} \cdot (e')^\dagger$. It is sufficient to prove that $\hat{k} \cdot (e')^\dagger$ solves e in the algebra $(\nu F, \tau^{-1})$, i.e., it is equal to $[\tau^{-1}, \text{id}] \cdot (F[\hat{k} \cdot (e')^\dagger] + \text{id}) \cdot e$. This follows from the commutative diagram below, since $e = (\varepsilon_X + \hat{k}) \cdot e'$:

$$\begin{array}{ccccc}
 X & \xrightarrow{(e')^\dagger} & \nu H_\Sigma & \xrightarrow{\hat{k}} & \nu F \\
 \downarrow e' & & \uparrow [(\tau')^{-1}, \text{id}] & & \uparrow [\tau^{-1}, \text{id}] \\
 H_\Sigma X + \nu H_\Sigma & \xrightarrow{H_\Sigma(e')^\dagger + \text{id}} & H_\Sigma(\nu H_\Sigma) + \nu H_\Sigma & \xrightarrow{\varepsilon_{\nu H_\Sigma} + \text{id}} & F(\nu H_\Sigma) + \nu H_\Sigma \\
 \downarrow \varepsilon_X + \hat{k} & \searrow H_\Sigma[\hat{k} \cdot (e')^\dagger] + \text{id} & \downarrow H_\Sigma \hat{k} + \text{id} & \searrow (N) & \downarrow F\hat{k} + \hat{k} \\
 & & H_\Sigma(\nu F) + \nu H_\Sigma & \xrightarrow{\varepsilon_{\nu F} + \hat{k}} & F(\nu F) + \nu F \\
 & & (N) & & \\
 FX + \nu F & \xrightarrow{F[\hat{k} \cdot (e')^\dagger] + \text{id}} & & & F(\nu F) + \nu F
 \end{array}$$

The upper left-hand part expresses that $(e')^\dagger$ solves e' . For all the other inner parts consider the components of the corresponding coproducts separately. The right-hand components commute in each case trivially. The left-hand components of the parts denoted by (N) commute since ε is natural. For the upper right-hand part recall that \hat{k} is a homomorphism, i.e., $\tau \cdot \hat{k} = F\hat{k} \cdot \varepsilon_{\nu H_\Sigma}$.

- (2) The proof of $e_n^\dagger = \tilde{k} \cdot (e')_n^\dagger$ is performed by induction on $n \in N$. The case $n = 0$ is trivial since $\tilde{\varepsilon}_\omega$ preserves the least element (see Remark 32) and $(e')_0^\dagger$ is the constant map of that value. The induction step follows from the commutative diagram below:

$$\begin{array}{ccccc}
 & & X & \xrightarrow{(e')_{n+1}^\dagger} & \mu H_\Sigma & \xrightarrow{\tilde{\varepsilon}_\omega} & \mu F \\
 & \swarrow e & \downarrow e' & & \uparrow [\varphi', \text{id}] & & \uparrow [\varphi, \text{id}] \\
 FX + \nu F & \xleftarrow{\varepsilon_X + \hat{k}} & H_\Sigma X + \nu H_\Sigma & & H_\Sigma(\mu H_\Sigma) + \mu H_\Sigma & & F(\mu H_\Sigma) + \mu F \\
 & & \downarrow \text{id} + \partial'_n & & \downarrow \varepsilon_{\mu H_\Sigma} + \tilde{\varepsilon}_\omega & & \\
 & & H_\Sigma X + \mu H_\Sigma & \xrightarrow{H_\Sigma(e')_n^\dagger + \text{id}} & & & \\
 \downarrow \text{id} + \partial_n & \swarrow \varepsilon_X + \tilde{\varepsilon}_\omega & & & \downarrow F\tilde{\varepsilon}_\omega + \text{id} & & \\
 FX + \mu F & \xrightarrow{F(e')_n^\dagger + \text{id}} & & & F(\mu F) + \mu F & & \\
 & & & & \downarrow F\tilde{\varepsilon}_\omega + \text{id} & & \\
 & & & & & & F(\mu F) + \mu F
 \end{array}$$

The lower triangle commutes since $e_n^\dagger = \tilde{\varepsilon}_\omega \cdot (e')_n^\dagger$ by induction hypothesis. The upper square is the definition of $(e')_{n+1}^\dagger$ and the part right of it commutes due to $\tilde{\varepsilon}_\omega$ being a homomorphism, see Lemma 19. The middle part commutes by naturality of ε . For the lower left-hand part see Remark 36.

- (3) It remains $s \sqcup (e_n^\dagger)$ to verify that $(e_n)_{n \in \mathbb{N}}$ is an ω -chain in $(\nu F)^X$. For $(e')_n^\dagger$ this follows from $(e')_n^\dagger = \partial_n \cdot (e')^\dagger$, see Example 45. Thus, we only need to observe that $(e')_n^\dagger \leq (e')_{n+1}^\dagger$ implies $\tilde{\varepsilon}_\omega \cdot (e')_n^\dagger \leq \tilde{\varepsilon}_\omega \cdot (e')_{n+1}^\dagger$. Indeed, see Remark 32. \blacktriangleleft

► **Definition 55.** For every coalgebra $\alpha: X \rightarrow FX$ we define approximate homomorphisms $h_n: X \rightarrow \mu F$ by induction on $n \in \mathbb{N}$ as follows: h_0 is the least element of $(\mu F)^X$, and given h_n we put

$$h_{n+1} \equiv X \xrightarrow{\alpha} FX \xrightarrow{Fh_n} F(\mu F) \xrightarrow{\varphi} \mu F.$$

► **Corollary 56.** For every coalgebra (X, α) the unique homomorphism to νF is the join of the ω -chain of approximate homomorphisms in $(\nu F)^X$.

Proof. Let $h: (X, \alpha) \rightarrow (\nu F, \tau)$ be the unique homomorphism. Form the recursive equation

$$e \equiv X \xrightarrow{\alpha} FX \xrightarrow{\text{inl}} FX + \nu F.$$

Then $e_n^\dagger = h_n$ for every $n \in \mathbb{N}$. This is clear for $n = 0$. The induction step follows from the square in Definition 52: observe that $(\text{id} + \partial_n) \cdot e = (\text{id} + \partial_n) \cdot \text{inl} \cdot \alpha = \text{inl} \cdot \alpha$.

Moreover, h is a solution of e : from $\tau \cdot h = Fh \cdot \alpha$ we get $h = \tau^{-1} \cdot Fh \cdot \alpha = \tau^{-1} \cdot (Fh + \text{id}) \cdot \text{inl} \cdot \alpha$, as required. Thus, our corollary follows from the preceding theorem. ◀

► **Corollary 57.** Let F be a finitary set functor. For every nonempty set Y the solutions of recursive equations in the free iterative algebra ΨY are obtained as joins of ω -chains of the approximate solutions in the free algebra ΦY .

This is just an application of the above theorem to the functor $F(-) + Y$ and a choice $p \in Y$ making ΨY a poset by cutting.

7 Conclusions and Open problems

Terminal coalgebras of finitary set functors F carry a canonical partial order which is a cpo whenever F is bicontinuous. This was observed by the author a long time ago. The present paper describes this order in a completely new manner, using the cutting of Σ -trees for a signature Σ presenting F . In the bicontinuous case the terminal coalgebra is the conservative completion of the initial algebra of F . Moreover the algebra structure of μF determines the coalgebra structure of νF as the unique continuous extension of the inverted map.

The above results are applied to free completely iterative algebras ΨX for F on all nonempty sets X . In the bicontinuous case ΨX is the conservative completion of the free algebra ΦX on X , and the algebra structure of ΨX is the unique continuous extension of that of ΦX . For finitary set functors in general, ΦX and ΨX have the same conservative completions. We have demonstrated this on several examples of “everyday” finitary functors. Our main result is that solutions of recursive equations in ΨX can be obtained as joins of ω -chains of (canonically defined) approximate solutions in ΦX .

It is an open problem whether an analogous result can be proved for accessible set functors in general. Another important question is whether there is a reasonable class of locally finitely presentable categories such that a similar order of free iterative algebras can be presented for every finitary endofunctor.

References

- 1 P. Aczel and N. Mendler. A final coalgebra theorem. *CSLI Lecture Notes*, 14, 1988. Center for the Study of Languages and Information, Stanford.
- 2 J. Adámek. On terminal coalgebras determined by initial algebras. ArXive 2736791.
- 3 J. Adámek. Free algebras and automata realizations in the language of categories. *Comment. Math. Univ. Carolinae*, 15:589–602, 1974.

- 4 J. Adámek. Final colagebras are ideal completions if initial algebras. *J. Logic Comput.*, 2:217–242, 2002.
- 5 J. Adámek and S. Milius. Terminal coalgebras and free iterative theories. *Information and Computation*, 204:1139–1172, 2006.
- 6 J. Adámek and V. Trnková. *Automata and algebras in a category*. Academic Publishers, 1990.
- 7 B. Banaschewski and E. Nelson. Completions of partially ordered sets. *SIAM J. Comput.*, 11:521–528, 1982.
- 8 M. Barr. Terminal coalgebras in well-founded set theory. *Theoret. Comput. Sci.*, 124:182–192, 1994.
- 9 C.C. Elgot. Monadic computation and iterative algebraic theories. In *Logic Colloquium '73*. North Holland Publ. Amsterdam, 1975.
- 10 S. Milius. Completely iterative algebras and completely iterative monads. *Information and Computation*, 196:1–41, 2005.
- 11 J. Worrell. On the final sequence of a finitary set functor. *Theoret. Comput. Sci.*, 338:184–199, 2005.


Strongly Unambiguous Büchi Automata Are Polynomially Predictable With Membership Queries

Dana Angluin 

Yale University, New Haven, CT, USA
dana.angluin@yale.edu

Timos Antonopoulos

Yale University, New Haven, CT, USA
timos.antonopoulos@yale.edu

Dana Fisman 

Ben-Gurion University, Beer-Sheva, Israel
<http://www.cs.bgu.ac.il/~dana>
dana@cs.bgu.ac.il

Abstract

A Büchi automaton is *strongly unambiguous* if every word $w \in \Sigma^\omega$ has at most one final path. Many properties of strongly unambiguous Büchi automata (SUBAs) are known. They are fully expressive: every regular ω -language can be represented by a SUBA. Equivalence and containment of SUBAs can be decided in polynomial time. SUBAs may be exponentially smaller than deterministic Muller automata and may be exponentially bigger than deterministic Büchi automata. In this work we show that SUBAs can be learned in polynomial time using membership and certain non-proper equivalence queries, which implies that they are polynomially predictable with membership queries. In contrast, under plausible cryptographic assumptions, non-deterministic Büchi automata are not polynomially predictable with membership queries.

2012 ACM Subject Classification Theory of computation \rightarrow Automata over infinite objects; Theory of computation

Keywords and phrases Polynomially predictable languages, Automata learning, Strongly unambiguous Büchi automata, Automata succinctness, Regular ω -languages, Grammatical inference

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.8

Funding This research was supported by grant 2016239 from the United States - Israel Binational Science Foundation (BSF), Jerusalem, Israel and by the Office of Naval Research (ONR) award #N00014-17-1-2787.

1 Introduction

Reactive systems – systems which interact with their environment via inputs and outputs in an ongoing manner, are ubiquitous in today’s life: operating systems, hardware systems, protocols and networking systems are just a few of the classical examples; self-driving cars and autonomous robots are today’s leading examples. A computation of a reactive system can be abstracted by an infinite word over an alphabet consisting of inputs and outputs. The behavior of a reactive system can thus be seen as a language of infinite words (ω -words). Thus, under the assumption that the systems are finite-state, automata that process infinite words (ω -automata), and the class of languages they recognize – the *regular ω -languages* – are a useful model for reactive systems. Indeed, this is the main model used in the design, verification and synthesis of reactive systems.

In various applications, the need to infer the language of a system at hand (or in mind) has arisen. In many settings the inference problem can assume the existence of an entity answering *membership queries* (is a certain word in the language?) and *equivalence queries* (is the current hypothesis of the identity of the language correct?). This setting, enables the use



© Dana Angluin, Timos Antonopoulos, and Dana Fisman;
licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 8; pp. 8:1–8:17

Leibniz International Proceedings in Informatics



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

of the L^* algorithm [2], an algorithm for inferring a regular language using membership queries and equivalence queries. Indeed, L^* or its improved descendants (see in [22]) have been used for tasks including black-box checking [29], assume-guarantee reasoning [18], specification mining [1], error localization [14], learning interfaces [28], regular model checking [24], finding security bugs [13], code refactoring [27, 31], learning verification fixed-points [33], as well as analyzing botnet protocols [15] and smart card readers [13].

A disadvantage of using L^* in applications that model behavior using ω -words is that it limits the learned languages to the class of *safety languages*, a strict subset of the regular ω -languages, for which the complement can be described by a language of finite words. However, many interesting properties of reactive systems, in particular, *liveness* and *fairness*, require richer classes of regular ω -languages. For this reason it is desirable to obtain a learning algorithm for the full class of regular ω -languages.

Learnability results regarding the class of regular ω -languages can be summarized shortly as follows. The full class of regular ω -languages can be learned either using a non-polynomial reduction to finite words, termed $(L)_\S$ [21], or using a representation by families of DFAs (F DFA), which may be exponentially more succinct than $(L)_\S$, although the running time of the algorithm may be polynomial in $(L)_\S$ in the worst case [5]. The maximal sub-class of the regular ω -languages which is known to be polynomially learnable is the set of languages accepted by deterministic weak parity automata (DwPA) [25].

In this work we show that while under plausible cryptographic assumptions, the class of ω -regular languages is not polynomially predictable with membership queries when the target language is represented using a *non-deterministic Büchi automaton* (Theorem 1), it is polynomially predictable with membership queries when the target language is represented using a *strongly unambiguous Büchi automaton* (Corollary 15).

The result on polynomial predictability with membership queries of *strongly unambiguous Büchi automata* (SUBA) is a corollary of a result (Theorem 12) on learning SUBAs in polynomial time using membership and non-proper equivalence queries (where hypotheses are represented using mod-2-MAs for a related language of finite words).¹ This contrast in learnability results for the class of regular ω -languages arises because the running time of the learning algorithm is bounded as a function of the size of the *representation* of the target language, and NBAs (non-deterministic Büchi automata) may be exponentially more succinct than SUBAs. Thus we also focus on *succinctness* comparisons between alternative representations.

In §2, we provide the preliminaries regarding Büchi automata and strongly unambiguous Büchi automata. In §3, we discuss the framework of learning with membership queries (MQs) and equivalence queries (EQs), and discuss related learnability results for regular ω -languages. In §4, we discuss the framework of polynomial predictability with MQs; relate it to the framework of learning with MQs and EQs; and provide the negative result regarding learnability using NBAs. The positive result about learnability using SUBAs is proved in §7, after some more necessary definitions are provided.

Complexity of learning algorithms is measured with respect to the size of the representation of the unknown target language. We thus provide, in §5, size comparison results between SUBAs and other models of regular ω -languages for which learning algorithms have been obtained. We show that SUBAs may be exponentially more succinct than FDFAs and DwPAs, but the other direction is also true: FDFAs and DwPAs can be exponentially more succinct than SUBAs. We further show that SUBAs can be exponentially more succinct than the DFA representation for $(L)_\S$ or its reverse.

¹ Mod-2-MAs are explained in the body of the paper, in §6.

The learning algorithm for SUBAs uses a reduction to unambiguous finite automata on finite words (UFAs) and uses the model of *mod-2-MA*; a special type of *multiplicity automata*. In §6 we explain multiplicity automata and mod-2-MAs and study size comparisons relating mod-2-MAs, DFAs and NFAs. A UFA of size n can be represented by a mod-2-MA of size at most n , and UFAs, NFAs and mod-2-MAs may be exponentially more succinct than DFAs. We show that NFAs may be exponentially more succinct than mod-2-MAs, and that if there exist infinitely many Mersenne primes, then mod-2-MAs are exponentially more succinct than NFAs.

Finally, as mentioned, we provide the positive result for learning SUBAs in §7 and conclude with a short discussion in §8.

2 Strongly Unambiguous Büchi Automata

Carton and Michel [12] introduced the definition of a complete strongly unambiguous Büchi automaton (CUBA) and proved that every regular ω -language can be accepted by a CUBA. Bousquet and Löding [9] introduced the relaxed definition of a strongly unambiguous Büchi automaton (SUBA) and proved that the containment and equivalence problems for SUBAs are solvable in polynomial time. Their proof reduces these problems for SUBAs to the containment and equivalence problems for unambiguous finite automata, which were shown to be solvable in polynomial time by Stearns and Hunt [20]. The usual translation of a linear temporal logic (LTL) formula into a Büchi automaton produces a SUBA; see the papers of Wilke [34, 35] for more on the relationship between LTL and CUBAs.

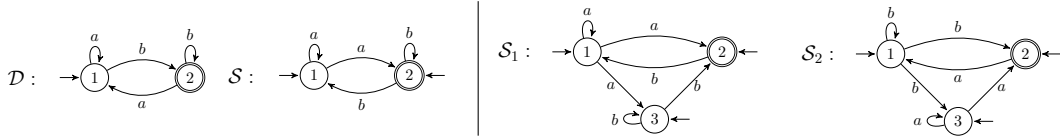
Given a finite alphabet Σ of symbols, we consider both the set of finite words Σ^* and the set of infinite or ω -words Σ^ω , which are maps from the positive integers to Σ . The term *word* refers to a finite word or an ω -word. A subset of Σ^* is a *language* and a subset of Σ^ω is an ω -*language*. For a word w , the notation $w[j]$ refers to the symbol of w indexed by j , where indices start at 1. For words w and v and a finite word u , if $w = uv$, then u is a *prefix* of w and v is a *suffix* of w .

We consider two types of automata: nondeterministic finite automata (NFAs), which accept sets of finite words, and nondeterministic Büchi automata (NBAs), which accept sets of infinite words, that can each be represented as a tuple $\mathcal{A} = (\Sigma, Q, Q_0, \Delta, F)$, where Σ is the finite alphabet of input symbols, Q is the finite set of states, $Q_0 \subseteq Q$ is the set of initial states, $\Delta \subseteq Q \times \Sigma \times Q$ is the transition relation, and $F \subseteq Q$ is the set of final states.

A *path* of an NFA (resp. an NBA) on a word w is a finite (resp. infinite) sequence of states q_0, q_1, \dots such that for all j indexing symbols of w , $(q_{j-1}, w[j], q_j) \in \Delta$. A path is *initial* if $q_0 \in Q_0$. A path of an NFA is *final* if it ends with a final state. A path of an NBA is *final* if it passes infinitely often through a final state. A path is *accepting* if it is both initial and final. For an NFA or NBA, the word w is *accepted* if there exists at least one accepting path for w . We use $\llbracket \mathcal{A} \rrbracket$ for the set of words accepted by \mathcal{A} .

The transition relation Δ is *deterministic* if for any $q_1 \in Q$ and $\sigma \in \Sigma$, there is at most one $q_2 \in Q$ such that $(q_1, \sigma, q_2) \in \Delta$. If an NFA (resp. NBA) has a deterministic transition relation and at most one initial state, then it is a deterministic finite automaton (DFA) (resp. a deterministic Büchi automaton (DBA).) The transition relation Δ is *reverse deterministic* if the reverse relation $\Delta^r = \{(q_2, \sigma, q_1) \mid (q_1, \sigma, q_2) \in \Delta\}$ is deterministic. A state q of an NFA or NBA \mathcal{A} is *live* if it appears on an accepting path for some word w . The automaton is *trim* if it contains no non-live states. The non-live states may be removed from \mathcal{A} without affecting the set of words it accepts.

8:4 SUBAs Are Polynomially Predictable With MQ



■ **Figure 1** Left: A DBA \mathcal{D} for the language $(\Sigma^*b)^\omega$ that is not a SUBA, and a SUBA \mathcal{S} for the same language. Right: Two nonisomorphic minimum SUBAs \mathcal{S}_1 and \mathcal{S}_2 for $(a+b)^*(aa^*bb^*)^\omega$.

The class of languages accepted by NFAs or DFAs is the *regular languages*. The class of languages accepted by NBAs is the *regular ω -languages*. The class of languages accepted by DBAs is a proper subclass of the class of regular ω -languages.

Unambiguous Automata

- An NFA \mathcal{A} is said to be *unambiguous* (UFA) if every word $w \in \Sigma^*$ has at most one accepting path.
- An NBA \mathcal{B} is said to be *unambiguous* (UBA) if every word $w \in \Sigma^\omega$ has at most one accepting path.
- An NBA \mathcal{B} is said to be *strongly unambiguous* (SUBA) if every word $w \in \Sigma^\omega$ has at most one final path [9].
- An NBA \mathcal{B} is said to be *strongly unambiguous and complete* (CUBA) if every word $w \in \Sigma^\omega$ has exactly one final path [12].

Note that every DFA is a UFA and every UFA is an NFA, and these containments are proper. The above standard definitions for UFAs and UBAs refer to the uniqueness of accepting paths, while the definitions for SUBAs and CUBAs refer to the uniqueness of final paths, whether or not they are also initial. Every CUBA is a SUBA, and every SUBA is an UBA, and these containments are proper. Note that a DBA is not necessarily a SUBA. For instance, the DBA \mathcal{D} of Fig. 1 which recognizes all words with infinitely many b 's is not a SUBA, since b^ω is accepted from both states 1 and 2. The SUBA \mathcal{S} of Fig. 1 accepts this language. Carton and Michel [12] proved that every regular ω -language can be accepted by a CUBA (and therefore also by a SUBA).

In this paper we focus on the class of SUBAs. It is easy to see that for any $w \in \Sigma^\omega$ if there is a final path on w , it originates at a unique state, and therefore the transition function of any trim SUBA is reverse deterministic. Unlike in the case of DFAs representing regular languages of finite words, there need not be a canonical minimum SUBA for a language. In Fig. 1 we show two non-isomorphic SUBAs \mathcal{S}_1 and \mathcal{S}_2 with the minimum possible number of states for the set of ω -words over the alphabet $\{a, b\}$ that contain an infinite number of a 's and an infinite number of b 's.²

3 Learning Regular ω -Languages With Queries

The first framework of learning that we consider is that of an algorithm learning a target language L using equivalence queries (EQs) and membership queries (MQs). In a membership query, the learning algorithm specifies a word w and receives the answer 1 if $w \in L$ and

² Automata $\mathcal{A} = (\Sigma, Q, Q_0, \Delta, F)$ and $\mathcal{A}' = (\Sigma, Q', Q'_0, \Delta', F')$ are isomorphic if they are equivalent to each other under a renaming of the states. Formally, if there exists a map $h : Q \rightarrow Q'$ such that (a) $q \in Q_0$ iff $h(q) \in Q'_0$, (b) $q \in F$ iff $h(q) \in F'$ and (c) $(q_1, \sigma, q_2) \in \delta$ iff $(h(q_1), \sigma, h(q_2)) \in \delta'$.

the answer 0 if $w \notin L$, indicating whether w is a member of the target language. In an equivalence query, the learning algorithm specifies a hypothesis, that is, a set of words H , and receives either the answer “yes”, signifying that the sets H and L are equal, or the answer “no” together with an arbitrarily selected counterexample w such that w is an element of exactly one of L and H , that is, a counterexample that shows the sets H and L are not equal. If the answer to an equivalence query is “yes”, the learning algorithm has succeeded in identifying the target language L .

The above description omits a specification of how a word w is represented when it is the subject of a MQ by the learning algorithm or when it is returned as a counterexample to an EQ, as well as how a set of words H is represented by the learning algorithm in an EQ. It also omits a measure of the “size” of the target language L , which is a parameter to functions bounding the running time of the algorithm. We now address these omissions.

As is usual, a finite word w is represented by the finite sequence of symbols it contains, and its length is the length of that sequence. Arbitrary ω -words w are not represented; rather we restrict the words used in MQs and returned as counterexamples by EQs to be *ultimately periodic*, that is, words of the form $u(v)^\omega$ for finite words $u, v \in \Sigma^*$, denoting u concatenated with an infinite sequence of copies of v . This restriction is justified by the fact that two regular ω -languages that agree on the classification of all ultimately periodic words are in fact equal [10]. Given an ω -language L and a symbol $\$$ not in the alphabet of L , we define

$$(L)_\$ = \{u\$v \mid u(v)^\omega \in L\}.$$

Then two regular ω -languages L_1 and L_2 are equal if and only if $(L_1)_\$ = (L_2)_\$$. Consistent with this definition, we assume that the ultimately periodic word $u(v)^\omega$ is represented by the finite string $u\$v$. Calbrix et al. [11] introduced this definition and showed that if L is a regular ω -language, then $(L)_\$$ is a regular language.

To represent both the target language L and a hypothesis set H , the natural choice is an automaton of the type being considered (for example, a SUBA), with an appropriate size measure (for example, the number of states of the automaton.) If this is the situation, the EQs are termed *proper*. However, it is sometimes useful to allow a different type of representation of hypotheses H in the EQs, in which case the EQs are *non-proper*. This issue will be considered further in Section 7.

In defining the running time of a learning algorithm, each EQ or MQ is counted as an oracle call, that is, one step. The learning algorithm must be able to cope with arbitrary counterexamples, which may be arbitrarily long, so it is said to run in polynomial time if its running time is bounded by a polynomial in the size of the smallest automaton accepting the target language L and the length of the longest counterexample returned by EQs.

In this learning framework, Maler and Pnueli [25] gave a polynomial time learning algorithm using MQs and proper EQs for a strict subclass of languages accepted by DBAs, namely the class of ω -languages L such that both L and its complement $(\Sigma^\omega \setminus L)$ are accepted by DBAs. This class of languages is characterized by the deterministic weak parity automata (DwPA) and is a strict subclass of the class of all regular ω -languages.

Farzan et al. [21] applied the learning algorithm \mathbf{L}^* to the problem of learning a DFA accepting the regular language $(L)_\$$, providing an algorithm using MQs and proper EQs to learn an arbitrary regular ω -language represented by NBAs.³ One issue with this approach is

³ \mathbf{L}^* runs in time polynomial in n and ℓ where n is the size of the minimal DFA for the language, and ℓ the size of the largest counterexample, asks at most n equivalence queries and at most $O(\ell n^2)$ membership queries [2]. The complexity of the algorithm proposed by Farzan et al. is thus polynomial with respect to the size of a DFA for $(L)_\$$.

that the DFA for $(L)_\S$ may be quite large: for an NBA with m states, Calbrix et al. provide an upper bound of $2^m + 2^{2m^2+m}$ on the size of a DFA for $(L)_\S$. In Section 5 we show that the minimum DFA for $(L)_\S$ and its reverse, $(L)_\S^r$, may also be exponentially larger than a SUBA for L .

Angluin and Fisman [4] proposed a representation of regular ω -languages using families of DFAs, a representation that may be exponentially more concise than the minimum DFA for $(L)_\S$, and gave a learning algorithm \mathbf{L}^ω based on that representation. However, the intermediate hypotheses of the learning algorithm could be large, in the worst case as large as a minimum DFA for $(L)_\S$.

4 A Negative Result for Learning NBAs

NBAs may be exponentially more succinct than SUBAs (see Section 5). However, in this section we show that under plausible cryptographic assumptions, there is no polynomial time algorithm to learn NBAs using equivalence and membership queries. We now define a second, more relaxed, notion of learning, namely polynomial predictability with membership queries.

The learning framework of *polynomial predictability with membership queries* assumes that there is an upper bound n on the length of relevant example words and that there is an arbitrary unknown probability distribution D on these words. The learning algorithm has access to words randomly drawn according to D and labeled according to the target concept L , as well as MQs to L . The algorithm is also given an upper bound s on the target concept and an accuracy parameter $\epsilon > 0$. We refer to the operation of drawing a word according to D and getting the result of its membership in L as *drawing a classified word*. The algorithm runs for some time, drawing classified words and making MQs until it requests one test word to predict. This word is also drawn according to D and, without making any further draws of classified words or MQs, the algorithm outputs 1 or 0 as a prediction of whether the word is a member of L or not. For the algorithm to be successful, this prediction must be correct with probability at least $(1 - \epsilon)$. In the running time of the algorithm, drawing a classified word, making a MQ and requesting the test word to predict count as oracle calls, that is, each counts as one step.

A class \mathcal{C} of languages with a particular choice of representations is *polynomially predictable with membership queries* if there exists a (possibly randomized) learning algorithm A in the framework just described, that takes as input n , s and ϵ , such that for any positive integer n , any probability distribution D over words of length at most n , for any target language $L \in \mathcal{C}$, for any positive integer s that is an upper bound on the size of the smallest representation of L , and for any $\epsilon > 0$, the algorithm A with access to words drawn according to D and classified according to L , and access to MQs for L , runs for time bounded by a polynomial in n , s , and $1/\epsilon$ and with probability at least $(1 - \epsilon)$ correctly predicts the classification of the test word to predict.

Comparing this definition of learning to that in Section 3, the definition using EQs and MQs requires complete representations of the hypotheses in a specific form, while polynomial predictability with MQs only requires the ability to make predictions of the classifications of new examples, with no restriction on how the (implicit) hypothesis is represented. The definition of polynomial predictability with MQs is more relaxed than the definition of polynomial time learning with EQs and MQs in the sense that if a class \mathcal{C} can be learned in polynomial time with EQs and MQs, where the representation used in the EQs has a polynomial time membership test, then \mathcal{C} is also polynomially predictable with membership queries. Angluin [2] showed that if the hypotheses used in EQs have a

polynomial time membership test, then polynomial time learnability with EQs and MQs implies PAC-learnability with MQs; this in turn implies polynomial predictability with MQs.

We now prove the following negative result for learning NBAs.

► **Theorem 1.** *If we assume the computational intractability of any of the following three problems: (1) testing quadratic residues modulo a composite, (2) inverting RSA encryption, or (3) factoring Blum integers, then the concept class of regular ω -languages is not polynomially predictable with membership queries, when the target language is represented by an NBA.*

This follows from the analogous result for nondeterministic finite automata proved by Angluin and Kharitonov [7, Corollary 7]. The proof below is a straightforward reduction of predicting NFAs with membership queries to predicting NBAs with membership queries.

Proof. We first describe a general transformation of an NFA \mathcal{A} over the input alphabet Σ to a related NBA \mathcal{A}' over an augmented input alphabet. Choose a new alphabet symbol $a \notin \Sigma$. Given an NFA $\mathcal{A} = (\Sigma, Q, Q_0, \Delta, F)$, we construct from it the NBA $\mathcal{A}' = (\Sigma', Q', Q'_0, \Delta', F')$ as follows. Let $\Sigma' = \Sigma \cup \{a\}$. Choose a new state, say $q' \notin Q$, and let $Q' = Q \cup \{q'\}$ and $Q'_0 = Q_0$. For the transition relation, let $\Delta' = \Delta \cup \{(q, a, q') \mid q \in F\} \cup \{(q', a, q')\}$. Thus, there is a new transition on a from every final state of \mathcal{A} to the new state q' , and a transition on a from q' to itself. Let $F' = \{q'\}$, so the new state is the only final state of \mathcal{A}' . It is not difficult to see that if $L \subseteq \Sigma^*$ is the language accepted by \mathcal{A} , then the ω -language accepted by \mathcal{A}' is $L \cdot a^\omega$.

If we have a learning algorithm \mathbf{A}' that polynomially predicts NBA acceptance using membership queries, we can use it to construct an algorithm \mathbf{A} that polynomially predicts NFA acceptance using membership queries. To implement \mathbf{A} running with a target language L accepted by the NFA \mathcal{A} and the probability distribution D on Σ^* , we simulate the algorithm \mathbf{A}' , answering its queries as follows.

Suppose \mathbf{A}' makes a membership query with (u, v) representing the ultimately periodic word $u(v)^\omega$. If $u(v)^\omega = xa^\omega$ for some $x \in \Sigma^*$ then we make a MQ to L with the word x and return the resulting answer. Otherwise, the answer to the MQ is 0.

Suppose \mathbf{A}' requests a random classified word. Then we request a word from Σ^* chosen according to D and classified according to L , which returns a pair (x, y) where $x \in \Sigma^*$ and $y \in \{0, 1\}$ indicates whether or not $x \in L$. The element we supply to \mathbf{A}' is the pair $((x, a), y)$, which indicates the choice of the ω -word xa^ω and the classification y .

Finally, when \mathbf{A}' requests the test word to predict, we request the test word to predict, and receive a string $x \in \Sigma^*$, chosen according to D . The test word that we supply to \mathbf{A}' to predict is (x, a) , representing the ω -word xa^ω .

The queries of \mathbf{A}' are answered as though it is learning the NBA \mathcal{A}' derived from \mathcal{A} , with the distribution on ω -words giving probability $D(x)$ to xa^ω for $x \in \Sigma^*$, and probability zero to words not of this form. Thus, if \mathbf{A}' predicts the correct classification of (x, a) by \mathcal{A}' , then \mathbf{A} predicts the correct classification of x by \mathcal{A} . ◀

Because of the relationship between polynomial time learnability with EQs and MQs and polynomial predictability with membership queries, we have the following.

► **Corollary 2.** *If we assume the truth of any of the three cryptographic assumptions listed in Theorem 1 then there is no polynomial time algorithm to learn NBAs using EQs and MQs, such that the representation used in the EQs has a polynomial time membership test.*

5 Size Comparisons for SUBAs

Because the performance of a learning algorithm is bounded as a function of the size of the representation of the target language, we investigate size comparisons between SUBAs and other representations of regular ω -languages. For their conversion of an NBA of n states to an equivalent SUBA, Carton and Michel [12] give an upper bound of $(12n)^n$ on the number of states in the resulting CUBA. Bousquet and Löding [9] show that there is a family of languages L_n such that L_n is accepted by a DBA (which is also an NBA) of $n + 1$ states but any SUBA to accept L_n must have at least 2^{n-1} states, showing that DBAs and NBAs may be exponentially more succinct than SUBAs. Here we focus on comparisons to representations used in learning algorithms, namely deterministic weak parity automata (DwPAs), families of DFAs (FDFAs) and DFAs for $(L)_\S$.

5.1 Comparison to FDFAs and DwPAs

A family of DFAs (FDFA) $\mathcal{F} = (\mathcal{M}, \{\mathcal{A}_q\})$ over an alphabet Σ consists of a leading deterministic automaton $\mathcal{M} = (\Sigma, Q, q_0, \delta)$ and progress DFAs $\mathcal{A}_q = (\Sigma, S_q, s_{0q}, \delta_q, F_q)$ for each $q \in Q$. Let \mathcal{M} be an automaton and (u, v) a pair of finite words representing the ultimately periodic word uv^ω . The normalization of (u, v) with respect to \mathcal{M} is the pair (x, y) such that $x = uv^i$, $y = v^j$ and $0 \leq i < j$ are the smallest for which uv^i and uv^{i+j} reach the same state of \mathcal{M} . A pair of finite words (u, v) is accepted by an FDFA $\mathcal{F} = (\mathcal{M}, \{\mathcal{A}_q\})$ if y is accepted by \mathcal{A}_{q_x} where q_x is the state reached by \mathcal{M} when reading x and (x, y) is the normalization of (u, v) with respect to \mathcal{M} . For a comprehensive discussion on FDFAs see [3].

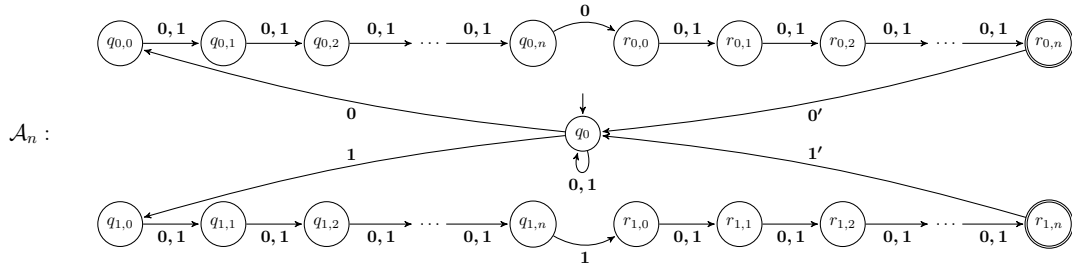
A deterministic weak parity automaton (DwPA) is a tuple $(\Sigma, Q, q_0, \delta, \kappa)$ where the first four components are the same as for DBAs, and $\kappa : Q \rightarrow \{1, \dots, k\}$ maps a state to a number from a finite set of naturals, referred to as *color*. A DwPA accepts an ω -word w if the maximal color *visited* along the run of the DwPA on w is even.

► **Theorem 3.** *There exists a family $\{L_n\}$ of ω -languages such that L_n is recognized by a SUBA of $2n + 2$ states, while any FDFA and any DwPA recognizing L_n require at least 2^n states.*

Proof. The proof uses the same family of languages used by Bousquet and Löding [9] to prove an exponential translation may be required when going from SUBAs to deterministic Muller automata. The family they proposed is given by $L_n = \Sigma^* a \Sigma^{n-1} ab^\omega$ where $\Sigma = \{a, b\}$. Fig 2. in [9] shows a SUBA for L_n with $n + 2$ states. To see that an FDFA requires at least n states, we note that the number of states in a leading automaton of an FDFA is at least the number of equivalence classes in the right congruence \sim_L where for all finite words x and y we have that $x \sim_L y$ iff $\forall w \in \Sigma^\omega. xw \in L \iff yw \in L$. The proof follows since the number of equivalence classes derived from \sim_{L_n} is at least 2^n [26]. The proof for DwPA follows from this as well since the number of equivalence classes of \sim_L is also a lower bound for the number of states of any deterministic ω -automaton for a language L [6]. ◀

► **Theorem 4.** *There exists a family $\{L_n\}$ of ω -languages such that L_n is recognized by an FDFA using $n + 3$ states and by a DwPA using $n + 2$ states, while any SUBA recognizing L_n requires at least 2^n states.*

Proof of Thm. 4 . The proof uses the same family of languages used by Bousquet and Löding [9] to prove an exponential translation may be required when going from deterministic Büchi automata to SUBAs. The family they proposed is given by $L_n = \Sigma^{n-1} a \Sigma^\omega$ where $\Sigma = \{a, b\}$. A FDFA for L_n has the same structure as a DFA for $\Sigma^{n-1} a \Sigma^*$, namely states



■ **Figure 2** A SUBA with $4n + 5$ states for L_n used in the proof of Thm. 5 for which any DFA to recognize $(L_n)_\$$ or its reverse requires at least 2^n states.

$q_0, q_1, \dots, q_n, q_{n+1}$ where q_i transits with a or b to q_{i+1} for $0 \leq i < n$, q_n transits to q_{n+1} with a and q_{n+1} transits to itself with a or b . The progress automata for all states q_i for $i \leq n$ is empty, and the progress automaton for state q_{n+1} is the one state accepting DFA. To build a DwPA for L_n we can take the same structure and the coloring $\kappa(q_{n+1}) = 2$ and $\kappa(q_i) = 1$ for any $0 \leq i \leq n$. ◀

5.2 Comparison to DFAs for $(L)_\$$ or Its Reverse

One may notice that the families of languages used in the comparisons of SUBA with DMA, DBA, F DFA and DwPA are similar to the families of languages used to show that a DFA for the reverse of a language L can be exponentially bigger or exponentially smaller than a DFA for L itself. This has to do with the fact that SUBAs are backward deterministic. We thus asked ourselves whether there exists a family of languages $\{L_n\}$ that a SUBA can characterize with polynomially in n many states, while both a DFA for $(L_n)_\$$ and a DFA for its reverse, $(L_n)_\r , would require exponentially many states. The answer is positive.

► **Theorem 5.** *There exists a family $\{L_n\}$ of ω -languages such that L_n is recognized by a SUBA of $4n + 5$ states, while any DFA to recognize $(L_n)_\$$ or its reverse requires at least 2^n states.*

Proof. We define a family of ω -languages over the alphabet $\{0, 1, 0', 1'\}$ by

$$L_n = \left((0 + 1)^* (0(0 + 1)^n 0(0 + 1)^n 0' + 1(0 + 1)^n 1(0 + 1)^n 1') \right)^\omega,$$

for all positive integers n . We define an NBA \mathcal{A}_n of $4n + 5$ states to accept L_n (depicted in Fig. 2), and show that it is a SUBA. It is not difficult to verify that \mathcal{A}_n accepts L_n . To see that \mathcal{A}_n is a SUBA, note that any final run on an ω -word w must pass infinitely often through $r_{0,n}$ or $r_{1,n}$. After the former, the only possible symbol is $0'$, and after the latter, the only possible symbol is $1'$, and these symbols can occur nowhere else. Because the transition function of \mathcal{A}_n is reverse deterministic, this completely determines the sequence of states traversed in a final run on w .

Next we show that a DFA to accept $(L_n)_\$$ must have at least 2^n states. Assume to the contrary that there is a DFA \mathcal{M} with fewer than 2^n states that accepts this language. Then there exist two different strings $u, v \in \{0, 1\}^n$ that reach the same state q of \mathcal{M} from the initial state. Without loss of generality, there exist $w, u_1, v_1 \in \{0, 1\}^*$ such that $u = w0u_1$ and $v = w1v_1$. Let x be any element of $\{0, 1\}^*$ with length $n - |u_1|$. Then $ux = w0u_1x$ and $vx = w1v_1x$ reach the same state of \mathcal{M} from the initial state. However, $w0u_1x(0u_1x0'0u_1x)^\omega$ is an element of L_n while $w1v_1x(0u_1x0'0u_1x)^\omega$ is not, so $ux\$0u_1x0'0u_1x \in (L_n)_\$$, while $vx\$0u_1x0'0u_1x \notin (L_n)_\$$, a contradiction because these two strings must reach the same state of \mathcal{M} from its initial state.

8:10 SUBAs Are Polynomially Predictable With MQ

Finally we show that a DFA to accept $(L_n)_S^r$, the reverse of $(L_n)_S$ must have at least 2^n states. Assume that there is a DFA \mathcal{M}^r of fewer than 2^n states that accepts the language $(L_n)_S^r$. There exist two different strings $u, v \in \{0, 1\}^n$ that reach the same state of \mathcal{M}^r from its initial state, and strings $w, u_1, v_1 \in \{0, 1\}^*$ such that $u = w0u_1$ and $v = w1v_1$. Let $x, y \in \{0, 1\}^*$ be arbitrary strings of lengths $|x| = n - |u_1|$ and $|y| = n - |v_1|$. Consider the two strings

$$z_0 = w0u_1x00'y = ux00'y, \text{ and } z_1 = w1v_1x00'y = vx00'y,$$

which reach the same state of \mathcal{M}^r from its initial state. Considering the reverses of these two strings, $z_0^r = y^r0'0x^ru_1^r0w^r$ is a possible period of L_n , that is,

$$0x^ru_1^r0w^r(z_0^r)^\omega \in L_n, \text{ but } 0x^ru_1^r0w^r(z_1^r)^\omega \notin L_n.$$

Thus, $z_0\$w0u_1x0 \in (L_n)_S^r$ while $z_1\$w0u_1x0 \notin (L_n)_S^r$, a contradiction because z_0 and z_1 reach the same state of \mathcal{M}^r from its initial state. \blacktriangleleft

6 Multiplicity Automata

A multiplicity automaton represents a function f mapping finite strings Σ^* to elements of a field \mathcal{K} . The definitions are formulated using vectors and matrices and their products over \mathcal{K} . A *multiplicity automaton* of dimension d is specified as a tuple $\mathcal{A} = (\Sigma, v_I, \{\mu_\sigma\}_{\sigma \in \Sigma}, v_F)$, where Σ is the input alphabet, $v_I \in \mathcal{K}^d$ is the initial state, for each symbol $\sigma \in \Sigma$, $\mu_\sigma \in \mathcal{K}^{d \times d}$ is the transition map on σ , and $v_F \in \mathcal{K}^d$ is the output map. To specify the function f computed by \mathcal{A} we first define a function μ from Σ^* to $\mathcal{K}^{d \times d}$ inductively as follows. For the empty string ε , $\mu(\varepsilon)$ is the $d \times d$ identity matrix. Given $\sigma \in \Sigma$ and $w \in \Sigma^*$, $\mu(\sigma w) = \mu_\sigma \cdot \mu(w)$, where \cdot denotes matrix product. Thus, for a word $w = \sigma_1\sigma_2 \cdots \sigma_n$, $\mu(w) = \mu_{\sigma_1} \cdot \mu_{\sigma_2} \cdots \mu_{\sigma_n}$. Then the value output by \mathcal{A} on input word w is given by

$$f(w) = v_I^\top \mu(w) v_F,$$

where the vectors v_I and v_F from \mathcal{K}^d are interpreted as $d \times 1$ column vectors.

Multiplicity automata have many useful properties. Assume that the arithmetic operations in the field \mathcal{K} take one step. Then computing $f(w)$ requires computing the product of $|w|$ square matrices and two vectors of dimension d and takes time polynomial in $|w|$ and d . Thon and Jaeger [32] (who use the term *linear sequential systems* for multiplicity automata) give polynomial time algorithms to find a basis for the state space of a multiplicity automaton, to minimize a multiplicity automaton and to test two multiplicity automata for equivalence. The algorithm to find a basis for the states also yields a shortest string w (if any) that is not mapped to 0, and shows that such a string must have length less than the dimension of the automaton. This is because if the output is 0 on all the basis elements, the function is identically 0. Given multiplicity automata \mathcal{A}_1 and \mathcal{A}_2 of dimensions d_1 and d_2 computing f_1 and f_2 , there are multiplicity automata for the sum $f_1 + f_2$ (of dimension $d_1 + d_2$) and product $f_1 \cdot f_2$ (of dimension $d_1 \cdot d_2$) that may be constructed in polynomial time.

We consider the special case of multiplicity automata over the Galois Field $\mathcal{K} = \text{GF}(2)$ of the two elements $\{0, 1\}$, in which addition is defined modulo 2. These are termed *mod-2-multiplicity automata*, abbreviated as mod-2-MAs. The outputs of a mod-2-MA \mathcal{A} are either 0 or 1, so we may consider them as language acceptors by defining $\llbracket \mathcal{A} \rrbracket$ to be the set of elements of Σ^* mapped to 1 by \mathcal{A} . We next show how to convert a mod-2-MA to an equivalent DFA.

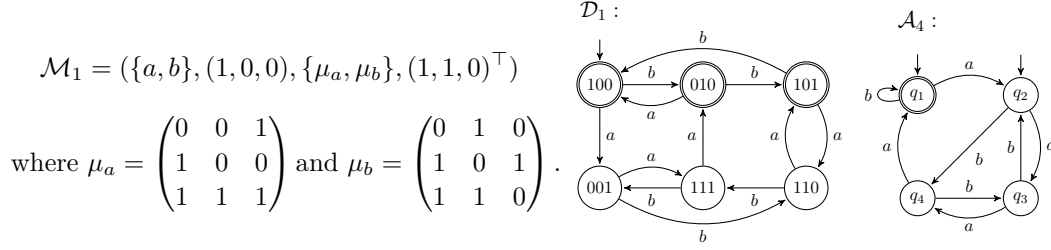


Figure 3 Left: a mod-2-MA \mathcal{M}_1 of dimension 3. Middle: the DFA \mathcal{D}_1 constructed from the mod-2-MA \mathcal{M}_1 according to Lemma 6. Right: the UFA \mathcal{A}_4 from Denis et al. [19].

Lemma 6. *Let $\mathcal{A} = (\Sigma, v_I, \{\mu_\sigma\}_{\sigma \in \Sigma}, v_F)$ be a mod-2-MA of dimension n . There exists a DFA \mathcal{A}' with at most 2^n states such that $\llbracket \mathcal{A} \rrbracket = \llbracket \mathcal{A}' \rrbracket$.*

Proof. For \mathcal{A}' the set of states is all row vectors $v \in \{0, 1\}^n$ and the initial state is v_I^\top . For states v_1 and v_2 there is a transition from v_1 to v_2 on σ if and only if $v_2 = v_1 \mu_\sigma$. The set of final states is all states v such that the inner product of v and v_F^\top is 1. Then the definition of the output of \mathcal{A} guarantees that for all $w \in \Sigma^*$, the output of \mathcal{A} on w is 1 if and only if \mathcal{A}' accepts w . ◀

As an example of this construction, consider the mod-2-MA \mathcal{M}_1 of dimension 3 given in the left of Fig. 3. The equivalent DFA \mathcal{D}_1 constructed from \mathcal{M}_1 as described in the proof of Lemma 6 is shown in the middle of Fig. 3, with unreachable states omitted. To see that the blowup in this conversion is inevitable, let's make some connection to NFAs and UFAs first.

The language accepted by \mathcal{M}_1 and \mathcal{D}_1 is the same as the language accepted by the UFA called \mathcal{A}_4 by Denis et al. [19], shown on the right in Fig. 3, which can be verified by determinizing \mathcal{A}_4 and comparing with \mathcal{D}_1 . Note that no NFA of fewer than 4 states can accept this language, because it must ensure that a^k is accepted if and only if k is 0 or 1 modulo 4. Thus, a mod-2-MA may be more concise than the smallest equivalent NFA, an issue we consider further below. For the reverse direction, we have the following, observed in Example 2.3 by Beimel et al. [8].

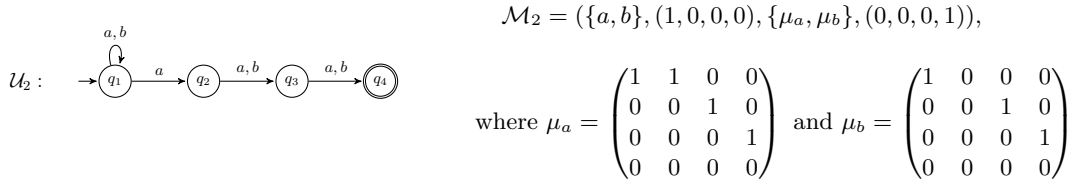
Lemma 7. *Let $\mathcal{A} = (\Sigma, Q, Q_0, \Delta, F)$ be an NFA of n states. There exists a mod-2-MA \mathcal{A}' of dimension n such that for every $w \in \Sigma^*$, the output of \mathcal{A}' on input w is 1 if and only if the number of accepting paths for w in \mathcal{A} is odd.*

Proof. Suppose the states of \mathcal{A} are $Q = \{q_1, q_2, \dots, q_n\}$. We define a mod-2-MA \mathcal{A}' of dimension n as follows. For vectors in $\{0, 1\}^n$, dimension i represents state q_i for $i = 1, 2, \dots, n$. The vector v_I is the characteristic vector of Q_0 , the vector v_F is the characteristic vector of F , and for each $\sigma \in \Sigma$, $[\mu_\sigma]_{i,j} = 1$ if and only if $(q_i, \sigma, q_j) \in \Delta$. An inductive proof shows that $[\mu(w)]_{i,j} = 1$ if and only if the number of paths on w from q_i to q_j is odd. Multiplying on the left by the transpose of v_I selects the paths starting at an initial state, and multiplying on the right by v_F adds up the results for the final states, giving 1 if and only if the total number of accepting paths on input w in \mathcal{A} is odd. ◀

In the case of a UFA, for each $w \in \Sigma^*$ the number of accepting paths is either 0 or 1, immediately implying the following fact, also observed by Beimel et al.

Corollary 8. *For any UFA \mathcal{A} of n states, there exists a mod-2-MA \mathcal{A}' of dimension n such that $\llbracket \mathcal{A} \rrbracket = \llbracket \mathcal{A}' \rrbracket$.*

8:12 SUBAs Are Polynomially Predictable With MQ



■ **Figure 4** The UFA \mathcal{U}_2 accepting $\Sigma^* a \Sigma \Sigma$ for $\Sigma = \{a, b\}$ and the mod-2-MA for the same language constructed according to Lemma 7.

As an example of this construction, consider the language $\Sigma^* a \Sigma \Sigma$, consisting of all strings over $\Sigma = \{a, b\}$ with an a as the third symbol from the end. This is accepted by the UFA \mathcal{U}_2 of 4 states, shown in Fig. 4 on the left. The mod-2-MA \mathcal{M}_2 of dimension 4 constructed from \mathcal{U}_2 according to Lemma 7 is given on the right of Fig. 4.

► **Remark 9.** In the case of mod-2-MAs, the results listed above for general multiplicity automata give polynomial time algorithms to minimize the number of dimensions, and to test equivalence, emptiness and universality, as well as polynomial time constructions for the symmetric difference (by addition), complement (by adding the constant 1), intersection (by multiplication), and union (by intersection and complement) of two given automata. If the language is nonempty, a shortest accepted word may be found in polynomial time and has length less than the number of dimensions (see [32]).

Beimel et al. [8] consider the problem of learning a multiplicity automaton computing a function f using suitably generalized equivalence and membership queries (see Section 3 for the basic definitions of learning with queries.) In particular, the answer to a membership query on word $w \in \Sigma^*$ is the value of $f(w) \in \mathcal{K}$, and the answer to an equivalence query with a multiplicity automaton \mathcal{H} is either “yes” or a counterexample, that is, a word $w \in \Sigma^*$ such that the output of \mathcal{H} on w is not equal to $f(w)$. Beimel et al. give an algorithm to learn a multiplicity automaton in time polynomial in the rank of the Hankel matrix of f (which is bounded above by the dimension of the multiplicity automaton) and the length of the longest counterexample. In contrast to \mathbf{L}^* , in which rows of the observation table that are unequal become distinct states, in the learning algorithm for multiplicity automata, a row becomes a new basis vector only if it is linearly independent of the existing basis vectors.

Given that DFAs, UFAs, NFAs and mod-2-MAs all accept the class of regular languages, an important distinction between them is succinctness, the number of states in the smallest automaton to accept a given regular language. An NFA, UFA or mod-2-MA of n states can be converted to an equivalent DFA of at most 2^n states. DFAs are UFAs, which are NFAs, so NFAs are at least as succinct as UFAs, which are at least as succinct as DFAs. By Corollary 8, each UFA can be converted to a mod-2-MA of the same size, so mod-2-MAs are at least as succinct as UFAs and DFAs. The family of languages L_n given by $(a+b)^* a (a+b)^n$ shows that NFAs, UFAs and mod-2-MAs may be exponentially more succinct than DFAs.

Schmidt [30] considers the family of complements of the languages L_n given by $\{ww \mid w \in \{0,1\}^n\}$, and shows that the complement of L_n can be accepted by an NFA of $O(n^2)$ states, but requires at least 2^n states for a UFA. His argument is that the rank of the binary matrix $F(x,y) = xy \notin L_n$ where $x,y \in \{0,1\}^n$ is at least 2^n , and therefore that a UFA must have at least 2^n states. This also shows that a mod-2-MA for the complement of L_n must have dimension at least 2^n . This leaves the question of whether mod-2-MAs may be non-polynomially more concise than UFAs or NFAs. Here we consider the comparison of the sizes of NFAs and mod-2-MAs over a unary alphabet.

► **Lemma 10.** *There is a family of unary languages $\{L_n\}$ such that the smallest NFA that accepts L_n has size $O(n^2)$ while the smallest mod-2-MA that accepts L_n has dimension at least $2^{\Theta(n/\log n)}$.*

Proof. Given n , let p_1, p_2, \dots, p_ℓ denote the primes less than or equal to n and let $P(n)$ denote their product. Define L_n to contain all words a^t such that t is not a positive integer multiple of $P(n)$. We define an NFA \mathcal{A}_n to accept L_n as follows. The states are $Q = \{q_0\} \cup \{r_{i,j} \mid 1 \leq i \leq n, 0 \leq j < p_i\}$, where q_0 is the only initial state and all states are final except for those in $\{r_{i,j} \mid 1 \leq i \leq k, j = p_i - 1\}$. There are transitions on a from q_0 to each $r_{i,0}$ and from each $r_{i,j}$ to $r_{i,j+1}$, where the addition is modulo p_i . Then ε is accepted, and a^t is rejected only if t is a positive integer multiple of each p_i , that is, a positive integer multiple of $P(n)$, so \mathcal{A}_n accepts L_n and has $O(n^2)$ states.

Let \mathcal{A}' be a mod-2-MA of dimension N accepting L_n . Then there is a mod-2-MA of dimension $N + 1$ accepting the complement of L_n (Remark 9). But the shortest word in the complement of L_n has length $P(n)$, so $P(n) < N + 1$. Because the number of primes less than or equal to n is $\Theta(n/\log n)$ and each prime is at least 2, the lower bound follows. ◀

For more information on sizes of finite automata accepting unary languages, see the paper of Chrobak [16, 17], which gives more refined bounds.

In the other direction, we prove a conditional lower bound in the following lemma. A *Mersenne prime* is a prime of the form $2^d - 1$ for some positive integer d . Unfortunately, it is unknown whether there are infinitely many Mersenne primes. For this paper, a *shift register sequence* of dimension d is an infinite periodic sequence $\{a_n\}$ of bits defined by initial conditions $a_i = b_i$ for $i = 0, 1, \dots, d - 1$ and a linear recurrence

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_d a_{n-d},$$

for all $n \geq d$, where each $c_i \in \{0, 1\}$ and the addition is modulo 2. The maximum possible minimum period of a shift register sequence is $2^d - 1$, and it is known that for every positive integer d there are shift register sequences of maximum period. These are known as *maximal length* or *pseudo noise* sequences. A maximal length sequence has 2^{d-1} ones and $2^{d-1} - 1$ zeros in the period. Golomb's book [23] is a definitive reference for shift register sequences. An example of a maximal length sequence of dimension 4 is given by $a_0 = 0, a_1 = 0, a_2 = 0, a_3 = 1$ and for all $n \geq 4, a_n = a_{n-3} + a_{n-4} \pmod{2}$. This recurrence generates a sequence with period 000100110101111.

► **Lemma 11.** *If there are infinitely many Mersenne primes then there is a family of unary languages L_d such that for infinitely many values of d , L_d is accepted by a mod-2-MA of dimension d but no NFA of fewer than $2^d - 1$ states accepts L_d .*

Proof. Suppose the unary alphabet is $\Sigma = \{\#\}$. For a language $L \subseteq \Sigma^*$ we may define the infinite sequence χ_n^L where $\chi_n^L = 1$ if $\#^n \in L$ and 0 otherwise. Given a maximal length shift register sequence $\{a_n\}$ of dimension d , there exists a mod-2-MA \mathcal{A} of dimension d such that the language L accepted by \mathcal{A} has $\chi_n^L = a_n$ for all $n \geq 0$. Such a mod-2-MA \mathcal{A} can be constructed as follows. Let $a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_d a_{n-d}$, with initial conditions $a_i = b_i$ for $0 \leq i < d$, be the linear recurrence used to define the given maximal length shift register sequence. Then let the mod-2-MA \mathcal{M} be the tuple

$$(\{\#\}, (b_0, \dots, b_{d-1}), \{\mu_\#\}, (1, 0, \dots, 0)^\top),$$

where the matrix μ_{\sharp} is as prescribed on the right.

$$\mu_{\sharp} = \begin{pmatrix} 0 & 0 & \dots & 0 & c_d \\ 1 & 0 & \dots & 0 & c_{d-1} \\ 0 & 1 & \dots & 0 & c_{d-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & c_1 \end{pmatrix}$$

It follows that $(\chi_m^L, \chi_{m+1}^L, \dots, \chi_{m+d-1}^L) \cdot \mu_{\sharp}$ is equal to $(\chi_{m+1}^L, \chi_{m+2}^L, \dots, \chi_{m+d}^L)$, for any $m \geq 0$, and therefore, for any $n \geq 0$, the value of χ_n^L is equal to $(b_0, b_1, \dots, b_{d-1}) \cdot (\mu_{\sharp})^n \cdot (1, 0, \dots, 0)^{\top}$.

Let \mathcal{A}' be an NFA accepting L corresponding to a maximal length shift register sequence of dimension d . Then \mathcal{A}' must contain at least one reachable cycle of states with at least one final state. The length of that cycle must not be 1 or relatively prime to $2^d - 1$, or else positions of the period that should be 0's will eventually be assigned 1. More specifically, since the sequence is not constant and since the period of the shift register sequence is $2^d - 1$, there exist $m_1, m_2 < 2^d$ such that for all ℓ the value of $a_{\ell \cdot (2^d - 1) + m_1}$ is 0 and the value of $a_{\ell \cdot (2^d - 1) + m_2}$ is 1. On the other hand if there is a cycle of length p in \mathcal{A}' , then for some $k \geq 0$, the value of $a_{k+j_1 \cdot p}$ is equal to $a_{k+j_2 \cdot p}$, for all $j_1, j_2 \geq 0$. It can be shown that if p is coprime to $2^d - 1$, then there exist $z_1, z_2, x_1, x_2 \geq 0$, such that $x_1 \cdot (2^d - 1) + m_1$ is equal to $k + z_1 \cdot p$ and $x_2 \cdot (2^d - 1) + m_2$ is equal to $k + z_2 \cdot p$, a contradiction. Thus p must not be coprime to $2^d - 1$. Therefore, if $2^d - 1$ is prime, that is, if $2^d - 1$ is a Mersenne prime, p must be a multiple of $2^d - 1$ and as a result \mathcal{A} must have at least $2^d - 1$ states. ◀

7 Learning SUBAs in Polynomial Time With MQs and Non-Proper EQs

In contrast to the negative result in Section 4 for learning NBAs, we show that SUBAs can be learned in polynomial time using MQs and non-proper EQs. Since these two classes of automata both accept all the regular ω -languages, a key difference is in the succinctness of the representation of a regular ω -language by an NBA versus a SUBA.

► **Theorem 12.** *There is a polynomial time algorithm to learn all regular ω -languages, when the target language is represented by a SUBA, using membership queries and non-proper equivalence queries (using mod-2-MAs to represent hypotheses).*

The existence of the learning algorithm is established by the following two results.

► **Lemma 13** (Bousquet and Löding [9]). *Let \mathcal{A} be a SUBA of n states accepting the language $L = \llbracket \mathcal{A} \rrbracket$. Then there is a UFA of $O(n^2)$ states for the language $(L)_{\S}$.*

Bousquet and Löding prove that there are polynomial time algorithms to determine containment and equivalence of two regular ω -languages represented by SUBAs. Their proof gives a construction that takes a SUBA \mathcal{A} of n states accepting a language L and produces a UFA of at most $n + 2n^2$ states that accepts $(L)_{\S}$.

► **Lemma 14** (Beimel et al. [8]). *There is a polynomial time algorithm to learn UFAs with membership and non-proper equivalence queries (using mod-2-MAs to represent hypotheses).*

Beimel et al. [8] give an algorithm to learn multiplicity automata over a field \mathcal{K} using (generalized) MQs and EQs that runs in time polynomial in the dimension of the target automaton and the length of the longest counterexample. They remark (p. 519) that their

results imply a polynomial time algorithm for learning UFAs. Specifically, combining their algorithm with the fact that UFAs can be transformed to mod-2-MAs of the same size (Corollary 8) yields the lemma.

Proof of Theorem 12. Let \mathcal{A} be a SUBA of n states with input alphabet Σ accepting the language L . We assume $\$ \notin \Sigma$, and let $\Sigma_{\$} = \Sigma \cup \{\$\}$. For a MQ to L , the input is a string $u\$v$ and the answer is 1 or 0 according to whether $u(v)^\omega \in L$, that is, whether $u\$v \in (L)_{\$}$. For a non-proper EQ to L , the input is a mod-2-MA \mathcal{H} over the alphabet $\Sigma_{\$}$ and the answer is “yes” if the language accepted by \mathcal{H} is precisely $(L)_{\$}$. Otherwise, the answer is a counterexample $w \in (\Sigma_{\$})^*$ such that \mathcal{H} accepts w and $w \notin (L)_{\$}$ or \mathcal{H} rejects w and $w \in (L)_{\$}$.

The learning algorithm of Beimel et al. may use these EQs and MQs to L to learn a mod-2-MA for the language $(L)_{\$}$. Because $(L)_{\$}$ is accepted by a UFA of $O(n^2)$ states, and therefore by a mod-2-MA of dimension $O(n^2)$ (Corollary 8), the learning algorithm runs in time polynomial in n and the length of the longest counterexample. ◀

In contrast to the negative result in Section 4 for the representation of regular ω -languages by NBAs, we have the following corollary.

▶ **Corollary 15.** *The class of ω -regular languages is polynomially predictable with membership queries when the target language is represented by a SUBA.*

Proof. By the discussion preceding Theorem 1, it is sufficient to note that there is a polynomial time algorithm that takes as inputs a mod-2-MA \mathcal{H} and a finite word w and decides whether \mathcal{H} accepts w . ◀

8 Discussion

We have shown that there is a polynomial time algorithm to learn strongly unambiguous Büchi automata (SUBAs) using membership queries and non-proper EQs, where the EQs use mod-2-MAs to represent $(L)_{\$}$. This implies that SUBAs are polynomially predictable with membership queries. By contrast, we have shown that under plausible cryptographic assumptions, general NBAs are not polynomially predictable with MQs. In applications, careful thought should be given to the choice of representations, considering both succinctness and learnability.

Given that the standard translation of a linear temporal logic (LTL) formula to an NBA yields a SUBA [34, 35], it is natural to ask what our results imply about the learnability of LTL formulas. The first step of the standard translation of an LTL formula ϕ constructs a state set consisting of all subsets of subformulas of ϕ , so the constructed SUBA may be of size exponential in the size of ϕ . Thus an algorithm that runs in time polynomial in the size of the SUBA does not avoid this exponential blow up. However, the difficulty of learning LTL formulas may be unavoidable: LTL formulas are at least as expressive and succinct as Boolean formulas, and the results of Angluin and Kharitonov [7] show that under the same cryptographic assumptions in Theorem 1, Boolean formulas are not polynomially predictable with membership queries.

Several avenues of research are suggested by our results. Can the equivalence queries used by the learning algorithm of Theorem 12 be modified to use SUBAs, or at least NBAs, as hypotheses? Farzan et al. [21] show how to convert the DFA hypotheses used by their algorithm to NBAs for equivalence queries, a method that does not extend to mod-2-MAs. The question of how different two minimal SUBAs for the same language can be appears to be open; note that the SUBAs in Fig. 1 are isomorphic if we permit a permutation of

the alphabet symbols. Given the useful properties of mod-2-MAs, perhaps the idea of using mod-2-MAs for $(L)_\S$ as a general representation of regular ω -languages L should be explored. On a minor point, is there a proof that mod-2-MAs may be exponentially more succinct than NFAs without the assumption that there are infinitely many Mersenne primes? Finally, none of our results bear on the open questions of whether deterministic Büchi automata, deterministic parity automata or deterministic Muller automata are learnable in polynomial time with equivalence and membership queries.

References

- 1 G. Ammons, R. Bodík, and J. R. Larus. Mining specifications. In *Conference Record of POPL 2002: The 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, OR, USA, January 16-18, 2002*, pages 4–16, 2002.
- 2 D. Angluin. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.
- 3 D. Angluin, U. Boker, and D. Fisman. Families of DFAs as Acceptors of omega-Regular Languages. In *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016, August 22-26, 2016 - Kraków, Poland*, pages 11:1–11:14, 2016.
- 4 D. Angluin and D. Fisman. Learning Regular Omega Languages. In *Algorithmic Learning Theory - 25th International Conference, ALT 2014, Bled, Slovenia, October 8-10, 2014. Proceedings*, pages 125–139, 2014.
- 5 D. Angluin and D. Fisman. Learning regular omega languages. *Theor. Comput. Sci.*, 650:57–72, 2016.
- 6 D. Angluin and D. Fisman. Regular omega-Languages with an Informative Right Congruence. In *GandALF*, volume 277 of *EPTCS*, pages 265–279, 2018.
- 7 D. Angluin and M. Kharitonov. When Won't Membership Queries Help? *J. Comput. Syst. Sci.*, 50(2):336–355, 1995.
- 8 A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. Learning Functions Represented As Multiplicity Automata. *J. ACM*, 47(3):506–530, May 2000.
- 9 N. Bousquet and C. Löding. Equivalence and Inclusion Problem for Strongly Unambiguous Büchi Automata. In *Language and Automata Theory and Applications, 4th International Conference, LATA 2010, Trier, Germany, May 24-28, 2010. Proceedings*, pages 118–129, 2010. doi:10.1007/978-3-642-13089-2_10.
- 10 J.R. Büchi. On a Decision Method in Restricted Second Order Arithmetic. In *International Congress on Logic, Methodology and Philosophy*, pages 1–11. Stanford Univ. Press, 1962.
- 11 Hugues C., M. Nivat, and A. Podelski. Ultimately Periodic Words of Rational w -Languages. In *Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics*, pages 554–566, London, UK, 1994. Springer-Verlag.
- 12 O. Carton and M. Michel. Unambiguous Büchi automata. *Theor. Comput. Sci.*, 297(1-3):37–81, 2003. doi:10.1016/S0304-3975(02)00618-7.
- 13 G. Chalupar, S. Peherstorfer, E. Poll, and J. de Ruitter. Automated Reverse Engineering using Lego®. In *8th USENIX Workshop on Offensive Technologies (WOOT 14)*, San Diego, CA, August 2014. USENIX Association.
- 14 M. Chapman, H. Chockler, P. Kesseli, D. Kroening, O. Strichman, and M. Tautschnig. Learning the Language of Error. In *Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings*, pages 114–130, 2015.
- 15 C. Y. Cho, D. Babic, E. C. R. Shin, and D. Song. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 426–439, 2010. doi:10.1145/1866307.1866355.

- 16 M. Chrobak. Finite Automata and Unary Languages. *Theor. Comput. Sci.*, 47(3):149–158, 1986.
- 17 M. Chrobak. Errata to: Finite Automata and Unary Languages. *Theor. Comput. Sci.*, 47(3):149–158, 2003.
- 18 J. M. Cobleigh, D. Giannakopoulou, and C. S. Păsăreanu. Learning Assumptions for Compositional Verification. In *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS '03*, pages 331–346, Berlin, Heidelberg, 2003. Springer-Verlag. URL: <http://dl.acm.org/citation.cfm?id=1765871.1765903>.
- 19 F. Denis, A. Lemay, and A. Terlutte. Residual Finite State Automata. *Fundam. Inform.*, 51:339–368, January 2002.
- 20 E. E. Stearns and B. Hunt. On the Equivalence and Containment Problems for Unambiguous Regular Expressions, Regular Grammars and Finite Automata. *SIAM J. Comput.*, 14:598–611, August 1985.
- 21 A. Farzan, Y-F. Chen, E.M. Clarke, Y-K. Tsay, and B-Y. Wang. Extending Automated Compositional Verification to the Full Class of Omega-Regular Languages. In *TACAS*, pages 2–17, 2008.
- 22 D. Fisman. Inferring regular languages and ω -languages. *J. Log. Algebr. Meth. Program.*, 98:27–49, 2018.
- 23 S. W. Golomb. *Shift Register Sequences*. Aegean Park Press, Laguna Hills, CA, USA, 1981.
- 24 P. Habermehl and T. Vojnar. Regular Model Checking Using Inference of Regular Languages. *Electr. Notes Theor. Comput. Sci.*, 138(3):21–36, 2005.
- 25 O. Maler and A. Pnueli. On the Learnability of Infinitary Regular Sets. *Inf. Comput.*, 118(2):316–326, 1995.
- 26 O. Maler and L. Staiger. On Syntactic Congruences for Omega-Languages. *Theor. Comput. Sci.*, 183(1):93–112, 1997.
- 27 T. Margaria, O. Niese, H. Raffelt, and B. Steffen. Efficient test-based model generation for legacy reactive systems. In *HLDVT*, pages 95–100. IEEE Computer Society, 2004.
- 28 W. Nam and R. Alur. Learning-Based Symbolic Assume-Guarantee Reasoning with Automatic Decomposition. In *ATVA*, volume 4218 of *Lecture Notes in Computer Science*, pages 170–185. Springer, 2006.
- 29 D. Peled, M. Y. Vardi, and M. Yannakakis. Black Box Checking. In *FORTE*, pages 225–240, 1999.
- 30 E. M. Schmidt. *Succinctness of Descriptions of Context-free, Regular and Finite Languages*. PhD thesis, Cornell University, Ithaca, NY, USA, 1978.
- 31 M. Schuts, J. Hooman, and F. W. Vaandrager. Refactoring of Legacy Software Using Model Learning and Equivalence Checking: An Industrial Experience Report. In *Integrated Formal Methods - 12th International Conference, IFM 2016, Reykjavik, Iceland, June 1-5, 2016, Proceedings*, pages 311–325, 2016.
- 32 M. R. Thon and H. Jaeger. Links between multiplicity automata, observable operator models and predictive state representations: a unified learning framework. *Journal of Machine Learning Research*, 16:103–147, 2015.
- 33 A. Vardhan, K. Sen, M. Viswanathan, and G. Agha. Using Language Inference to Verify Omega-Regular Properties. In *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, pages 45–60, 2005.
- 34 T. Wilke. Past, Present, and Infinite Future. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 95:1–95:14, 2016.
- 35 T. Wilke. Backward Deterministic Büchi Automata on Infinite Words. In *FSTTCS*, 2017.

A Robust Class of Linear Recurrence Sequences

Corentin Barloy

École Normale Supérieure de Paris, France

Nathanaël Fijalkow

CNRS, LaBRI, Bordeaux, France

The Alan Turing Institute of data science, London, United Kingdom

Nathan Lhote

University of Warsaw, Poland

Filip Mazowiecki

LaBRI, Université de Bordeaux, France

Abstract

We introduce a subclass of linear recurrence sequences which we call poly-rational sequences because they are denoted by rational expressions closed under sum and product. We show that this class is robust by giving several characterisations: polynomially ambiguous weighted automata, copyless cost-register automata, rational formal series, and linear recurrence sequences whose eigenvalues are roots of rational numbers.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases linear recurrence sequences, weighted automata, cost-register automata

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.9

Related Version The full version is available on arXiv [4] at <https://arxiv.org/abs/1908.03890>.

Funding The second author was supported by the CODYS project ANR-18-CE40-0007.

Acknowledgements We thank Théodore Lopez for reporting a maths typo in Lemma 10, S. Akshay for fruitful discussions, and the anonymous reviewers for their useful suggestions.

1 Introduction

The study of sequences of numbers originated in mathematics and has deep connections with many fields. A prominent class of sequences is that of *linear recurrence sequences*, such as the Fibonacci sequence

$$0, 1, 1, 2, 3, 5, 8, 13, \dots$$

Despite the simplicity of linear recurrence sequences many problems related to them remain open, and are the object of active research. In theoretical computer science the two main questions are:

- How to finitely represent sequences?
- How to algorithmically analyse properties of sequences?

In this paper we focus on problems related to the first question. The question of representation has led to important insights in the structure of linear recurrence sequences by giving several equivalent characterisations, some of which we briefly review here. We refer to Section 2 and the next sections for technical definitions.



© Corentin Barloy, Nathanaël Fijalkow, Nathan Lhote, and Filip Mazowiecki; licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 9; pp. 9:1–9:16

Leibniz International Proceedings in Informatics



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

9:2 A Robust Class of Linear Recurrence Sequences

Linear recurrence sequences. A sequence of rational numbers $\mathbf{u} = \langle u_n \rangle_{n \in \mathbb{N}} = \langle u_0, u_1, u_2, \dots \rangle$ is a linear recurrence system (LRS) if there exist real numbers a_1, \dots, a_k such that for all $n \geq 0$

$$u_{n+k} = a_1 u_{n+k-1} + \dots + a_k u_n. \quad (1)$$

In this paper we will consider only sequences of rational numbers, therefore, we additionally assume that a_i are rational numbers. The smallest k for which \mathbf{u} satisfies an equation of the form (1) is called the order of \mathbf{u} . The Fibonacci sequence $\langle F_n \rangle_{n \in \mathbb{N}}$ is an LRS of order 2 satisfying the recurrence $F_{n+2} = F_{n+1} + F_n$.

Rational expressions. Studying the closure properties of linear recurrence sequences yields the following result, an instance of the Kleene-Schützenberger theorem [20]: linear recurrence sequences form the smallest class of sequences containing the sequences $\langle a, 0, 0, \dots \rangle$ for a rational number a and closed under sum, Cauchy product, and Kleene star.

Weighted automata. The model of weighted automata is a well studied quantitative extension of classical automata. In general a weighted automaton recognises a function $f : \Sigma^* \rightarrow \mathbb{R}$, hence when considering a unary alphabet this becomes $f : \{a\}^* \rightarrow \mathbb{R}$, and identifying $\{a\}^*$ with \mathbb{N} we can see f as a sequence of numbers. Whenever we write about sequences recognised by models like weighted automata, we implicitly assume that these are over a unary alphabet.

Cost-register automata. Several characterisations of weighted automata have been introduced [6, 12, 3]. We will be interested in the model of *cost-register automata* (CRA). These are deterministic models with registers whose contents are blindly updated (i.e., without transitions like zero tests). It was shown that considering linear updates yields a model equivalent to weighted automata.

We summarise in one theorem the equivalences above, which is the starting point of our work. Technical definitions are given in the paper.

► **Theorem 1** (Folklore, see for instance [5, 20, 7]). *The following classes of sequences are effectively equivalent.*

- *Linear recurrence sequences,*
- *Sequences recognised by weighted automata,*
- *Sequences recognised by linear cost-register automata,*
- *Sequences denoted by rational expressions,*
- *Sequences whose formal series are rational, i.e. of the form $\frac{P}{Q}$ where P, Q are polynomials.*

Algorithmic analysis of linear recurrence sequences

The questions regarding algorithmic analysis are far from being answered. A very simple and natural problem, the Skolem problem, is still unsolved [21, 18]: given a linear recurrence sequence, does it contain a zero? Recent breakthrough results sharpened our understanding of the Skolem problem [16, 17], but one of the outcomes is that the general problem for the whole class of linear recurrence sequences is beyond our reach at the moment, since it would impact notoriously difficult problems from number theory. We refer the reader to the recent survey about what is known to be decidable for linear recurrence sequences [18].

Our contributions

Since the full class of linear recurrence sequences is too hard to be algorithmically analysed (we only mentioned the Skolem problem but many related problems are also difficult), let us revise our ambitions, go back to the drawing board, and study tractable subclasses.

In this paper we introduce *poly-rational sequences* which is a strict fragment of linear recurrence sequences. We give several equivalent characterisations of this class following the equivalence results stated in Theorem 1. Our results are summarised in the following theorem.

- **Theorem 2.** *The following classes of sequences are effectively equivalent.*
- *Sequences denoted by poly-rational expressions (Section 2),*
 - *Sequences recognised by polynomially ambiguous weighted automata (Section 3),*
 - *Sequences recognised by copyless cost-register automata (Section 4),*
 - *Sequences whose formal series are of the form $\frac{P}{Q}$ where P, Q are polynomials and the roots of Q are roots of rational numbers (Section 5),*
 - *Linear recurrence sequences whose eigenvalues are roots of rational numbers (Section 5).*

We do not discuss the efficiency of reductions proving the equivalences. Our constructions are elementary, and in most cases they yield blow ups in the size of representation.

We note that the Skolem problem and its variants are known to be decidable, and NP-hard, for the subclass of poly-rational sequences. The decidability easily follows from the fact that our class is subsumed by other classes for which such results were obtained (see e.g. [19], for the case where all eigenvalues are roots of algebraic real numbers). The Skolem problem is known to be NP-hard already for the class of LRS whose eigenvalues are roots of unity [1]. This implies that the Skolem problem for the class of poly-rational sequences is also NP-hard, which is the best known lower bound even for the full class of linear recurrence sequences.

Related works

The intractability of the Skolem problem for linear recurrence sequences also impacts the other equivalent models, leading to the study of several restrictions. A classical approach to tame weighted automata is to bound the ambiguity of weighted automata, i.e. bounding the number of accepting runs with a function depending on the length of the word. Many positive results have been obtained in the past years following this approach [11, 10, 8].

Another restriction studied in the model of cost-register automata is the *copyless* restriction: registers are not allowed to be copied more than once. It was conjectured that the copyless restriction would result in good decidability properties [3], but this has been recently falsified [2].

2 Linear recurrence sequences and rational expressions

We let $\mathbf{u} = \langle u_n \rangle_{n \in \mathbb{N}} = \langle u_0, u_1, u_2 \dots \rangle$ denote a sequence of rational numbers.

Linear recurrence sequences

We will assume that an LRS \mathbf{u} is given by the numbers a_1, \dots, a_k and the values of the first k elements: u_0, \dots, u_{k-1} . The recurrence (1) induces the sequence \mathbf{u} . We let **LRS** denote the class of LRS. Given an LRS we define its characteristic polynomial as

$$Q(x) = x^k - a_1 x^{k-1} - \dots - a_{k-1} x - a_k.$$

9:4 A Robust Class of Linear Recurrence Sequences

The roots of the characteristic polynomial are called the *eigenvalues* of the LRS.

Formal series

Formal series are a different representation for sequences. The sequence $\langle u_n \rangle_{n \in \mathbb{N}}$ induces the formal series $S(x) = \sum_{n \in \mathbb{N}} u_n x^n$, with the interpretation that the coefficient of x^n is the value of the n -th element in the sequence. Note that a polynomial represents a sequence with a finite support.

► **Example 3.** A standard example of an LRS is the Fibonacci sequence $\langle F_n \rangle_{n \in \mathbb{N}}$ defined by the recurrence $F_{n+2} = F_{n+1} + F_n$ and initial values $F_0 = 0, F_1 = 1$. Its characteristic polynomial is $p(x) = x^2 - x - 1$, whose roots are $\frac{1+\sqrt{5}}{2}$ and $\frac{1-\sqrt{5}}{2}$. The corresponding formal series is $S(x) = \sum_{n=0}^{\infty} F_n x^n$. Using the definition of F we obtain $S(x) = x + xS(x) + x^2S(x)$ and thus $S(x) = \frac{x}{1-x-x^2}$.

Rational expressions

We start by defining three classes of sequences.

- **Fin:** a sequence \mathbf{u} is in **Fin**, or equivalently \mathbf{u} has finite support, if the set $\{n \in \mathbb{N} : u_n \neq 0\}$ is finite;
- **Arith:** a sequence \mathbf{u} is in **Arith**, or equivalently \mathbf{u} is arithmetic, if $u_0 = a, u_{n+1} = u_n + b$ for some rational numbers a, b ;
- **Geo:** a sequence \mathbf{u} is in **Geo**, or equivalent \mathbf{u} is geometric, if $u_0 = a, u_{n+1} = \lambda \cdot u_n$, for some rational numbers a, λ .

We let \mathbf{Geo}_λ denote the class of geometric sequences with a fixed parameter λ .

We now define some classical operators. Here $\mathbf{u}, \mathbf{v}, \mathbf{u}^1, \dots, \mathbf{u}^k$ are sequences.

- **Sum:** $\mathbf{u} + \mathbf{v}$ is the component wise sum of sequences;
- **Cauchy product:** $\mathbf{u} \cdot \mathbf{v} = \langle \sum_{p+q=n} u_p \cdot v_q \rangle_{n \in \mathbb{N}}$; inducing $(\mathbf{u})^n$ defined by $(\mathbf{u})^0 = \langle 1, 0, 0, 0, \dots \rangle$ and $(\mathbf{u})^{n+1} = (\mathbf{u})^n \cdot \mathbf{u}$, in particular $(\mathbf{u})^1 = \mathbf{u}$;
- **Kleene star:** $(\mathbf{u})^* = \sum_{n \in \mathbb{N}} (\mathbf{u})^n$, it is only defined when $u_0 = 0$;
- **Hadamard product:** $\mathbf{u} \times \mathbf{v}$ is the component wise product of sequences;
- **Shift:** $\langle a, \mathbf{u} \rangle = \langle a, u_0, u_1, \dots \rangle$, defined for any rational number a ;
- **Shuffle:** $\text{shuffle}(\mathbf{u}^1, \mathbf{u}^2, \dots, \mathbf{u}^k) = \langle u_0^1, u_0^2, \dots, u_0^k, u_1^1, u_1^2, \dots, u_1^k, u_2^1, \dots \rangle$.

We write $\mathbf{Rat}[\mathcal{C}, \text{op}_1, \dots, \text{op}_k]$ for the smallest class of sequences containing \mathcal{C} and closed under the operators $\text{op}_1, \dots, \text{op}_k$. Rational expressions in Theorem 1 are classically defined as follows [20]:

$$\mathbf{Rat} = \mathbf{Rat}[\mathbf{Fin}, +, \cdot, *].$$

The class **Rat** contains all classes defined above, and is closed under all mentioned operators, i.e.

$$\mathbf{Rat} = \mathbf{Rat}[\mathbf{Fin} \cup \mathbf{Arith} \cup \mathbf{Geo}, +, \cdot, *, \times, \text{shift}, \text{shuffle}].$$

We now introduce a class of sequences denoted by a fragment of rational expressions, whose study is the purpose of this article. The class is called poly-rational sequences, because they are denoted by rational expressions using sum and product.

► **Definition 4** (Poly-rational sequences).

$$\mathbf{PolyRat} = \mathbf{Rat}[\mathbf{Arith} \cup \mathbf{Geo}, +, \times, \text{shift}, \text{shuffle}].$$

In other words **PolyRat** is the smallest class of sequences containing arithmetic and geometric sequences that is closed under sum, Hadamard product, shift, and shuffle. A trivial observation is that **Fin** \subseteq **PolyRat** since using shift one can generate any sequence with finite support. One could try to simplify the definition of **PolyRat** replacing **Arith** \cup **Geo** with **Fin**. Unfortunately, the operators $+$, \times , shift, shuffle are too restricted, and geometric and arithmetic sequences could not be generated. In fact, the class would collapse to **Fin**.

Since **Rat** contains **Arith** and **Geo** and is closed under Hadamard product, shift, and shuffle, we have **PolyRat** \subseteq **Rat**. We will show that the inclusion is indeed strict. As we will see in this paper, the class **PolyRat** has many equivalent and surprising characterisations.

3 Characterisation with polynomially ambiguous weighted automata

We refer to e.g. [7] for an excellent introduction to weighted automata. We consider weighted automata over the rational semiring $(\mathbb{Q}, +, \cdot)$, where $+$ and \cdot are the standard sum and product. For an alphabet Σ , weighted automata recognise functions assigning rational numbers to finite words, i.e. $f : \Sigma^* \rightarrow \mathbb{Q}$. In this paper we will consider only one-letter alphabets so the set of words is $\{a\}^* = \{\varepsilon, a, a^2, \dots\}$, which is identified with \mathbb{N} . Therefore, weighted automata recognise functions $f : \mathbb{N} \rightarrow \mathbb{Q}$, i.e. weighted automata recognise sequences of rational numbers.

Formally, a weighted automaton is a tuple $\mathcal{A} = (Q, M, I, F)$, where Q is a finite set of states, M is a $Q \times Q$ matrix over \mathbb{Q} and I, F are the initial and final vectors, respectively, of dimension Q (for convenience we label the coordinates by elements of Q). The sequence recognised by the automaton \mathcal{A} is $\llbracket \mathcal{A} \rrbracket$ defined by $\llbracket \mathcal{A} \rrbracket(n) = I^t M^n F$, where I^t is the transpose of I .

We give an equivalent definition of \mathcal{A} in terms of accepting runs. We say that a state $q \in Q$ is an initial state if $I(q) \neq 0$ and that it is a final state if $F(q) \neq 0$. If q is initial we say that its initial weight is $I(q)$, and if q is final then its final weight is $F(q)$. For two states $p, q \in Q$ we say that there is a transition from p to q if $M(p, q) \neq 0$. Such a transition is denoted $p \rightarrow q$ and its weights is $M(p, q)$. A run ρ is a sequence of consecutive transitions, and it is accepting if the first state is initial and the last state is final. The value of an accepting run $\rho = q_0 \rightarrow q_1 \rightarrow \dots \rightarrow q_n$ is

$$|\rho| = I(q_0) \cdot \left(\prod_{i=0}^{n-1} M(q_i, q_{i+1}) \right) \cdot F(q_n).$$

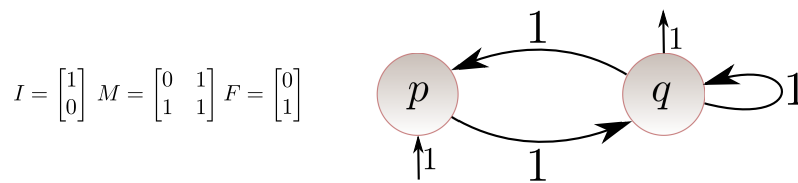
Let $Runs_{\mathcal{A}}(n)$ denote the set of all accepting runs of length n . An alternative and equivalent definition of $\llbracket \mathcal{A} \rrbracket$ is

$$\llbracket \mathcal{A} \rrbracket(n) = \sum_{\rho \in Runs_{\mathcal{A}}(n)} |\rho|.$$

► **Example 5.** Consider the automaton $\mathcal{A} = (Q, M, I, F)$ represented in Figure 1. We have $\llbracket \mathcal{A} \rrbracket(n) = F_n$, where $\langle F_n \rangle_{n \in \mathbb{N}}$ is the Fibonacci sequence from Example 3.

The ambiguity of an automaton \mathcal{A} is the function $a_{\mathcal{A}} : \mathbb{N} \rightarrow \mathbb{N}$ which associates to n the number of accepting runs $|Runs_{\mathcal{A}}(n)|$. We consider the following classes:

- **DetWA** – the class of deterministic weighted automata, i.e. such that for any p , there exists at most one q such that $M(p, q) \neq 0$;
- **kWA** for fixed $k \in \mathbb{N}$ – the class of k -ambiguous weighted automata, i.e. when $a_{\mathcal{A}}(n) \leq k$ for all n ;



■ **Figure 1** A weighted automaton recognising the Fibonacci sequence.

- **FinWA** = $\bigcup_{k \in \mathbb{N}} \mathbf{kWA}$ – the class of finitely ambiguous weighted automata, i.e. when there exists k such that $a_{\mathcal{A}}(n) \leq k$ for all n ;
- **PolyWA** – class of polynomially ambiguous automata, i.e. when there exists a polynomial $P : \mathbb{N} \rightarrow \mathbb{N}$ such that $a_{\mathcal{A}}(n) \leq P(n)$ for all n ;
- **WA** – the full class of weighted automata.

For example, the automaton in Example 5 is not polynomially ambiguous because the number of accepting runs is exponential. We will see that this is no accident by proving in Section 5 that the Fibonacci sequence is not in **PolyWA**.

We present our first characterisation of **PolyRat**.

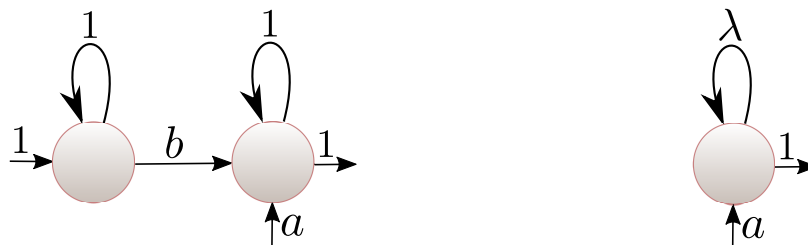
► **Theorem 6.** **PolyRat = PolyWA**

Proof of Theorem 6

This subsection is divided into two parts for both inclusions.

PolyRat \subseteq PolyWA

Figure 2 shows how to recognise the arithmetic and the geometric sequences. For each finitely



■ **Figure 2** The weighted automaton on the left recognises the arithmetic sequence with parameters (a, b) and it is linearly ambiguous. The weighted automaton on the right recognises the geometric sequence with parameters a, λ and it is deterministic.

supported sequence a simple weighted automaton can be constructed. It remains to prove that the class **PolyWA** is closed under the operators. The sum and products correspond to union and product of automata, it is readily verified that these standard constructions preserve the polynomial ambiguity. Below we deal with shift and shuffle operators.

Suppose we have a polynomially ambiguous automaton \mathcal{A} for \mathbf{u} and we want to construct a new polynomially ambiguous automaton \mathcal{A}' for $\langle a, \mathbf{u} \rangle$. We start with the case when $a = 0$. Then \mathcal{A}' has the same set of states as \mathcal{A} plus one new state q_0 , which is the only initial state in \mathcal{A}' . All transitions from \mathcal{A} are inherited. There are additionally only outgoing transitions from q_0 to all states that are initial in \mathcal{A} ; the weight of each transition is the initial weight of the corresponding state in \mathcal{A} . It is readily verified that \mathcal{A}' recognises $\langle 0, \mathbf{u} \rangle$ and that \mathcal{A}' is

polynomially ambiguous. For $a \neq 0$ it suffices to add one more state that is both initial and final with initial weight 1 and final weight a .

To deal with shuffle we start with the following preliminary construction. Fix some $k > 0$ and a polynomially ambiguous automaton \mathcal{A} recognising \mathbf{u} . We construct $\mathcal{A}[k]$ recognising $\mathbf{u}' = (\underbrace{u_0, 0, \dots, 0}_k, \underbrace{u_1, 0, \dots, 0}_k, u_2, \dots)$, i.e. elements u_i are separated by $k - 1$ elements with 0.

The idea to construct \mathcal{A}' is that the set of states have an additional component $\{0, \dots, k - 1\}$, and they behave like \mathcal{A} every k -th step; in the remaining steps they only wait. Formally, the set of states of $\mathcal{A}[k]$ is $Q \times \{0, \dots, k - 1\}$, where Q is the set of states of \mathcal{A} . The initial (final) states are $(q, 0)$ such that q is initial (final) in \mathcal{A} with the same weight. For every transition $p \rightarrow q$ in \mathcal{A} there is a transition $(p, 0) \rightarrow (q, 1)$ in $\mathcal{A}[k]$ with the same weight. The remaining transitions are $(q, i) \rightarrow (q, (i + 1) \bmod k)$ with weight 1, defined for every $i > 0$ and every $q \in Q$. It is readily verified that $\mathcal{A}[k]$ recognises \mathbf{u}' .

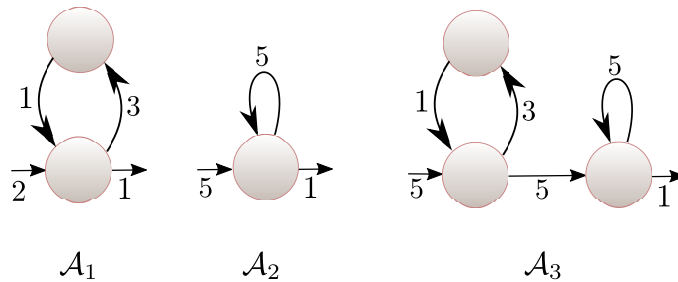
Let $\mathcal{A}_0, \dots, \mathcal{A}_{k-1}$ be polynomially ambiguous automata recognising $\mathbf{u}_0, \dots, \mathbf{u}_{k-1}$. For every \mathcal{A}_i let $\mathcal{A}_i[k]$ be an automaton as above, additionally shifted i times with 0's. Then $\text{shuffle}(\mathbf{u}_0, \dots, \mathbf{u}_{k-1})$ is recognised by the disjoint union of $\mathcal{A}_i[k]$.

PolyWA \subseteq PolyRat

The first step is to decompose polynomially ambiguous automata into a union of automata that we will call *chained loops*. We say that the states $p_0, p_1, \dots, p_{k-1} \in Q$ form a *loop* if $M(p_i, p_j) \neq 0$ is equivalent to $j = i + 1 \bmod k$ and a *path* if $M(p_i, p_j) \neq 0$ is equivalent to $j = i + 1$ (in particular p_{k-1} has no successor). A chained loop of size k is an automaton over the set states of $\{q_0, \dots, q_{k-1}\} \cup P$ such that

- q_0 is the unique initial state;
- q_0, \dots, q_{k-1} form a path;
- each q_i is contained in at most one loop (the states in P are used only as intermediate states in the loops);
- q_{k-1} is the unique final state with $F(q_{k-1}) = 1$.

We define the concatenation of two chained loops $\mathcal{A}_1, \mathcal{A}_2$: this is the chained loop obtained by constructing the union of the two automata with the initial state being the initial state of \mathcal{A}_1 , the final state being the final state of \mathcal{A}_2 , and rewiring the output of \mathcal{A}_1 to the initial state of \mathcal{A}_2 , see e.g. Figure 3.



■ **Figure 3** Three example chained loops. The initial and final weights are depicted by ingoing and outgoing edges. The chained loop \mathcal{A}_1 recognises the sequence defined by $f_1(2n) = 2 \cdot 3^n$, $f_1(2n+1) = 0$ whose power series is $\frac{2}{1-3x^2}$. The chained loop \mathcal{A}_2 recognises the sequence $f_2(n) = 5^{n+1}$ whose power series is $\frac{5}{1-5x}$. The chained loop \mathcal{A}_3 is the concatenation of \mathcal{A}_1 and \mathcal{A}_2 and it recognises the sequence $f_3(n) = \sum_{i=1}^n f_1(i-1) \cdot f_2(n-i)$ whose power series is $\frac{10x}{(1-3x^2)(1-5x)}$.

► **Lemma 7.** *Every polynomially ambiguous weighted automaton is equivalent to a union of chained loops.*

Proof. Let \mathcal{A} be a polynomially ambiguous weighted automaton. Without loss of generality \mathcal{A} is trimmed, i.e. every state occurs in at least one accepting run.

We first note that any state in \mathcal{A} is contained in at most one loop. Indeed, a state contained in two loops induces a sequence of words with exponential ambiguity. This implies that a sequence (q_0, q_1, \dots, q_k) with $q_i \neq q_j$ for $i \neq j$ induces at most one chained loop of which it is the path. There are finitely many such sequences because k is bounded by the number of states of \mathcal{A} .

We claim that \mathcal{A} is equivalent to the union of all chained loops induced by such sequences. Indeed, there is a bijection between the runs of \mathcal{A} and the runs of all the chained loops, respecting the values of runs. Consider a run ρ of \mathcal{A} , where a state q appears multiple times. Then between each occurrence of q this is the same run, because they are loops over q and there can be only one loop containing q . So $\rho = uv^k w$, where v is the (only) loop containing q . Repeating this for u and w , we obtain a unique decomposition of ρ into

$$q_0 \cdot \ell_0^{m_0} \cdot q_0 \rightarrow q_1 \cdot \ell_1^{m_1} \cdot q_1 \rightarrow \dots \rightarrow q_k \cdot \ell_k^{m_k} \cdot q_k,$$

where ℓ_i is a loop over q_i (we can have $m_i = 0$) and $q_i \neq q_j$ for $i \neq j$. ◀

Our aim is to use the decomposition result stated in Lemma 7 to prove the inclusion **PolyWA** \subseteq **PolyRat**. It will be convenient for reasoning to use formal series.

► **Lemma 8.**

- *The formal series induced by a chained loop of size 1 with a loop is of the form $\frac{\alpha}{1-\lambda x^\ell}$, where $\alpha = I(q_0)$, λ is the product of the weights in the loop and ℓ is the length of the loop. If there is no loop this reduces to α .*
- *Let S_1, S_2 be the formal series induced by the chained loops \mathcal{A}_1 and \mathcal{A}_2 , then the formal series induced by the concatenation of \mathcal{A}_1 and \mathcal{A}_2 is $x \cdot S_1 \cdot S_2$.*
- *Let S_1, S_2 be the formal series induced by two automata \mathcal{A}_1 and \mathcal{A}_2 , then the formal series induced by the union of \mathcal{A}_1 and \mathcal{A}_2 is $S_1 + S_2$.*

Proof. The first and the third item are immediate, we focus on the second. For convenience let us assume that $\mathcal{A}_1(-1) = 0$. By definition the concatenation of two chained loops recognises the sequence defined by

$$[[\mathcal{A}]](n) = \sum_{i=0}^n [[\mathcal{A}_1]](i-1) \cdot [[\mathcal{A}_2]](n-i)$$

since an accepting run in the concatenation is the concatenation of an accepting run in \mathcal{A}_1 and an accepting run in \mathcal{A}_2 . The only issue is that the output state of \mathcal{A}_1 was changed into a transition, and to include this step we write $\mathcal{A}_1(i-1)$ instead of $\mathcal{A}_1(i)$. Hence the formal series is indeed the Cauchy product of S_1 and S_2 , shifted by one. ◀

We are now half-way through the proof of the inclusion **PolyWA** \subseteq **PolyRat**: thanks to Lemma 7, we can restrict our attention to unions of chained loops, and thanks to Lemma 8, we know what are the formal series induced by the sequences computed by such automata. More specifically, they are obtained from formal series of the form $\frac{\alpha}{1-\lambda x^\ell}$ by taking sums and Cauchy products (with an additional shift).

To prove that **PolyRat** contains such sequences it is tempting to attempt showing that the sequences above are in **PolyRat** and the closure of **PolyRat** under sums and Cauchy

products. Unfortunately, the closure under Cauchy product is not clear (although it will follow from the final result that it indeed holds).

We sidestep this issue by observing that we only need to be able to do Cauchy products of formal series of a special form. Indeed, the formal series described above are of the form $\frac{P}{Q}$ where P, Q are rational polynomials and the roots of Q are roots of rational numbers: this is true of $\frac{\alpha}{1-\lambda x^\ell}$ and is clearly closed under sums and Cauchy products (with the additional shift).

Notice that every chained loop can be obtained as concatenations of chained loops of size 1. Thus Lemma 8 gives a characterisation of formal series corresponding to unions of chained loops: these are sums of products of $\frac{\alpha}{1-\lambda x^\ell}$ and polynomials. We further simplify this characterisation applying the following lemma.

► **Lemma 9.** *Consider the formal series $\frac{P}{Q}$ where P, Q are rational polynomials and the roots of Q are roots of rational numbers. Then $\frac{P}{Q}$ can be written as the sum of formal series of the form $\frac{R}{(1-\lambda x^\ell)^k}$ for rational polynomials R , rational numbers λ , and ℓ, k natural numbers.*

Proof. This is a direct consequence of the fact that $\mathbb{Q}[x]$ is a Euclidean ring. The exact statement following from this is that any product $\prod_{i=1}^n \frac{R_i}{P_i}$ where the polynomials P_i are mutually prime (meaning, for each i , the polynomials P_i and $\prod_{j \neq i} P_j$ are coprime) can be written as a sum of $\frac{Q_i}{P_i}$ for some rational polynomials Q_i .

To conclude, we observe that any polynomial whose roots are roots of rational numbers can be written as a product of mutually prime polynomials of the form $(1 - \lambda x^\ell)^k$. ◀

By Lemma 8 and Lemma 9 it follows that for every finite union of chained loops its formal series is a sum of $\frac{R}{(1-\lambda x^\ell)^k}$ for rational polynomials R , rational numbers λ , and ℓ, k natural numbers. Combining this with Lemma 7 we get that the formal series computed by **PolyWA** are of the same form. Thus we have reduced proving the inclusion **PolyWA** \subseteq **PolyRat** to proving that sequences whose formal series are sums of formal series of the form $\frac{R}{(1-\lambda x^\ell)^k}$ are in **PolyRat**.

Since **PolyRat** is closed under sum, it suffices to consider one such formal series. Moreover, due to the closure under shifts we can assume that the polynomial R is equal to 1; as stated in the lemma below.

► **Lemma 10.** *The sequence whose formal series is $\frac{1}{(1-\lambda x^\ell)^k}$ is in **PolyRat**.*

Proof. We know that

$$\frac{1}{(1-\lambda x^\ell)^k} = \sum_{n \in \mathbb{N}} \binom{n+k-1}{k-1} \lambda^n x^{\ell \cdot n}.$$

Note that $\binom{n+k-1}{k-1}$ is a polynomial in n of degree at most $k-1$, i.e. $\binom{n+k-1}{k-1} = \sum_{p=0}^{k-1} a_p n^p$. It follows that

$$\frac{1}{(1-\lambda x^\ell)^k} = \sum_{p=0}^{k-1} a_p \cdot \sum_{n \in \mathbb{N}} n^p \lambda^n x^{\ell \cdot n}$$

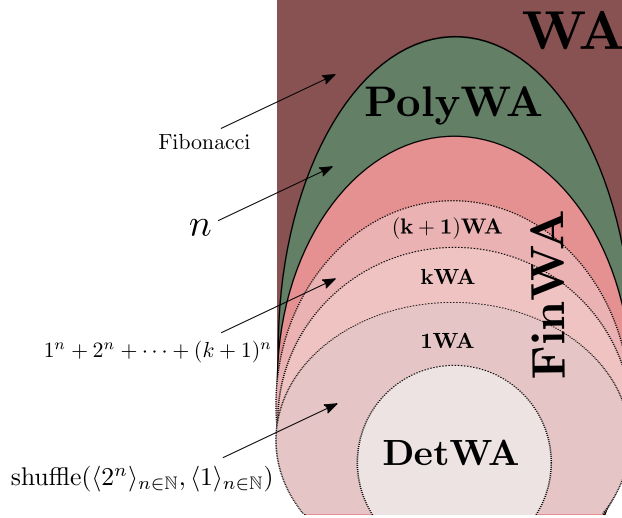
It is enough to prove that for each p the sequence whose formal series is

$$\sum_{n \in \mathbb{N}} a_p n^p \lambda^n x^{\ell \cdot n}$$

is in **PolyRat**. Using an arithmetic sequence and Hadamard products we construct $\langle a_p n^p \rangle_{n \in \mathbb{N}}$. Multiplying it using Hadamard product with the geometric sequence $\langle \lambda^n \rangle_{n \in \mathbb{N}}$ yields $\langle a_p n^p \lambda^n \rangle_{n \in \mathbb{N}}$. Shuffling the obtained sequence with $\ell-1$ null sequences yields the desired sequence. ◀

3.1 Application: the ambiguity hierarchy of weighted automata

We show that the natural classes of weighted automata defined by ambiguity can be described using subclasses of rational expressions.



■ **Figure 4** The strict ambiguous hierarchy of weighted automata.

► **Lemma 11.**

- $\mathbf{DetWA} = \bigcup_{\lambda \in \mathbb{Q}} \mathbf{Rat}[\mathbf{Geo}_\lambda, \text{shift}, \text{shuffle}]$;
- $\mathbf{FinWA} = \mathbf{Rat}[\mathbf{Geo}, +, \text{shift}, \text{shuffle}]$.

Proof. We start by proving $\mathbf{DetWA} = \bigcup_{\lambda \in \mathbb{Q}} \mathbf{Rat}[\mathbf{Geo}_\lambda, \text{shift}, \text{shuffle}]$.

(\subseteq) Since the automaton is deterministic it has a shape of a lasso, i.e. the states can be partitioned into a path such that the last state on the path is in a loop. Let λ be the value obtained by multiplying all values on the loop, let l be the length of the loop and let m be the length of the path. Then it is easy to see that the sequence is obtained by first taking a shuffle of l sequences in \mathbf{Geo}_λ and then shifting it m times.

(\supseteq) We already know that \mathbf{Geo}_λ are definable by deterministic weighted automata from Figure 2. Closure under shift follows from the construction in the proof of $\mathbf{PolyRat} \subseteq \mathbf{PolyWA}$ because it preserves the property of being deterministic. The shuffle construction preserves this property only up to a certain point. The construction of each automaton $\mathcal{A}_i[k]$ is deterministic but taking their sum does not yield explicitly a deterministic automaton. It suffices to observe that by construction $\mathcal{A}_i[k]$ are all lasso automata with loops of the same length. Moreover, every word is accepted by at most one $\mathcal{A}_i[k]$. To define the final automaton consider $\mathcal{A}_i[k]$ with the longest path. The final automaton will be $\mathcal{A}_i[k]$ with modified transitions and final outputs. Indeed we add the automata one by one, and for every accepting state we readjust the ingoing and outgoing transitions to give the correct value.

Proof of $\mathbf{FinWA} = \mathbf{Rat}[\mathbf{Geo}, +, \text{shift}, \text{shuffle}]$.

(\subseteq) By Lemma 7 we know that each automaton in \mathbf{FinWA} is a union of chained loops. It is easy to see that every such chained loop has to be a lasso otherwise it will contradict the assumption that the automaton is finitely ambiguous. Then the construction follows by doing the construction for every lasso as in the proof of $\mathbf{DetWA} = \bigcup_{\lambda \in \mathbb{Q}} \mathbf{Rat}[\mathbf{Geo}_\lambda, \text{shift}, \text{shuffle}]$ and using $+$ to deal with the union.

(\supseteq) This follows the same steps as the proof of $\mathbf{DetWA} = \bigcup_{\lambda \in \mathbb{Q}} \mathbf{Rat}[\mathbf{Geo}_\lambda, \text{shift}, \text{shuffle}]$. It is even simpler because we can take a union of two automata and remain in the class of \mathbf{FinWA} . \blacktriangleleft

We give examples witnessing the strict inclusions $\mathbf{DetWA} \subsetneq \mathbf{FinWA} \subsetneq \mathbf{PolyWA} \subsetneq \mathbf{WA}$ and $\mathbf{kWA} \subsetneq (\mathbf{k} + 1)\mathbf{WA}$.

► **Lemma 12.**

- $\mathbf{a} = \text{shuffle}(\langle 2^n \rangle_{n \in \mathbb{N}}, \langle 1 \rangle_{n \in \mathbb{N}})$ is in $\mathbf{1WA}$ but not in \mathbf{DetWA} ,
- \mathbf{u}_k defined by $u_n = 1^n + 2^n + \dots + (k+1)^n$ is in $(\mathbf{k} + 1)\mathbf{WA}$ but not in \mathbf{kWA} ,
- \mathbf{v} defined by $v_n = n$ is in \mathbf{PolyWA} but not in \mathbf{FinWA} ;
- *Fibonacci* is in \mathbf{WA} but not in \mathbf{PolyWA} .

We omit the simple but technical proofs of the first three items. Only the last item will be proved in Section 5, it follows from the fact that $\mathbf{PolyWA} = \mathbf{PolyRat}$ is equal to the class of LRS whose eigenvalues are roots of rational numbers. As mentioned in Example 3 the characteristic polynomial of the Fibonacci sequence is $x^2 - x - 1$, so its eigenvalues are not roots of rationals.

4 Characterisation with copyleft cost-register automata

Cost-register automata (CRA) [3] are deterministic automata with write-only registers, where each transition updates the registers using addition and multiplication. Like in Section 3 we will consider only the variant of the model over a one-letter alphabet recognising functions $f : \mathbb{N} \rightarrow \mathbb{Q}$.

Let \mathcal{X} be a set of *variables (registers)*. The set of *expressions* $\text{Expr}(\mathcal{X})$ is generated by the following grammar

$$e ::= x \mid r \mid e + e \mid e \cdot e,$$

where $x \in \mathcal{X}$ and $r \in \mathbb{Q}$. A *substitution* is a mapping $\nu : \mathcal{X} \rightarrow \text{Expr}(\mathcal{X})$. We let $\text{Subs}(\mathcal{X})$ denote the set of all substitutions. A *valuation* is a function $\sigma : \mathcal{X} \rightarrow \mathbb{Q}$, it is a special case of substitutions, where expressions are limited to constants. We freely compose these objects: for instance let $\mathcal{X} = \{x\}$, define the valuation $\nu_0(x) = 0$, the substitution $\sigma(x) = x + 1$ and the expression $e = 2x$. Then $\nu_0 \circ \sigma^n \circ e = 2n$. Note that we use the non-standard order for functional composition. We see this computation as the output of a 1-register machine which initialises x with 0, increments its value at each step and outputs its double value.

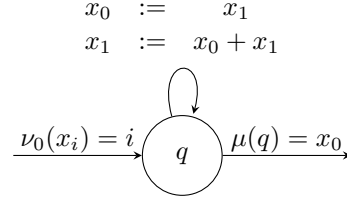
Formally, a CRA is a tuple $\mathcal{A} = (Q, \mathcal{X}, \delta, q_0, \nu_0, \mu)$, where Q is the set of states, \mathcal{X} is the set of registers, $\delta : Q \rightarrow Q \times \text{Subs}(\mathcal{X})$ is the transition function, q_0 is the initial state, $\nu_0 : \mathcal{X} \rightarrow \mathbb{Q}$ is the initial valuation and $\mu : Q \rightarrow \mathbb{Q}$ is the final output function. The output of \mathcal{A} on n is defined by the unique run of length n : let $q_0 \rightarrow q_1 \rightarrow \dots \rightarrow q_n$ such that $\delta(q_i) = (q_{i+1}, \sigma_{i+1})$

$$\llbracket \mathcal{A} \rrbracket(n) = \nu_0 \circ \sigma_1 \circ \dots \circ \sigma_n \circ \mu(q_n).$$

A CRA is said to be linear if its transitions and output function use only linear expressions, i.e. such that in the grammar $e \cdot e$ is restricted to $e \cdot r$. We let \mathbf{LCRA} denote the class of sequences recognised by linear CRA, which is known to be equivalent to the class \mathbf{WA} [3]. The linear CRA represented in Figure 5 recognises the Fibonacci sequence.

A substitution σ is called *copyleft* if each register is used at most once in $\sigma(x)$ for every x . It is easy to observe that a composition of copyleft substitutions is a copyleft substitution.

9:12 A Robust Class of Linear Recurrence Sequences



■ **Figure 5** A linear CRA recognising the Fibonacci sequence. There is only one state and two variables $\mathcal{X} = \{x_0, x_1\}$. Since there is only one state the transitions are presented using only the expression that is applied every time.

A CRA is said to be copyless if in each transition, each substitution is copyless. For example in Figure 5 the register x_1 is used twice in the substitution so it is not a copyless automaton. We let **CCRA** denote the class of sequences recognised by copyless cost register automata (CCRA). In [13] it is shown that **CCRA** is a subclass of linear CRA. We show that this is another class characterising **PolyRat**.

► **Theorem 13.** **PolyRat = CCRA**

PolyRat \subseteq CCRA

This inclusion is easy to prove, it requires to perform the classical constructions as in Section 3 and to note that they respect the copyless restriction.

CCRA \subseteq PolyRat

We make use of a simple property in [15]. A substitution is in *normal form* if there exists an order on the registers $x_1 < \dots < x_k$ such that the substitutions updating registers respect the order: $\sigma(x_i)$ can use only registers x_j such that $x_j \geq x_i$. A CCRA is in normal form if all substitutions used by it are in normal form, with the same order on the registers. It is known that every CCRA has an equivalent CCRA in normal form [15, Proposition 1]. We will use this fact only to prove Lemma 14, but in the construction we will assume that the CCRA is in normal form.

Consider a CCRA \mathcal{A} , we prove that the sequence \mathbf{u} it recognises is in **PolyRat**. We assume without loss of generality that \mathcal{A} is in normal form. Since \mathcal{A} is deterministic it has the shape of a lasso: a tail of length k and a loop of length ℓ . Let us fix $n \in \mathbb{N}$ and $\ell' \in \{0, \dots, \ell - 1\}$, the run is

$$q_0 \rightarrow \dots \rightarrow q_k \rightarrow (p_0 \rightarrow \dots \rightarrow p_{\ell-1})^n \rightarrow p_0 \rightarrow \dots \rightarrow p_{\ell'} \quad (2)$$

Let $\delta(q_i) = (q_{i+1 \bmod \ell}, \beta_i)$ for $i \in \{0, \dots, k\}$, with the convention that $q_{k+1} = p_0$, and $\delta(p_i) = (p_{i+1 \bmod \ell}, \sigma_i)$ for $i \in \{0, \dots, \ell - 1\}$. Define

$$\nu'_0 = \nu_0 \circ \beta_0 \circ \dots \circ \beta_k \quad ; \quad \sigma = \sigma_0 \circ \dots \circ \sigma_{\ell-1} \quad ; \quad e = \sigma_0 \circ \dots \circ \sigma_{\ell-1} \circ \mu(p_{\ell'})$$

Notice that σ is a copyless substitution since it is a composition of copyless substitutions. We define the sequence $\mathbf{u}[\ell']$ by

$$u_n[\ell'] = \nu'_0 \circ \sigma^n \circ e$$

We will prove in Lemma 14 that the sequence $\mathbf{u}[\ell']$ is in **PolyRat**. The decomposition of the runs into a lasso implies the following equality:

$$\mathbf{u} = \langle u_0, u_1, \dots, u_{k-1}, \text{shuffle}(\mathbf{u}[0], \dots, \mathbf{u}[\ell - 1]) \rangle,$$

which implies that \mathbf{u} is in **PolyRat**, provided the lemma below is true.

► **Lemma 14.** *For every copyless substitution σ in normal form, for all initial valuation ν and for all expression e , the sequence*

$$\langle \nu \circ \sigma^n \circ e \rangle_{n \in \mathbb{N}}$$

*is in **PolyRat**.*

Proof. We prove that the sequence $\mathbf{u}_x = \nu \circ \sigma^n(x)$ is in **PolyRat** for every register x , i.e. the lemma holds for $e = x$. The general case follows since **PolyRat** is closed under addition and product.

We consider two cases. Suppose x is not used in $\sigma(x)$. We prove that for n big enough the sequence stabilises, i.e. $\sigma^n(x) = \sigma^{n+1}(x) = c$ for some constant c . We show this by induction on the order $<$ from the assumed normal form. If x is the largest element in the order $<$ then $\sigma(x)$ is a constant and thus $\sigma^n(x) = \sigma^{n+1}(x)$. For the induction step suppose x is not the largest element. If $\sigma(x)$ is a constant then the claim is trivial. Otherwise let x_1, \dots, x_m be registers used in $\sigma(x)$. Since σ is copyless then x_i is not used in $\sigma(x_i)$ for every i . Hence by the induction assumption for every i there exists n_i such that $\sigma^n(x_i) = \sigma^{n+1}(x_i)$ for all $n \geq n_i$. It suffices to take $n = \max_i \{n_i \mid 1 \leq i \leq m\} + 1$. Since constant sequences are geometric sequences with $\lambda = 1$ then \mathbf{u}_x can be defined in **PolyRat** using shift.

Now suppose that x is used in $\sigma(x)$. The expression $\sigma(x)$ is equivalent to $\sum_{i=0}^m a_i \cdot x_i$ for some constants a_i , where $x_0 = x$ and x_i are pairwise different. Since σ is copyless then for all $i > 0$ we know that $\sigma(x_i)$ does not use x_i . By the previous paragraph there exists N such that $\sigma^N(x_i) = \sigma^{N+1}(x_i) = c_i$ for some constants c_i for all $i > 0$. Let $n \geq N$. Then

$$\nu \circ \sigma^{n+1}(x) = \nu \circ \sigma^n \circ \sigma(x) = \nu \circ \left(\sum_{i=0}^m a_i \cdot \sigma^n(x_i) \right) = a_0 \cdot (\nu \circ \sigma^n(x)) + \sum_{i=1}^m a_i \cdot c_i.$$

Let $a = a_0$ and $b = \sum_{i=1}^m a_i \cdot c_i$. We proved that for $n \geq N$ the sequence \mathbf{u}_x satisfies $u_x(n+1) = a \cdot u_x(n) + b$. It remains to prove that this sequence is in **PolyRat**. It is enough to show that $\mathbf{u}'_x(n) = \mathbf{u}_x(n+N)$ is in **PolyRat** since to obtain \mathbf{u}_x it suffices to use shift N times. There are two cases. If $a = 1$ then $\mathbf{u}'_x(n)$ is an arithmetic sequence, which concludes the proof. If $a \neq 1$ then

$$u'_x(n) = a^n \cdot u'_x(0) + \sum_{i=0}^{n-1} a^i \cdot b = a^n \cdot u'_x(0) + b \cdot \frac{a^n - 1}{a - 1}.$$

This is a sum of a geometric sequence $a^n \cdot (u'_x(0) + \frac{b}{a-1})$; and a constant sequence $-\frac{b}{a-1}$; which proves \mathbf{u}'_x is in **PolyRat**. ◀

► **Remark 15.** One can extract from this proof the equivalence between linear **CCRA** and **Rat[Arith \cup Geo, +, shift, shuffle]**.

It was recently shown that **CCRA** are strictly less expressive than weighted automata [15]. The proof goes by analysing the Fibonacci sequence. We will get as a corollary of our results a self-contained proof that **LCRA** and **CCRA** are different.

5 Characterisation with linear recurrence sequences and formal series

Our last two characterisations are as follows.

► **Theorem 16.** **PolyRat** is the class of LRS whose eigenvalues are roots of rational numbers, and equivalently whose formal series are $\frac{P}{Q}$ with P, Q rational polynomials and the roots of Q are roots of rational numbers.

Before proving the theorem, we note that we can now substantiate the claim that the Fibonacci sequence is not in **PolyRat** (hence not in **CCRA** and **PolyWA**), since its eigenvalues are not roots of rational numbers.

We rely on the following classical result about LRS, see e.g. [9].

► **Lemma 17.** Let \mathbf{u} be an LRS and Q its characteristic polynomial. The formal series induced by \mathbf{u} is $\frac{P}{Q}$ for some rational polynomial P .

For both inclusions we rely on Theorem 6 stating that **PolyRat** = **PolyWA** and the decompositions obtained in the subsequent lemmas.

PolyRat \subseteq LRS whose eigenvalues are roots of rational numbers

By Lemma 7 and Lemma 8 the formal series of sequences in **PolyWA** are sums and Cauchy products of formal series of the form $\frac{R}{1-\lambda x^\ell}$, where R is a rational polynomial, $\ell \in \mathbb{N}$ and $\lambda \in \mathbb{Q}$. The roots of $1 - \lambda x^\ell$ are roots of $\frac{1}{\lambda}$, so the roots of the characteristic polynomial are roots of rational numbers.

LRS whose eigenvalues are roots of rational numbers \subseteq **PolyRat**

Consider an LRS whose eigenvalues are roots of rational numbers. Thanks to Lemma 17 the formal series it induces is $\frac{P}{Q}$ with P, Q rational polynomials and the roots of Q are roots of rational numbers. By Lemma 9 the formal series can be written as a sum of formal series of the form $\frac{R}{(1-\lambda x^\ell)^k}$ for rational polynomials R , rational number λ , and ℓ, k natural numbers. It follows from Lemma 10 and the closure of **PolyRat** under sum and shift that such sequences belong to **PolyRat**.

6 Conclusion

We introduced a class of linear recurrence sequences and obtained several characterisations. The most surprising equivalence is **CCRA** = **PolyWA**. This equality is very particular to our setting: for instance the two classes are incomparable, i.e. neither of the inclusions hold, for tropical semirings [15, 14]. We also conjecture that these classes are incomparable over the rational semiring for general alphabets (of size bigger than 1).

We leave open the precise complexity of the Skolem problem for **PolyRat**. Recent progress has been made for a subclass of **PolyRat** [1]: the Skolem problem for LRS whose eigenvalues are roots of unity is NP-complete. Our class is more general since we consider LRS whose eigenvalues are roots of rational numbers, so the NP-hardness also applies. However the algorithm constructed in [1] does not extend to our class.

References

- 1 S. Akshay, Nikhil Balaji, and Nikhil Vyas. Complexity of Restricted Variants of Skolem and Related Problems. In *42nd International Symposium on Mathematical Foundations of Computer Science, MFCS 2017, August 21-25, 2017 - Aalborg, Denmark*, pages 78:1–78:14, 2017. doi:10.4230/LIPIcs.MFCS.2017.78.
- 2 Shaull Almagor, Michaël Cadilhac, Filip Mazowiecki, and Guillermo A. Pérez. Weak Cost Register Automata Are Still Powerful. In *Developments in Language Theory - 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings*, pages 83–95, 2018. doi:10.1007/978-3-319-98654-8_7.
- 3 Rajeev Alur, Loris D’Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular Functions and Cost Register Automata. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 13–22, 2013. doi:10.1109/LICS.2013.65.
- 4 Corentin Barloy, Nathanaël Fijalkow, Nathan Lhote, and Filip Mazowiecki. A Robust Class of Linear Recurrence Sequences. *CoRR*, abs/1908.03890, 2019. arXiv:1908.03890.
- 5 Mireille Bousquet-Mélou. Algebraic Generating Functions in Enumerative Combinatorics and Context-Free Languages. In *STACS 2005, 22nd Annual Symposium on Theoretical Aspects of Computer Science, Stuttgart, Germany, February 24-26, 2005, Proceedings*, pages 18–35, 2005. doi:10.1007/978-3-540-31856-9_2.
- 6 Manfred Droste and Paul Gastin. Weighted automata and weighted logics. *Theoretical Computer Science*, 380(1-2):69–86, 2007. doi:10.1016/j.tcs.2007.02.055.
- 7 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata*. Springer, 1st edition, 2009.
- 8 Nathanaël Fijalkow, Cristian Riveros, and James Worrell. Probabilistic Automata of Bounded Ambiguity. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory, CONCUR 2017, September 5-8, 2017, Berlin, Germany*, volume 85 of *LIPIcs*, pages 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.CONCUR.2017.19.
- 9 Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete mathematics - a foundation for computer science (2. ed.)*. Addison-Wesley, 1994.
- 10 Daniel Kirsten and Sylvain Lombardy. Deciding Unambiguity and Sequentiality of Polynomially Ambiguous Min-Plus Automata. In *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, pages 589–600, 2009. doi:10.4230/LIPIcs.STACS.2009.1850.
- 11 Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theoretical Computer Science*, 327(3):349–373, 2004. doi:10.1016/j.tcs.2004.02.049.
- 12 Stephan Kreutzer and Cristian Riveros. Quantitative Monadic Second-Order Logic. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 113–122, 2013. doi:10.1109/LICS.2013.16.
- 13 Filip Mazowiecki and Cristian Riveros. Maximal Partition Logic: Towards a Logical Characterization of Copyless Cost Register Automata. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, pages 144–159, 2015. doi:10.4230/LIPIcs.CSL.2015.144.
- 14 Filip Mazowiecki and Cristian Riveros. Pumping Lemmas for Weighted Automata. In *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, pages 50:1–50:14, 2018. doi:10.4230/LIPIcs.STACS.2018.50.
- 15 Filip Mazowiecki and Cristian Riveros. Copyless cost-register automata: Structure, expressiveness, and closure properties. *Journal of Computer and System Sciences*, 100:1–29, 2019. doi:10.1016/j.jcss.2018.07.002.
- 16 Joël Ouaknine and James Worrell. On the Positivity Problem for Simple Linear Recurrence Sequences,. In *Automata, Languages, and Programming - 41st International Colloquium*,

9:16 A Robust Class of Linear Recurrence Sequences

- ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 318–329, 2014. doi:10.1007/978-3-662-43951-7_27.
- 17 Joël Ouaknine and James Worrell. Ultimate Positivity is Decidable for Simple Linear Recurrence Sequences. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, pages 330–341, 2014. doi:10.1007/978-3-662-43951-7_28.
 - 18 Joël Ouaknine and James Worrell. On linear recurrence sequences and loop termination. *SIGLOG News*, 2(2):4–13, 2015. doi:10.1145/2766189.2766191.
 - 19 Rachid Rebiha, Arnaldo Vieira Moura, and Nadir Matringe. On the Termination of Linear and Affine Programs over the Integers. *CoRR*, abs/1409.4230, 2014. arXiv:1409.4230.
 - 20 Marcel Paul Schützenberger. On the Definition of a Family of Automata. *Information and Control*, 4(2-3):245–270, 1961. doi:10.1016/S0019-9958(61)80020-X.
 - 21 Terence Tao. *Structure and randomness: pages from year one of a mathematical blog*. American Mathematical Society Providence, RI, 2008.

Coverage and Vacuity in Network Formation Games

Gili Bielous

School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel
gili.bielous@mail.huji.ac.il

Orna Kupferman

School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel
orna@cs.huji.ac.il

Abstract

The frameworks of coverage and vacuity in formal verification analyze the effect of mutations applied to systems or their specifications. We adopt these notions to network formation games, analyzing the effect of a change in the cost of a resource. We consider two measures to be affected: the cost of the Social Optimum and extremums of costs of Nash Equilibria. Our results offer a formal framework to the effect of mutations in network formation games and include a complexity analysis of related decision problems. They also tighten the relation between algorithmic game theory and formal verification, suggesting refined definitions of coverage and vacuity for the latter.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory; Theory of computation → Network games; Theory of computation → Network formation; Software and its engineering → Formal methods

Keywords and phrases Network Formation Games, Vacuity, Coverage

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.10

Related Version A full version of the paper is available at <https://www.cs.huji.ac.il/~ornak/publications/csl20.pdf>.

1 Introduction

Following the emergence of the Internet, there has been an explosion of studies employing game-theoretic analysis to explore applications such as network formation and routing in computer networks [21, 1, 20, 4]. In *network-formation games* (for a survey, see [37]), the network is modeled by a weighted graph. The weight of an edge indicates the cost of activating the transition it models, which is independent of the number of times the edge is used. Players have reachability objectives, each given by a source and a target vertex. Under the common Shapley cost-sharing mechanism, the cost of an edge is shared evenly by the players that use it. The players are selfish agents who attempt to minimize their own costs, rather than to optimize some global objective. In network-design settings, this would mean that the players selfishly select a path instead of being assigned one by a central authority. The study of networks from a game-theoretic point of view focuses on optimal strategies for the underlying players, stable outcomes of a given setting, namely equilibrium points, and outcomes that are optimal for the society as a whole.

A different type of reasoning about networks is the study of their on-going behaviors. In particular, in recent years we see growing use of formal-verification methods in the context of software-defined networks [34, 33]. The study of networks from a formal-verification point of view focuses on specification and verification of their behavior. The primary problem here is *model checking*: given a system (in particular, a network) and a specification for its desired behavior, decide whether the system satisfies the specification [18]. Typically, the system is given by means of a labeled graph and the specification is given by a temporal-logic



© Gili Bielous and Orna Kupferman;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 10; pp. 10:1–10:18

Leibniz International Proceedings in Informatics



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

formula. An important element in model-checking methodologies is an assessment of the quality of the modeling of the system and the specifications as well as the exhaustiveness of the model-checking process. Researchers have developed a number of *sanity checks*, aiming to detect errors in the modeling [27]. Two leading sanity checks are *vacuity* and *coverage*. In vacuity, the goal is to detect cases where the system satisfies the specification in some unintended trivial way [10, 31, 14]. In coverage, the goal is to increase the exhaustiveness of the specification by detecting components of the system that do not play a role in the verification process [24, 25, 16, 15]. Both vacuity and coverage checks are based on analyzing the effect of applying *local mutations* to the system or the specification. The intuition is that model checking of an exhaustive well-formed specification should be sensitive to such mutations.

Beyond the practical importance of sanity checks, their study highlights some general important theoretical properties regarding the sensitivity of systems and specifications to mutations. Examples to such properties include *duality* between mutations applied to the system and the specification [29], and trade-offs between desired and undesired insensitivity to mutations (for example, fault tolerance is associated with a desired insensitivity to mutations) [17]. A fundamental property of mutations in the context of formal verification is *monotonicity*: mutations to temporal-logic formulas are monotone, in the sense that if ψ is a formula and φ is a sub-formula of ψ that appears in a positive polarity (that is, nested in an even number of negations), then when we mutate ψ to ψ' by replacing φ by φ' , then $\psi' \rightarrow \psi$ iff $\varphi' \rightarrow \varphi$. Monotonicity turns out to be a very helpful property in the context of vacuity checking. Indeed, the basic notion in vacuity is of a subformula φ *not affecting* the satisfaction of a specification ψ . Formally, consider a system \mathcal{S} satisfying a specification ψ . A subformula φ of ψ *does not affect* (the satisfaction of) ψ in \mathcal{S} if \mathcal{S} also satisfies all specifications obtained by mutating φ to some other subformula [10]. Thanks to monotonicity, we can check whether φ affects ψ by examining only the most challenging mutation, namely one that replaces φ by false and the most helpful mutation, namely one that replaces φ by true.

Our goal in this paper is to examine the sensitivity of network-formation games (NFGs, for short) to mutations applied to costs. While our study adopts from formal verification the notion of mutation-based analysis, we examine the effect of mutations on measures from game theory: the cost of stable and optimal outcomes. Recall that a strategy of a player in an NFG is a path from a source to a target vertex. A *profile* in the game is a vector of strategies, one for each player. A *Social Optimum* (SO) is a profile that minimizes the total cost to all players. A *Nash equilibrium* (NE) is a profile in which no player can decrease her cost by a unilateral deviation from her current strategy, that is, assuming that the strategies of the other players do not change.

Consider an NFG N . We say that the edge e of N *SO-affects* N if a change in the cost of e leads to a change in the cost of the SO. Formally, there exists $x \geq 0$ such that the cost of the SO profiles in N is different from the cost of the SO profiles in $N[e \leftarrow x]$, that is N with e being assigned cost x . We consider the function $cost_{SO}^e(N) : \mathbb{R} \rightarrow \mathbb{R}$, mapping a cost $x \geq 0$ to the cost of the SO profiles in $N[e \leftarrow x]$. That is, $cost_{SO}^e(N)$ describes the cost of the SO in N as a function of the cost of the edge e . We say that $cost_{SO}$ is monotonically increasing if for every NFG N and edge e of N , the function $cost_{SO}^e(N)$ is monotonically increasing. Likewise, $cost_{SO}$ is continuous if for every NFG N and edge e , the function $cost_{SO}^e(N)$ is continuous. For the best and worst NEs, we similarly define when an edge e *bNE-affects* and *wNE-affects* N , and define the functions $cost_{bNE}$ and $cost_{wNE}$, which describe the cost of the best and worst NEs as a function of the cost of an edge.

Our first set of results concerns the way edge costs affect the SO. Here, the results are quite expected: $cost_{SO}$ is monotonically increasing and continuous, which leads to simple solutions to related decision problems: as is the case with model checking and temporal-logic specifications, we can decide whether an edge e SO-affects N by checking the cost of the SO in $N[e \leftarrow 0]$ and $N[e \leftarrow \infty_N]$, for a sufficiently large cost ∞_N . This leads to Δ_2^P and Θ_2^P upper bounds (depending on whether costs are given in binary or unary, respectively), which we show to be tight. Also, we show that it is NP-complete and DP-complete to decide whether we can mutate a cost in a way that would cause the SO to be below or agree exactly with, respectively, a given threshold. The technically challenging results here are the Δ_2^P -lower bound (it is tempting to believe that thanks to monotonicity, we could decide whether e SO-affects N using only logarithmically many queries to an NP oracle that bounds the SO) and the DP upper bound (the upper and lower bounds on the SO that we can obtain by querying an NP and a co-NP oracle need not be associated with the same edge).

Things become unexpected when we turn to study effects on the costs of the best and worst NEs. Here an edge may affect the bNE without participating in profiles that are NEs, and may thus affect the bNE both positively and negatively. In model checking, this is related to coverage and vacuity in a setting with multiple occurrences of subformulas. For example, the atomic proposition p appears in the formula $\psi = (\varphi_1 \rightarrow p) \wedge (p \rightarrow \varphi_2)$ both positively and negatively. Consequently, we cannot decide whether p affects the satisfaction of ψ by examining its replacement by only true or false (in the context of vacuity), and we do not know the effect of mutating p in the system on the satisfaction of ψ (in the context of coverage). We show that $cost_{bNE}$ is neither monotone nor continuous, and in fact a change in the cost of an edge may incentivize players in surprising ways. In particular (see Figure 5), an edge e may not participate in any bNE in $N[e \leftarrow x]$, for all $x \geq 0$, and still the bNE may decrease as we increase the cost of e . We show that these challenges can be overcome by more restricted notions such as piecewise monotonicity and monotonicity on the participation of the mutated edge in bNE profiles. In particular, we show that these notions produce the same (tight) complexity bounds for the analogous decision problems we introduce for the SO. We note that while the general phenomenon of non-monotonicity is known (e.g., Braess' Paradox [12], the effectiveness of burning money [23, 36] or tax increase [19]), we are the first, to the best of our knowledge, to provide a comprehensive study of effects caused by cost mutation.

Our results on NFGs give rise to two research directions in coverage and vacuity in formal verification. The first arises from the segmentation of \mathbb{R}^+ induced by the non-monotonicity of the bNE, which suggests a similar segmentation in the context of multi-valued specification formalisms [2]. The second is a study of coverage and vacuity in formalisms for specifying strategic on-going behaviors [3, 13]. We discuss these research directions in Section 5.

Due to lack of space, some of the proofs are omitted, and can be found in the full version, as listed above.

2 Preliminaries

2.1 Network formation games

A *network formation game* (NFG) is $N = \langle k, V, E, c, \gamma \rangle$, where k is a number of players, V is a set of vertices, $E \subseteq V \times V$ is a set of directed edges, $c : E \rightarrow \mathbb{R}^+$, where \mathbb{R}^+ is the set of positive real numbers including 0, is a cost function that maps each edge to the cost of forming it, and $\gamma = \{\langle s_1, t_1 \rangle, \dots, \langle s_k, t_k \rangle\}$ is a set of objectives, each specifying a source and a target vertex per player. Thus, for all $1 \leq i \leq k$, the objective of player i is to form a path

from s_i to t_i . A *strategy* for player i is a simple path $\pi_i \subseteq E$ from s_i to t_i . Note that since the path is simple, then π_i is indeed a subset of E . A *profile* $P = \langle \pi_1, \dots, \pi_k \rangle$ is a vector of strategies, one for each player. For an edge $e \in E$, we denote by $used_P(e)$ the number of players that use e in their strategy in P , thus these with $e \in \pi_i$. We say that $e \in P$ if $used_P(e) > 0$.

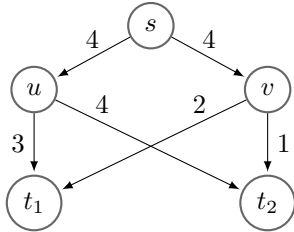
Players pay the cost of forming edges they use. If players share an edge, they also share its cost. Thus, the cost of a strategy π_i in a profile P is $cost_{N,P}(\pi_i) = \sum_{e \in \pi_i} \frac{c(e)}{used_P(e)}$. Note that since c is positive, it is indeed sufficient to consider only simple paths as strategies. The *cost* of P in N is the sum of costs of its strategies, that is $cost(N, P) = \sum_{i=1}^k cost_{N,P}(\pi_i)$. Equivalently, $cost(N, P) = \sum_{e \in P} c(e)$.

A *Social Optimum (SO)* of N is a profile with minimal cost. That is, a profile P is an SO if for every other profile P' we have that $cost(N, P) \leq cost(N, P')$. Note that there may be several profiles that are a social optimum. We denote by $SO(N)$ and $cost_{SO}(N)$ the set of such profiles and their cost, respectively.

We say that the profile P is a *Nash Equilibrium (NE)* in N if no player can decrease her cost by deviating to another strategy assuming the other players stay in their strategies¹. Formally, for all $1 \leq i \leq k$ and every $\pi'_i \neq \pi_i$, the cost of π'_i in $P' = \langle \pi_1, \dots, \pi_{i-1}, \pi'_i, \pi_{i+1}, \dots, \pi_k \rangle$ is no lower than the cost of π_i in P , i.e. $cost_{N,P}(\pi_i) \leq cost_{N,P'}(\pi'_i)$. A *best NE (bNE)* in N is an NE profile with minimal cost, i.e. a profile P is bNE iff P is an NE, and for every profile P' that is an NE, we have $cost(N, P) \leq cost(N, P')$. We denote by $bNE(N)$ and $cost_{bNE}(N)$ the set of profiles that are bNE, and their cost, respectively.

We dually define a *worst NE (wNE)* to be an NE profile with maximal cost, and denote by $wNE(N)$ and $cost_{wNE}(N)$ the set of such profiles and their cost, respectively. The *Price of Stability (PoS)* of N is the ratio between the cost of the bNE and the SO, that is, $PoS(N) = \frac{cost_{bNE}(N)}{cost_{SO}(N)}$.

► **Example 1.** Consider the NFG N appearing in Figure 1.



■ **Figure 1** The NFG N .

■ **Table 1** Players' costs in N .

Player 2	π_2^1	π_2^2
Player 1	$s \rightarrow u \rightarrow t_2$	$s \rightarrow v \rightarrow t_2$
π_1^1	6	5
$s \rightarrow u \rightarrow t_1$	5	7
π_1^2	8	3
$s \rightarrow v \rightarrow t_1$	6	4

Assume that N is formed by two players. The first has objective $\langle s, t_1 \rangle$. The available strategies for her are $\pi_1^1 = \{(s, u), (u, t_1)\}$ and $\pi_1^2 = \{(s, v), (v, t_1)\}$. The second player has objective $\langle s, t_2 \rangle$. The available strategies for her are $\pi_2^1 = \{(s, u), (u, t_2)\}$ and $\pi_2^2 = \{(s, v), (v, t_2)\}$. If Player 1 chooses the strategy π_1^1 and Player 2 uses the strategy π_2^1 , then they share the cost of the edge (s, u) , and their costs are $\frac{4}{2} + 3 = 5$ and $\frac{4}{2} + 4 = 6$ respectively. Table 1 describes the costs of the two players in the different profiles.

The profile with the lowest cost is $P = \langle \pi_1^2, \pi_2^2 \rangle$. Therefore, $SO(N) = \{P\}$, with cost $cost_{SO}(N) = 7$. Note that P is also the only NE in N . It is an NE since for the deviation $P' = \langle \pi_1^1, \pi_2^2 \rangle$, it holds that $4 = cost_{N,P}(\pi_1^2) < cost_{N,P'}(\pi_1^1) = 7$ and for the deviation

¹ Throughout this paper, we consider pure strategies and pure deviations, as is the case for the vast literature on cost-sharing games.

$P'' = \langle \pi_1^2, \pi_2^1 \rangle$ it holds that $3 = \text{cost}_{N,P}(\pi_2^2) < \text{cost}_{N,P''}(\pi_2^1) = 8$. It is the only NE in N since for every other profile there is a beneficial deviation. Therefore, P is both a bNE and a wNE. Since the bNE and the SO coincide, it follows that $\text{PoS}(N) = 1$. ◀

Consider an edge $e \in E$ and a value $x \in \mathbb{R}^+$. We denote by $c[e \leftarrow x]$ the cost function that agrees with c on every edge except e , which is assigned x . That is, $c[e \leftarrow x](e) = x$, and for all edge $e' \neq e$, we have $c[e \leftarrow x](e') = c(e')$. Let $N = \langle k, V, E, c, \gamma \rangle$, and let $e \in E$. We denote by $N[e \leftarrow x]$ the network obtained from N by changing the cost of e to x . Thus, $N[e \leftarrow x] = \langle k, V, E, c[e \leftarrow x], \gamma \rangle$.

Let c_1 and c_2 be cost functions. We say that c_2 *bounds* c_1 *from above*, denoted $c_1 \leq c_2$, if for all $e \in E$, we have $c_1(e) \leq c_2(e)$. We extend the notation to NFGs. Let $N_1 = \langle k, V, E, c_1, \gamma \rangle$ and $N_2 = \langle k, V, E, c_2, \gamma \rangle$ be two NFGs that differ only on their cost functions. If $c_1 \leq c_2$, we say that N_2 *bounds* N_1 *from above*, denoted $N_1 \leq N_2$.

► **Lemma 2.** *Let N_1 and N_2 be two NFGs that differ only on their cost functions. If $N_1 \leq N_2$, then for every profile P , we have $\text{cost}(N_1, P) \leq \text{cost}(N_2, P)$.*

2.2 Affecting edges in NFGs

Consider an NFG N and an edge e of N . We say that the edge e *SO-affects* N if there exists $x \geq 0$ such that $\text{cost}_{\text{SO}}(N[e \leftarrow x]) \neq \text{cost}_{\text{SO}}(N)$. That is, when changing the cost of e to x , the cost of the SO profiles of N changes. We define *bNE-affects*, *wNE-affects*, and *PoS-affects* in a similar way, referring to the costs of the best and worst NEs, and the PoS.

► **Example 3.** Consider the NFG N from Example 1, and consider the edge $e = (s, v)$. The edge e SO-affects N , since, for example, for $N[e \leftarrow 2]$ we have that $\langle \pi_1^2, \pi_2^2 \rangle$ is an SO with cost $5 < 7 = \text{cost}_{\text{SO}}(N)$. As another example, for $N[e \leftarrow 10]$ we have that $\langle \pi_1^1, \pi_2^1 \rangle$ is an SO with cost $11 > 7 = \text{cost}_{\text{SO}}(N)$. Next, consider the edge $e = (u, t_1)$. For every $x \geq 0$, we have $\text{cost}(N[e \leftarrow x], \langle \pi_1^1, \pi_2^1 \rangle) = x + 8$, $\text{cost}(N[e \leftarrow x], \langle \pi_1^1, \pi_2^2 \rangle) = x + 9$, $\text{cost}(N[e \leftarrow x], \langle \pi_1^2, \pi_2^1 \rangle) = 14$, and $\text{cost}(N[e \leftarrow x], \langle \pi_1^2, \pi_2^2 \rangle) = 7$. Therefore, $\text{cost}_{\text{SO}}(N[e \leftarrow x]) = \min\{x + 8, x + 9, 14, 7\} = 7 = \text{cost}_{\text{SO}}(N)$, and so e does not SO-affect N .

We proceed to bNE and wNE. Here, the change may affect the stability of profiles, and not just their cost. Consider the edge $e = (s, u)$. Table 2 describes the costs of the different profiles of $N[e \leftarrow (1 - \varepsilon)]$, for some $0 < \varepsilon < 1$.

■ **Table 2** Costs in $N[\langle s, u \rangle \leftarrow (1 - \varepsilon)]$.

Player 2	π_1^1	π_2^2
Player 1	$s \rightarrow u \rightarrow t_2$	$s \rightarrow v \rightarrow t_2$
π_1^1	$4\frac{1}{2} - \frac{\varepsilon}{2}$	5
$s \rightarrow u \rightarrow t_1$	$3\frac{1}{2} - \frac{\varepsilon}{2}$	$4 - \varepsilon$
π_1^2	$5 - \varepsilon$	3
$s \rightarrow v \rightarrow t_1$	6	4

■ **Table 3** Costs in $N[\langle u, t_1 \rangle \leftarrow x]$.

Player 2	π_1^1	π_2^2
Player 1	$s \rightarrow u \rightarrow t_2$	$s \rightarrow v \rightarrow t_2$
π_1^1	6	5
$s \rightarrow u \rightarrow t_1$	$2 + x$	$4 + x$
π_1^2	8	3
$s \rightarrow v \rightarrow t_1$	6	4

We previously saw that the only NE profile in N is $P = \langle \pi_1^2, \pi_2^2 \rangle$, with cost 7, and therefore it is both the bNE and the wNE. We can see that the cost of P is minimal for $N[e \leftarrow (1 - \varepsilon)]$. However, P is no longer an NE. Indeed, for the profile $P' = \langle \pi_1^1, \pi_2^2 \rangle$, obtained by a deviation of Player 1, we have that $4 - \varepsilon = \text{cost}_{N[e \leftarrow (1 - \varepsilon)], P'}(\pi_1^1) < \text{cost}_{N[e \leftarrow (1 - \varepsilon)], P}(\pi_1^2) = 4$. For $N[e \leftarrow (1 - \varepsilon)]$, the only NE profile is $\langle \pi_1^1, \pi_2^1 \rangle$, with cost $8 - \varepsilon$. For $0 < \varepsilon < 1$ it therefore holds that $7 = \text{cost}_{\text{bNE}}(N) < \text{cost}_{\text{bNE}}(N[e \leftarrow (1 - \varepsilon)]) = 8 - \varepsilon$, and the same for wNE.

10:6 Coverage and Vacuity in Network Formation Games

Therefore, the edge e both bNE-affects and wNE-affects N . Furthermore, e PoS-affects N , as $PoS(N) = 1$ and $PoS(N[e \leftarrow 1 - \varepsilon]) = \frac{8-\varepsilon}{7} > 1$.

Next, consider the edge $e = (u, t_1)$. We show that e does not bNE-affect nor does it wNE-affect N . To see this, consider the costs of the different profiles of $N[e \leftarrow x]$ for $x \geq 0$, described in Table 3. It can be easily verified that, for all $x \geq 0$, the only NE in $N[e \leftarrow x]$ is $\langle \pi_1^1, \pi_2^2 \rangle$. Therefore, $cost_{bNE}(N[e \leftarrow x]) = cost_{wNE}(N[e \leftarrow x]) = 7$. As e neither SO-affect nor bNE-affect N , it follows that e does not PoS-affect N .

It is also worth noting that it is not always the case that an edge either both bNE-affects and wNE-affects or both does not bNE-affect and wNE-affect N . As an example, consider the edge $e = (u, t_2)$. The cost table of $N[e \leftarrow x]$ appears in Table 4.

■ **Table 4** Costs in $N[\langle u, t_2 \rangle \leftarrow x]$.

Player 2	π_2^1	π_2^2
Player 1	$s \rightarrow u \rightarrow t_2$	$s \rightarrow v \rightarrow t_2$
π_1^1	$2 + x$	5
$s \rightarrow u \rightarrow t_1$	5	7
π_1^2	$4 + x$	3
$s \rightarrow v \rightarrow t_1$	6	4

It is not hard to see that for $0 \leq x \leq 3$, it holds that $P_1 = \langle \pi_1^1, \pi_2^1 \rangle$ and $P_2 = \langle \pi_1^2, \pi_2^2 \rangle$ are NEs in $N[e \leftarrow x]$. However, $cost(N[e \leftarrow x], P_1) = 7 + x$ and $cost(N[e \leftarrow x], P_2) = 7$. Therefore, $cost_{bNE}(N[e \leftarrow x]) = \min\{7 + x, 7\} = 7$, and $cost_{wNE}(N[e \leftarrow x]) = \max\{7 + x, 7\} = 7 + x$. Since for all $x > 3$, the profile P_2 is the only NE in $N[e \leftarrow x]$, it follows that e does not bNE-affect N , and e wNE-affects N . ◀

2.3 Monotonicity and continuity

Consider a function $f : \mathbb{R} \rightarrow \mathbb{R}$. We say that f is *monotonically increasing* if for all $x_1, x_2 \in \mathbb{R}$, we have that $x_1 \leq x_2$ implies $f(x_1) \leq f(x_2)$. For $x_0 \in \mathbb{R}$, we say that f is *continuous at x_0* if for every $\varepsilon > 0$ there exists $\delta > 0$ such that for all $x \in \mathbb{R}$, if $|x - x_0| < \delta$ then $|f(x) - f(x_0)| < \varepsilon$. Then, we say that f is *continuous* if f is continuous at x_0 for all $x_0 \in \mathbb{R}$.

For an edge $e \in E$, we define the function $cost_{SO}^e(N) : \mathbb{R} \rightarrow \mathbb{R}$ by $cost_{SO}^e(N)(x) = cost_{SO}(N[e \leftarrow x])$ if $x \geq 0$, and $cost_{SO}^e(N)(x) = cost_{SO}(N[e \leftarrow 0])$ otherwise. That is, $cost_{SO}^e(N)$ is the cost of the SO in N as a function of the cost of the edge e . We say that $cost_{SO}$ is *monotonically increasing*, if for every NFG N and edge e of N , the function $cost_{SO}^e(N)$ is *monotonically increasing*. That is, $cost_{SO}$ is *monotonically increasing* if an increase in the cost of any edge, for any NFG, can only cause an increase in the cost of the SO. Likewise, $cost_{SO}$ is *continuous*, if for every NFG N and edge e , the function $cost_{SO}^e(N)$ is *continuous*. We define the *monotonicity* and the *continuity* of $cost_{bNE}$, $cost_{wNE}$ and PoS in a similar way.

3 Affecting the Social Optimum

In this section we study the sensitivity of the SO to cost mutations. We first study the *monotonicity* and *continuity* of $cost_{SO}$, and then the complexity of relevant decision problems.

3.1 Monotonicity and continuity of the SO

► **Theorem 4** ($cost_{SO}$ is monotone). *For every NFG N and edge e of N , the function $cost_{SO}^e(N)$ is monotone.*

Proof. Let N_1 and N_2 be NFGs that differ only in their cost functions. We prove that if $N_1 \leq N_2$, then $cost_{SO}(N_1) \leq cost_{SO}(N_2)$. In particular, this holds for N_1 and N_2 being N with cost functions that differ only in the cost of e . Let $P_1 \in SO(N_1)$ and let $P_2 \in SO(N_2)$. By the minimality of the SO for N_1 , we get that $cost(N_1, P_1) \leq cost(N_1, P_2)$. By Lemma 2, as $N_1 \leq N_2$, we have that $cost(N_1, P_2) \leq cost(N_2, P_2)$. Therefore, $cost(N_1, P_1) \leq cost(N_2, P_2)$, and hence $cost_{SO}(N_1) \leq cost_{SO}(N_2)$. ◀

Since $cost_{SO}$ is monotonically increasing, a sufficient condition for an edge not to SO-affect the network is based on comparing the cost of the SO in the two extreme costs for the edge. The lowest cost is 0. For the highest cost, let ∞_N be a sufficiently large value for a cost of an edge to be considered extreme in N , in the sense that if an edge e with cost ∞_N is in some strategy, then the cost of that strategy is guaranteed to be larger than the cost of all strategies that do not contain e . For example, we can define ∞_N to be $1 + \sum_{e \in E} c(e)$.

► **Lemma 5.** *For every NFG N and edge e of N , the edge e does not SO-affect N iff $cost_{SO}(N[e \leftarrow 0]) = cost_{SO}(N[e \leftarrow \infty_N])$.*

Proof. Since $N[e \leftarrow 0] \leq N[e \leftarrow \infty_N]$ and the function $cost_{SO}(N)$ is monotonically increasing, then $cost_{SO}(N[e \leftarrow 0]) = cost_{SO}(N[e \leftarrow \infty_N])$ implies that for all $x \geq 0$, we have $cost_{SO}(N[e \leftarrow 0]) = cost_{SO}(N[e \leftarrow x]) = cost_{SO}(N[e \leftarrow \infty_N])$. Thus, for all $x \geq 0$, we have $cost_{SO}(N) = cost_{SO}(N[e \leftarrow x])$, so the cost of e does not SO-affect N . For the other direction, if the cost of e does not SO-affect N , then, by definition, for all $x \geq 0$, we have that $cost_{SO}(N) = cost_{SO}(N[e \leftarrow x])$. In particular, $cost_{SO}(N[e \leftarrow 0]) = cost_{SO}(N[e \leftarrow \infty_N])$, and we are done. ◀

Note that it follows that for an NFG N and edge e in it, if there is a profile $P \in SO(N)$ such that $e \in P$ and $c(e) > 0$, then e SO-affects N , as reducing its cost to 0 reduces also the cost of the SO.

In case e SO-affects N , we can characterize the behavior of $cost_{SO}(N[e \leftarrow x])$ as follows.

► **Lemma 6.** *Consider an NFG N and an edge e of N . If e SO-affects N , then there is a value $x \in \mathbb{R}$ such that the following hold.*

1. *For all values y with $y > x$, the edge e does not participate in any profile in $SO(N[e \leftarrow y])$ and $cost_{SO}(N[e \leftarrow y]) = x + cost_{SO}(N[e \leftarrow 0])$.*
2. *For all values y with $y < x$, the edge e participates in at least one profile in $SO(N[e \leftarrow y])$ and $cost_{SO}(N[e \leftarrow y]) = y + cost_{SO}(N[e \leftarrow 0])$.*
3. *The edge e participates in at least one profile in $SO(N[e \leftarrow x])$ and $cost_{SO}(N[e \leftarrow x]) = x + cost_{SO}(N[e \leftarrow 0])$.*

Proof. Since e SO-affects N , then, by Lemma 5, we have that $cost_{SO}(N[e \leftarrow 0]) < cost_{SO}(N[e \leftarrow \infty_N])$. It is not hard to see that taking x to be $\min\{y : cost_{SO}(N[e \leftarrow y]) = cost_{SO}(N[e \leftarrow \infty_N])\}$ satisfies the conditions in the lemma. In particular, when e participates in all profiles in the SO, then $x = \min \emptyset = \infty$. ◀

► **Theorem 7.** *For every NFG N and edge e of N , the function $cost_{SO}^e(N)$ is continuous.*

Proof. Consider an NFG N and edge e of N . First, if the edge e does not SO-affect N , then $cost_{SO}^e(N)$ is constant and therefore continuous. Otherwise, by Lemma 6, there is a value $x \in \mathbb{R}$ such that for all values y with $y \geq x$, we have that $cost_{SO}(N[e \leftarrow y]) = x + cost_{SO}(N[e \leftarrow 0])$, and for all values y with $y < x$, we have that $cost_{SO}(N[e \leftarrow y]) = y + cost_{SO}(N[e \leftarrow 0])$. Thus, continuity in all points except x follows immediately from continuity of linear functions. For the point x , Lemma 6 implies that for all $\epsilon > 0$, we have that $f(x + \epsilon) - f(x) = 0$, and $f(x) - f(x - \epsilon) = \epsilon$, so $cost_{SO}^e(N)$ is continuous also at x . ◀

3.2 Decision problems

The SO-cost decision problem is the problem of deciding, given an NFG N and a threshold $\kappa \geq 0$, whether $cost_{SO}(N) \leq \kappa$. The SO-cost problem is NP-complete [37]. In this section we study the following related decision problems.

1. **Edge-SO-affects:** Given an NFG N and an edge e of N , does e SO-affect N ? Thus, $\text{Edge-SO-affects} = \{\langle N, e \rangle \mid e \text{ SO-affects } N\}$.
2. **Edge-SO-optimization:** Given an NFG N , an edge e of N , and a threshold $\kappa \geq 0$, is there a value $x \geq 0$, such that $cost_{SO}(N[e \leftarrow x]) \leq \kappa$? Thus, $\text{Edge-SO-optimization} = \{\langle N, e, \kappa \rangle \mid \text{there exists } x \geq 0 \text{ such that } cost_{SO}(N[e \leftarrow x]) \leq \kappa\}$.
3. **SO-optimization:** Given an NFG N and a threshold $\kappa \geq 0$, is there an edge e of N and a value $x \geq 0$, such that $cost_{SO}(N[e \leftarrow x]) \leq \kappa$? Thus, $\text{SO-optimization} = \{\langle N, \kappa \rangle \mid \text{there exist } e \text{ and } x \geq 0 \text{ such that } cost_{SO}(N[e \leftarrow x]) \leq \kappa\}$.
4. **SO-control:** Given an NFG N and a threshold $\kappa \geq 0$, is there an edge e of N and a value $x \geq 0$, such that $cost_{SO}(N[e \leftarrow x]) = \kappa$? Thus, $\text{SO-control} = \{\langle N, \kappa \rangle \mid \text{there exist } e \text{ and } x \geq 0 \text{ such that } cost_{SO}(N[e \leftarrow x]) = \kappa\}$.

Analyzing the complexity of the problems, we assume that the costs of an NFG are given in binary. As we shall note below, this affects the complexity of the problems. In addition to the classes NP and co-NP, we are going to refer to the class $\Delta_2^P = P^{NP}$ (Θ_2^P), of decision problems that can be decided by a polynomial-time deterministic Turing machine that has access to polynomially many (logarithmically many, respectively) queries to an oracle to an NP-complete problem, and the class DP, of decision problems that are the intersection of an NP and a co-NP problem. That is, a decision problem \mathcal{L} is in DP if there are decision problems L_1, L_2 such that $L_1 \in \text{NP}$, $L_2 \in \text{co-NP}$ and $\mathcal{L} = L_1 \cap L_2$.

► **Theorem 8.** *The Edge-SO-affects problem is Δ_2^P -complete, and is Θ_2^P complete when costs are given in unary.*

Proof. We start with membership in Δ_2^P . Given an NFG N and an edge e in N , a deterministic Turing machine can use an oracle to SO-cost, calculate $cost_{SO}(N[e \leftarrow 0])$ and $cost_{SO}(N[e \leftarrow \infty_N])$ and compare them. Since the maximal cost of a profile is $\sum_{e \in E} c(e)$, and $cost_{SO}$ is the sum of costs of a subset of edges, rather than an arbitrary number in \mathbb{R} , the Turing machine can proceed by a binary search and thus the number of oracle calls is logarithmic in $\sum_{e \in E} c(e)$. When costs are given in binary, $\sum_{e \in E} c(e)$ is exponential in input, hence there are polynomially-many oracle calls. Thus, $\text{Edge-SO-affects} \in \Delta_2^P$. However, when costs are given in unary, $\sum_{e \in E} c(e)$ is polynomial in input, hence there are logarithmically-many oracle calls. Thus, $\text{Edge-SO-affects} \in \Theta_2^P$.

In the full version, we prove that the problem is Δ_2^P -hard by a reduction from maximum-satisfying-assignment, namely the problem of deciding, given a 3CNF formula φ if the lexicographically maximal assignment that satisfies φ has LSB that equals 1. It was shown by [26] that maximum-satisfying-assignment is Δ_2^P -complete. Essentially, given φ , we construct

an NFG N such that profiles corresponds to assignments, and the cost of a profile decreases with lexicographically greater satisfying assignments. The edge e participates in profiles that correspond to assignments in which the LSB is 1, and is minimal only when the maximal lexicographic assignment has LSB 1. Consequently, $\langle N, e \rangle \in \text{Edge-SO-affects}$ iff $\varphi \in \text{maximum-satisfying-assignment}$.

In the full version, we prove that when costs are given in unary, the problem is Θ_2^P -hard. The proof is by a reduction from VC-compare, namely the problem of deciding, given two undirected graphs $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$, whether the size of a minimal vertex cover of G_1 is less than or equal to the size of a minimal vertex cover of G_2 . Essentially, given G_1 and G_2 , we construct an NFG N that subsumes both graphs and the objectives of the players are defined so that profiles correspond to choosing a vertex cover in one of the graphs. The edge e participates in profiles in which the players choose to proceed with a cover in G_1 , which happens only when the size of a minimal vertex cover of G_1 is less than or equal to the size of a minimal vertex cover of G_2 . Consequently, $\langle N, e \rangle \in \text{Edge-SO-affects}$ iff $\langle G_1, G_2 \rangle \in \text{VC-compare}$. ◀

We continue to the optimization problems. The proof is easy and can be found in the full version. In particular, the lower bounds are by a reduction from the SO-cost problem.

► **Theorem 9.** *The Edge-SO-optimization and SO-optimization problems are NP-complete.*

For the upper-bound of the SO-control problem, we first need the following lemma.

► **Lemma 10.** *Let N be an NFG and let $\kappa \geq 0$ be a threshold. If there are (not necessarily distinct) edges e_1 and e_2 of N such that $\text{cost}_{SO}(N[e_1 \leftarrow 0]) \geq \kappa$ and $\text{cost}_{SO}(N[e_2 \leftarrow \infty]) \leq \kappa$, then there is an edge e of N and a value $x \geq 0$ such that $\text{cost}_{SO}(N[e \leftarrow x]) = \kappa$.*

Proof. Assume towards contradiction that for all edges e of N and value $x \geq 0$, it holds that $\text{cost}_{SO}(N[e \leftarrow x]) \neq \kappa$. In particular, this means that $\text{cost}_{SO}(N[e_1 \leftarrow 0]) > \kappa$ and $\text{cost}_{SO}(N[e_2 \leftarrow \infty]) < \kappa$. Hence, by monotonicity of $\text{cost}_{SO}^c(N)$, we get that $\text{cost}_{SO}(N) = \text{cost}_{SO}(N[e_2 \leftarrow c(e_2)]) \leq \text{cost}_{SO}(N[e_2 \leftarrow \infty]) < \kappa < \text{cost}_{SO}(N[e_1 \leftarrow 0]) \leq \text{cost}_{SO}(N[e_1 \leftarrow c(e_1)]) = \text{cost}_{SO}(N)$. ◀

► **Theorem 11.** *The SO-control problem is DP-complete.*

Proof. We start with membership. Let $L_1 = \{\langle N, \kappa \rangle \mid \text{there exist an edge } e \text{ and } x \geq 0 \text{ such that } \text{cost}_{SO}(N[e \leftarrow x]) \leq \kappa\}$ and $L_2 = \{\langle N, \kappa \rangle \mid \text{there exist an edge } e \text{ and } x \geq 0 \text{ such that } \text{cost}_{SO}(N[e \leftarrow x]) \geq \kappa\}$. Note that L_1 is SO-optimization and is therefore in NP. We show that L_2 is in co-NP. The complement of L_2 is $L_2^c = \{\langle N, \kappa \rangle \mid \text{for all edges } e \text{ and } x \geq 0 \text{ we have } \text{cost}_{SO}(N[e \leftarrow x]) < \kappa\}$. A witness for membership in L_2^c is a set S of $|E| = m$ profiles, one for each edge, satisfying $\text{cost}(N[e \leftarrow \infty], P_e) < \kappa$ for each $P_e \in S$. The witness is polynomial since we only require m profiles. By monotonicity, it holds that if such a profile P_e exists for an edge e , then for every $x \geq 0$, we have that $\text{cost}_{SO}(N[e \leftarrow x]) \leq \text{cost}(N[e \leftarrow x], P_e) \leq \text{cost}(N[e \leftarrow \infty], P_e) < \kappa$. If this holds for every edge, then $\langle N, \kappa \rangle \in L_2^c$. In the other direction, if there is an edge e such that for every profile P it holds that $\text{cost}(N[e \leftarrow \infty], P) \geq \kappa$, then $\text{cost}_{SO}(N[e \leftarrow \infty]) \geq \kappa$, and therefore $\langle N, \kappa \rangle \notin L_2^c$. Therefore, L_2^c is in NP, hence L_2 is in co-NP. We show that $L_1 \cap L_2 = \text{SO-control}$.

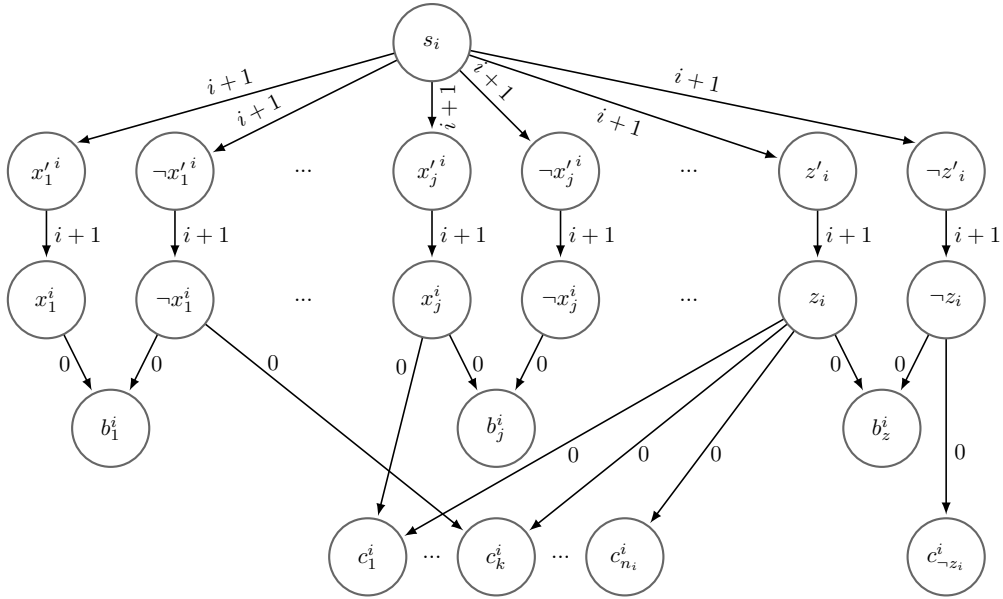
For the first direction, let $\langle N, \kappa \rangle \in \text{SO-control}$. Therefore, there is an edge $e \in E$ and a value $x \geq 0$ such that $\text{cost}_{SO}(N[e \leftarrow x]) = \kappa$. In particular, we have that $\text{cost}_{SO}(N[e \leftarrow x]) \leq \kappa$, therefore $\langle N, \kappa \rangle \in L_1$. Furthermore, $\text{cost}_{SO}(N[e \leftarrow x]) \geq \kappa$, therefore $\langle N, \kappa \rangle \in L_2$. Hence, $\langle N, \kappa \rangle \in L_1 \cap L_2$.

10:10 Coverage and Vacuity in Network Formation Games

For the other direction, let $\langle N, \kappa \rangle \in L_1 \cap L_2$. Since $\langle N, \kappa \rangle \in L_1$, there is $e_1 \in E$ and $x_1 \geq 0$ such that $\text{cost}_{SO}(N[e_1 \leftarrow x_1]) \leq \kappa$. If $\text{cost}_{SO}(N[e_1 \leftarrow \infty]) \geq \kappa$, then by continuity and the intermediate value theorem, there is $x \geq 0$ such that $\text{cost}_{SO}(N[e_1 \leftarrow x]) = \kappa$, hence $\langle N, \kappa \rangle \in \text{SO-control}$. If $\text{cost}_{SO}(N[e_1 \leftarrow \infty]) < \kappa$, we use the fact that $\langle N, \kappa \rangle \in L_2$. Hence, there is $e_2 \in E$ and $x_2 \geq 0$ such that $\text{cost}_{SO}(N[e_2 \leftarrow x_2]) \geq \kappa$. If $\text{cost}_{SO}(N[e_2 \leftarrow 0]) \leq \kappa$, then again by continuity and the intermediate value theorem, there is $x \geq 0$ such that $\text{cost}_{SO}(N[e_2 \leftarrow x]) = \kappa$. If $\text{cost}_{SO}(N[e_2 \leftarrow 0]) > \kappa$, then since $\text{cost}_{SO}(N[e_1 \leftarrow \infty]) < \kappa$ by Lemma 10, there is an edge $e \in E$ and a value $x \geq 0$ such that $\text{cost}_{SO}(N[e \leftarrow x]) = \kappa$, and therefore $\langle N, \kappa \rangle \in \text{SO-control}$.

We turn to prove that the problem is DP-hard. We reduce SAT-UNSAT to SO-control. SAT-UNSAT is the problem of deciding, given two 3CNF formulas φ_1 and φ_2 , whether φ_1 is satisfiable and φ_2 is not satisfiable. That is, $\langle \varphi_1, \varphi_2 \rangle \in \text{SAT-UNSAT}$ iff there exists an assignment f_1 to the variables of φ_1 such that f_1 satisfies φ_1 , and for all assignments f_2 to the variables of φ_2 , it holds that f_2 does not satisfy φ_2 . It was shown in [35] that SAT-UNSAT is DP-complete.

We propose the following reduction. For each formula φ_i , with $i \in \{1, 2\}$, we add a fresh variable z_i . We first construct a new formula φ'_i in the following way. For each clause, we disjunct the clause with z_i . We also conjunct the entire formula with $\neg z_i$. Note that if φ_i is satisfied by an assignment f_i , then φ'_i is satisfied by the assignment that agrees with f_i on all the variables in φ_i , and has $z_i = \mathbf{false}$. Furthermore, if φ_i is unsatisfiable, then φ'_i is unsatisfiable. Indeed, an assignment that satisfies φ'_i must have $z_i = \mathbf{false}$, implying that all other clauses are satisfied by an assignment that satisfies φ_i as well. Next, we construct an NFG $N_i = \langle k_i, V_i, E_i, c_i, \gamma_i \rangle$, for $i \in \{1, 2\}$, as follows (see Figure 2).



■ **Figure 2** The NFG N_i ; each edge denotes a set of two parallel edges with the same cost.

Let n_i be the number of variables in φ_i , and let m_i be the number of clauses in φ_i . Thus, the number of variables in φ'_i is $n_i + 1$, and the number of clauses in φ'_i is $m_i + 1$. We define $V_i = \bigcup_{1 \leq j \leq n_i+1} \{x_j^i, \neg x_j^i, x_j^{\prime i}, \neg x_j^{\prime i}, b_j^i\} \cup \bigcup_{1 \leq k \leq m_i+1} \{c_k^i\} \cup \{s_i\}$. That is, for each variable x_j^i of φ'_i , we have in V_i two vertices for the variable x_j^i , denoted $x_j^i, x_j^{\prime i}$, two vertices for its negation $\neg x_j^i$, denoted $\neg x_j^i, \neg x_j^{\prime i}$, and another vertex, denoted b_j^i . We also have a vertex for

each clause, and a source vertex. The edges and costs are as follows. There are two parallel edges, each with cost $i + 1$, from s_i to both $x_j^i, \neg x_j^i$ for every variable x_j^i of φ_i . There are two parallel edges, each with cost $i + 1$, from x_j^i to x_j^i and from $\neg x_j^i$ to $\neg x_j^i$ for every variable x_j^i of φ_i . There are two parallel edges, each with cost 0 from both $x_j^i, \neg x_j^i$ to b_j^i . Finally, for every clause c_k^i , there are two parallel edges, each with cost 0, from every literal appearing in c_k^i to the vertex c_k^i . Note that, in particular, this means that there are two parallel edges with cost 0 from z_i to all clauses except the clause $\neg z_i$. Finally, we have $k_i = n_i + 1 + m_i + 1$ players. The first $n_i + 1$ players are clause players, and the objective of Player $1 \leq k \leq n_i + 1$ is $\langle s_i, c_k^i \rangle$. The rest are variable players, and the objective of Player $n_i + 2 \leq j \leq n_i + m_i + 2$ is $\langle s_i, b_j^i \rangle$. To complete the construction, we fix $N = \langle k_1 + k_2, V_1 \cup V_2, E_1 \cup E_2, c_1 \cup c_2, \gamma_1 \cup \gamma_2 \rangle$ and $\kappa = 4n_1 + 6n_2 + 16$.

Note that since N_1 and N_2 are disjoint, it holds that $cost_{SO}(N) = cost_{SO}(N_1) + cost_{SO}(N_2)$. We argue that if φ_i , for $i \in [1, 2]$, is satisfiable, then $cost_{SO}(N_i) = 2(i + 1) \cdot (n_i + 1)$, and otherwise $cost_{SO}(N_i) = 2(i + 1) \cdot (n_i + 2)$. Thus, N has a distinct SO-cost to every combination of $\{\text{SAT}, \text{UNSAT}\} \times \{\text{SAT}, \text{UNSAT}\}$, which enables us to point to a threshold κ such that $\langle \varphi_1, \varphi_2 \rangle \in \text{SAT-UNSAT}$ iff $\langle N, \kappa \rangle \in \text{SO-control}$. Details can be found in the full version. ◀

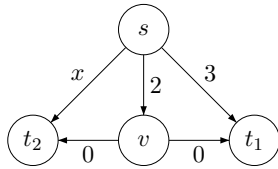
4 Affecting the Best Nash Equilibrium

In this section we study the sensitivity of the best NE to cost mutations. As we shall see, while the setting is less clean than in the SO case, we are able to obtain the same complexity bounds for analogous decision problems.

4.1 Monotonicity and continuity of the bNE

► **Theorem 12** ($cost_{bNE}$ is not monotone). *There is an NFG N and an edge e of N , such that the function $cost_{bNE}^e(N)$ is not monotone.*

Proof. Consider the NFG N appearing in Figure 3. The game is played between two players, with objectives $\langle s, t_1 \rangle$ and $\langle s, t_2 \rangle$. Let $e = \langle s, t_2 \rangle$. Table 5 describes the costs of the players in the possible four profiles of $N[e \leftarrow x]$. When $x \in [0, 1)$, the only NE is $\langle \pi_1^2, \pi_2^1 \rangle$, with cost $x + 2$. When $x > 1$, the only NE is $\langle \pi_1^1, \pi_2^2 \rangle$, with cost 2. So, for all $x \in (0, 1)$, we have that $cost_{bNE}(N[e \leftarrow x]) = 2 + x > 2 = cost_{bNE}(N[e \leftarrow 1])$, and thus $cost_{bNE}^e(N)$ is not monotone. ◀



■ **Figure 3** The NFG N .

■ **Table 5** Players' costs in N .

Player 2	π_2^1	π_2^2
Player 1	$s \rightarrow t_2$	$s \rightarrow v \rightarrow t_2$
π_1^1	x	2
$s \rightarrow t_1$	3	3
π_1^2	x	1
$s \rightarrow v \rightarrow t_1$	2	1

► **Theorem 13** ($cost_{bNE}$ is not continuous). *There is an NFG N and an edge e of N , such that the function $cost_{bNE}^e(N)$ is not continuous.*

10:12 Coverage and Vacuity in Network Formation Games

Proof. We use the same NFG N and edge e as in the proof of Theorem 12. It is easy to see that $cost_{bNE}^e(N)$ is not continuous at 1. ◀

While $cost_{bNE}$ is neither monotonous nor continuous, we now show that it is composed of finitely many linear segments. We say that a function $f : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ is *composed of linear segments* if there is a segmentation $0 = x_0 < x_1 < \dots < x_n < x_{n+1} = \infty$ of \mathbb{R}^+ , for some $n \geq 0$, such that for every $0 \leq i \leq n$ there is a linear function $f_i : \mathbb{R} \rightarrow \mathbb{R}$ such that for all $x \in [x_i, x_{i+1}]$ it holds that $f(x) = f_i(x)$. We call x_0, x_1, \dots, x_{n+1} the *edge points* of f . Given an NFG N , a profile P , and an edge e , the cost of P is a linear function with respect to the cost of e . Indeed, $cost(N, P) = \sum_{e' \in P \setminus \{e\}} c(e') + \mathbb{1}_{P,e} c(e)$, where $\mathbb{1}_{P,e} \in \{0, 1\}$ is an indicator of e being used in P . In particular, when $\mathbb{1}_{P,e} = 0$, then $cost(N, P)$ is a constant function.

► **Lemma 14.** *Given an NFG N , an edge e , and a profile P , the range of values x such that P is an NE in $N[e \leftarrow x]$ is a single (possibly empty) segment.*

Proof. By definition, a profile P is an NE if for every i and for every profile P' obtained from P by a deviation π'_i of Player i that $cost_{N,P}(\pi_i) \leq cost_{N,P'}(\pi'_i)$. Hence, P is an NE in $N[e \leftarrow x]$ in values x for which the set of constraints of the form $cost_{N,P}(\pi_i) \leq cost_{N,P'}(\pi'_i)$ holds. As each constraint is a linear inequality in a single variable (that is, x), the solution set is a single (perhaps empty) segment. ◀

We denote by $bumps(P)$ the set of edge points of the segment along which P is an NE in $N[e \leftarrow x]$. That is, $bumps(P) = \{a, b\}$ if P is an NE in $N[e \leftarrow x]$ for exactly all $a \leq x \leq b$. By Lemma 14, $bumps(P)$ contains at most two points. We further denote by $Bumps(N, e) = \bigcup_P bumps(P)$. Since the number of strategies per player and the number of players are finite, the number of profiles is finite as well. Hence, since $|bumps(P)| \leq 2$ for every profile P , we get that $Bumps(N, e)$ is finite.

Consider two profiles $P_1 \neq P_2$ in N . For an edge e , we say that a value $x \geq 0$ is an *intersection point* for e , P_1 , and P_2 , if $cost(N[e \leftarrow x], P_1) = cost(N[e \leftarrow x], P_2)$. Note that since $cost(N[e \leftarrow x], P)$ is linear for every profile P , there is at most one intersection point for every edge and two profiles. Let $Ints(N, e)$ be the set of all intersection points for e and pairs of profiles in N . Since the number of different profiles is finite, so is $Ints(N, e)$.

► **Theorem 15.** *Consider an NFG N and an edge e in N . Then, $cost_{bNE}(N[e \leftarrow x])$ is composed of finitely many linear segments, and is monotonically increasing within each segment.*

Proof. Recall that $cost_{bNE}^e(N)(x) = cost_{bNE}(N[e \leftarrow x]) = \min_{P \in bNE(N[e \leftarrow x])} cost(N[e \leftarrow x], P) = \min_{P \in bNE(N[e \leftarrow x])} \sum_{e' \in P \setminus \{e\}} c(e') + \mathbb{1}_{P,e} x$. Hence, $cost_{bNE}(N[e \leftarrow x])$ is composed of linear segments. The set of edge points refines $bumps(N, e) \cup Ints(N, e)$, and since it is finite, so are the number of segments. Furthermore, as $cost(N[e \leftarrow x], P)$ is monotonically increasing for every P , we get that $cost_{bNE}(N[e \leftarrow x])$ is monotonically increasing within each segment. ◀

Figure 4 below contains plots² of the function $cost_{bNE}(N[e \leftarrow x])$. The left plot describes $cost_{bNE}(N[e \leftarrow x])$ where N is the NFG from Example 1 and $e = \langle s, u \rangle$. To its right, we describe a three-player NFG N and the plot of $cost_{bNE}(N[e \leftarrow x])$ with $e = \langle s, v_2 \rangle$.

² The plots were generated by a simple Python program that gets as input an NFG by means of a NetworkX weighted directed graph, and naively follows the segmentation from Theorem 15.

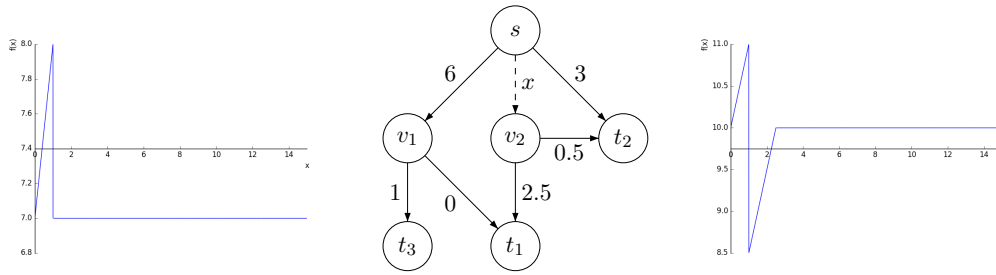


Figure 4 Plots for $cost_{bNE}(N[e \leftarrow x])$.

4.2 Decision problems

The bNE-cost decision problem is the problem of deciding, given an NFG N and a threshold $\kappa \geq 0$, whether $cost_{bNE}(N) \leq \kappa$. The bNE-cost problem is NP-complete [4]. In this section we study the following related decision problems.

1. **Edge-bNE-affects:** Given an NFG N and an edge e of N , does e bNE-affect N ? Thus, $Edge\text{-}bNE\text{-}affects = \{\langle N, e \rangle \mid e \text{ bNE-affects } N\}$.
2. **Edge-bNE-optimization:** Given an NFG N , an edge e of N , and a threshold $\kappa \geq 0$, is there a value $x \geq 0$, such that $cost_{bNE}(N[e \leftarrow x]) \leq \kappa$? Thus, $Edge\text{-}bNE\text{-}optimization = \{\langle N, e, \kappa \rangle \mid \text{there exists } x \geq 0 \text{ such that } cost_{bNE}(N[e \leftarrow x]) \leq \kappa\}$.
3. **bNE-optimization:** Given an NFG N and a threshold $\kappa \geq 0$, is there an edge e of N and a value $x \geq 0$, such that $cost_{bNE}(N[e \leftarrow x]) \leq \kappa$? Thus, $bNE\text{-}optimization = \{\langle N, \kappa \rangle \mid \text{there exist } e \text{ and } x \geq 0 \text{ such that } cost_{bNE}(N[e \leftarrow x]) \leq \kappa\}$.

Before we turn to analyze the complexity of the problems, let us illustrate the non-intuitive behavior of $cost_{bNE}$. Consider the NFG N appearing in Figure 5, and let $e = \langle s, v_2 \rangle$. As can be seen in Table 6, the profile $\langle \pi_1^3, \pi_2^3 \rangle$ is an NE with cost 10 independent of the value of x . Then, when $0 \leq x \leq \frac{1}{2}$, the profile $\langle \pi_1^2, \pi_2^2 \rangle$ is an NE with cost $10.5 + x$, and when $x \geq \frac{1}{2}$, the profile $\langle \pi_1^1, \pi_2^1 \rangle$ is an NE with cost 9. Accordingly, $cost_{bNE}(N[e \leftarrow x])$ is 10 when $0 \leq x < \frac{1}{2}$, and is 9 when $x \geq \frac{1}{2}$. Though observations of the non-intuitive behavior of network exists in literature (e.g., Braess' Paradox [12]), it is common that added/removed edges participate in equilibria profiles either before or after changing the network. In this example, however, the edge e , which bNE-affects N , does not participate in any bNE profile! Thus, $cost_{bNE}$ is fixed in the two segments $[0, \frac{1}{2})$ and $[\frac{1}{2}, \infty]$, yet still e bNE affects N .

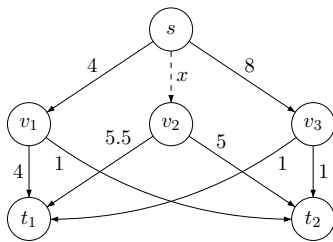


Figure 5 The NFG N .

Table 6 Players' costs in N .

Player 2	π_2^1	π_2^2	π_2^3
Player 1	s, v_1, t_2	s, v_2, t_2	s, v_3, t_2
π_1^1	3	$5 + x$	9
s, v_1, t_1	6	8	8
π_1^2	5	$5 + \frac{x}{2}$	9
s, v_2, t_1	$5.5 + x$	$5.5 + \frac{x}{2}$	$5.5 + x$
π_1^3	5	$5 + x$	5
s, v_3, t_1	9	9	5

► **Lemma 16.** Let N be an NFG, and let e be an edge in N . If there is an NE profile P such that $e \notin P$, then for all $x \geq c(e)$, we have that P is an NE in $N[e \leftarrow x]$.

Proof. Assume towards contradiction that there is $x > c(e)$ such that P is not an NE. Then, there is a player i with strategy π_i in P that has an incentive to unilaterally deviate to another strategy π'_i . Denote by P' the deviation profile resulting from i 's deviation. Since P is an NE in N , we have that $\text{cost}_{N,P}(\pi_i) \leq \text{cost}_{N,P'}(\pi'_i)$. Since $e \notin P$, we have that $\text{cost}_{N[e \leftarrow x],P}(\pi_i) = \text{cost}_{N,P}(\pi_i)$. Since $x > c(e)$ we have that $\text{cost}_{N,P'}(\pi'_i) \leq \text{cost}_{N[e \leftarrow x],P'}(\pi'_i)$. Therefore $\text{cost}_{N[e \leftarrow x],P}(\pi_i) \leq \text{cost}_{N[e \leftarrow x],P'}(\pi'_i)$, in contradiction to the fact that Player i has an incentive to deviate. \blacktriangleleft

Lemma 16, together with the segmentation of $bNE(N[e \leftarrow x])$, is used for proving the following characterization of an edge that does not bNE-affect N . The proof is based on a careful consideration of all cases and can be found in the full version.

► **Theorem 17.** *Let N be an NFG. An edge e in N does not bNE-affect N iff there is a profile $P \in bNE(N[e \leftarrow 0])$ such that $e \notin P$ and for all $x \geq 0$ it holds that $\text{cost}_{bNE}(N[e \leftarrow x]) \geq \text{cost}_{bNE}(N[e \leftarrow 0])$.*

► **Theorem 18.** *The Edge-bNE-affects problem is Δ_2^P -complete, and is Θ_2^P -complete when costs are given in unary.*

Proof. We start with membership. First, note that given an NFG N , and edge e of N , and a value $\kappa \geq 0$, we can decide in NP whether there is a profile P such that $e \notin P$ and $\text{cost}(N, P) = \kappa$.

Let $OPT_0 = \text{cost}_{bNE}(N[e \leftarrow 0])$. As argued in the membership claim for Theorem 8, we can find OPT_0 using polynomially-many queries to an NP oracle when costs are given in binary, and using logarithmically-many queries when costs are given in unary. Then, using a single query to Edge-bNE-optimization (with modification to strictly smaller) with input N, e , and OPT_0 , we can decide if there is a value $x \geq 0$ such that $\text{cost}_{bNE}(N[e \leftarrow x]) < OPT_0$. If so, then e affects N . Otherwise, use a single query to ask if there is a profile P such that $e \notin P$ and $\text{cost}(N[e \leftarrow 0], P) = OPT_0$. By Theorem 17, we have that e bNE-affects N iff the answer is no.

The hardness results for Δ_2^P and Θ_2^P can be found in the full version. In both cases we use the same reduction as in the hardness results for Theorem 8. In the case of Δ_2^P we make a slight variation. Then we show that the profiles described for the SO is a superset of the bNE profiles. \blacktriangleleft

Finally, for the optimization problems, the analysis is similar to the one in Theorem 9, except that we also have to argue that the witness value x is polynomial in input. The details can be found in the full version.

► **Theorem 19.** *The edge-bNE-optimization and bNE-optimization problems are NP-complete.*

► **Remark 20 (On the PoS and the worst NE).** Recall that $PoS(N) = \frac{\text{cost}_{bNE}(N)}{\text{cost}_{SO}(N)}$. If an edge e bNE-affects N , it does not necessarily imply that e PoS-affects N . Indeed, e may participate also in the SO. Nevertheless, the NFG N used in the proofs of Theorems 12 and 13 demonstrates that PoS is neither monotone nor continuous. To see this, note that for all $x \geq 0$, we have that $\text{cost}_{SO}(N[e \leftarrow x]) = 2$, we get that for $x \in [0, 1)$, we have that $PoS(N[e \leftarrow x]) = 1 + \frac{x}{2}$, and for $x \geq 1$, we have that $PoS(N[e \leftarrow x]) = 1$.

As for the worst NE, since the NFG N used in the proofs of Theorems 12 and 13 is such that $N[e \leftarrow x]$ has a single NE for all values of x , the considerations about the best and worst NE coincide, and thus N demonstrate that cost_{wNE} is neither monotone nor continuous.

5 Discussion and Future Work

We studied the effect of mutations applied to the cost of edges in network formation games. Our results about monotonicity and continuity of the SO and NE are aligned with similar folk results in similar settings in game theory. We are, however, the first to introduce a formal framework to study these phenomena, and to provide a complexity analysis of the decision problems they induce. We also point to new surprising effects of the mutations.

The mutations we study for NFGs are of a restricted type: an unbounded change in the cost of a single resource in the game. As has been the case in coverage and vacuity in formal verification, richer types of mutations reflect practical bounds on the possible mutations. For example, it would be interesting to study how one can control the bNE by a budget-restricted mutation of several edges. Also, while our definition of affect is Boolean, namely an edge SO-, bNE-, or wNE-affects a network or it does not, it is interesting to examine a quantitative approach, where we care how much an edge affects these measures. Finally, while our optimization problems care about an upper bound to the costs of the SO and bNE, in some applications it is interesting to control these values by both an upper and lower bound. We leave the richer setting and variants for future research.

Both game theory and formal verification aim at reasoning about behaviors of interacting entities, yet consider different aspects of the interaction. We view this work as another chain in an exciting transfer of concepts and ideas between the two areas [28]. In the context of game theory, this includes an extension of NFGs to objectives that are richer than reachability [9], to a timed setting [6], and to a setting where the strategies of the players are dynamic [7]. Beyond richer settings, it is shown in [30, 5] how ideas used in formal verification for abstraction and symbolic presentation of huge systems can be used for reasoning about NFGs. In the other direction, concepts from game theory are used in the formalization of strategic behaviors in formal verification (e.g., rational verification and synthesis [22, 38]). In the more economic view, cost-sharing mechanisms from NFGs are used in [8] in order to augment the problem of synthesis from component libraries by cost considerations.

Our contribution here started with the transfer of concepts from formal verification to game theory, yet our results suggest new research directions in coverage and vacuity in formal verification, and logic in general. Studies of coverage and vacuity so far concern Boolean specification formalisms [27]. In contrast, the objectives of the players in typical game-theoretic settings, in particular NFGs, are quantitative. Recently, there is growing interest in *multi-valued* specification formalisms, which specify the *quality* of systems, and not only their correctness [2]. Moreover, the systems we reason about may be multi-valued too. For the multi-valued setting, we need to develop a theory of quantified multi-valued propositions. In particular, the segmentation of values in \mathbb{R}^+ we perform for bNE, is analogous to a segmentation of $[0, 1]$ – the domain of values of atomic propositions and sub-formulas in typical multi-valued formalisms. Indeed, while mutations of sub-formulas that appear in a positive or negative polarity behave monotonically, sub-formulas with a mixed polarity may induce a non-trivial segmentation. Moreover, as has been the case with $\text{bumps}(P)$ in the bNE segmentation, the edge points of the segments may not be constants that appear in the formula. For example, when sub-formulas and atomic propositions take values in $[0, 1]$, then the maximal satisfaction value of the formula $p \wedge (\neg p)$ is when the satisfaction value of p is $\frac{1}{2}$.

Furthermore, the need to reason formally about multi-agent systems has led to a development of specification formalisms that enable reasoning about on-going strategic behaviors [3, 13, 32, 11]. Essentially, these formalisms, most notably ATL, ATL*, and Strategy Logic (SL), include quantification of strategies of the different agents and of the computations they may force the system into, making it possible to specify concepts like SO and NE.

While coverage and vacuity are traditionally viewed as sanity checks in model checking, in the context of SL specifications, they can also be used for revealing properties of games and strategic behaviors. Our work demonstrates how SL formulas that specify concepts like SO and NE explain properties like monotonicity. Indeed, non-monotonicity of the bNE corresponds to the mixed polarity of the objectives in the SL formula that describes an NE: a negative occurrence (left-hand side of an implication) when we refer to a deviation and a positive one (right-hand side of that implication) in for the current strategy. In contrast, in the formula for the SO, all occurrences of the objectives are positive, implying monotonicity. Moreover, for a specific given game, reasoning about the effect of mutations can be reduced to checking the coverage of SL formulas that specify properties of the game. Thus, a framework for coverage and vacuity in SL is interesting for both formal verification and game theory.

References

- 1 S. Albers, S. Elits, E. Even-Dar, Y. Mansour, and L. Roditty. On Nash Equilibria for a Network Creation Game. In *7th ACM-SIAM Symposium on Discrete Algorithms*, 2006.
- 2 S. Almagor, U. Boker, and O. Kupferman. Formalizing and Reasoning About Quality. *Journal of the ACM*, 63(3), 2016.
- 3 R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713, 2002.
- 4 E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The Price of Stability for Network Design with Fair Cost Allocation. *SIAM J. Comput.*, 38(4):1602–1623, 2008.
- 5 G. Avni, S. Guha, and O. Kupferman. An Abstraction-Refinement Methodology for Reasoning about Network Games. In *Proc. 26th Int. Joint Conf. on Artificial Intelligence*, pages 70–76, 2017.
- 6 G. Avni, S. Guha, and O. Kupferman. Timed Network Games. In *42nd Int. Symp. on Mathematical Foundations of Computer Science*, volume 83 of *LIPICs*, pages 37:1–37:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2017.
- 7 G. Avni, T.A. Henzinger, and O. Kupferman. Dynamic Resource Allocation Games. In *Algorithmic Game Theory - 9th International Symposium*, volume 9928 of *Lecture Notes in Computer Science*, pages 153–166. Springer, 2016.
- 8 G. Avni and O. Kupferman. Synthesis from Component Libraries with Costs. In *Proc. 25th Int. Conf. on Concurrency Theory*, volume 8704 of *Lecture Notes in Computer Science*, pages 156–172. Springer, 2014.
- 9 G. Avni, O. Kupferman, and T. Tamir. Network-formation games with regular objectives. *Information and Computation*, 251:165–178, 2016.
- 10 I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in ACTL formulas. *Formal Methods in System Design*, 18(2):141–162, 2001.
- 11 P. Bouyer-Decitre, O. Kupferman, N. Markey, B. Maubert, A. Murano, and G. Perelli. Reasoning about Quality and Fuzziness of Strategic Behaviours. In *Proc. 28th Int. Joint Conf. on Artificial Intelligence*, 2019.
- 12 D. Braess. Über ein paradoxon aus der verkehrsplanung. *Unternehmensforschung*, 12(1):258–268, 1968.
- 13 K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. *Information and Computation*, 208(6):677–693, 2010.
- 14 H. Chockler, A. Gurfinkel, and O. Strichman. Beyond vacuity: towards the strongest passing formula. *Formal Methods in System Design*, 43(3):552–571, 2013.
- 15 H. Chockler, O. Kupferman, R.P. Kurshan, and M.Y. Vardi. A Practical Approach to Coverage in Model Checking. In *Proc. 13th Int. Conf. on Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 66–78. Springer, 2001.

- 16 H. Chockler, O. Kupferman, and M.Y. Vardi. Coverage Metrics for Temporal Logic Model Checking. In *Proc. 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, number 2031 in Lecture Notes in Computer Science, pages 528–542. Springer, 2001.
- 17 H. Chockler, O. Kupferman, and M.Y. Vardi. Coverage Metrics for Formal Verification. *Software Tools for Technology Transfer*, 8(4-5):373–386, 2006.
- 18 E.M. Clarke, O. Grumberg, D. Kroening, D. Peled, and H. Veith. *Model Checking, Second Edition*. MIT Press, 2018.
- 19 R. Cole, Y. Dodis, and T. Roughgarden. How much can taxes help selfish routing? *J. Comput. Syst. Sci.*, 72(3):444–467, 2006.
- 20 J. Correa, A. Schulz, and N. Stier Moses. Selfish Routing in Capacitated Networks. *Math. Oper. Res.*, 29(4):961–976, 2004.
- 21 A. Fabrikant, A. Luthra, E. Maneva, C. Papadimitriou, and S. Shenker. On a Network Creation Game. In *ACM Symposium on Principles of Distributed Computing*, 2003.
- 22 D. Fisman, O. Kupferman, and Y. Lustig. Rational Synthesis. In *Proc. 16th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010.
- 23 J.D. Hartline and T. Roughgarden. Optimal mechanism design and money burning. In *Proc. 40th ACM Symp. on Theory of Computing*, pages 75–84, 2008.
- 24 Y. Hoskote, T. Kam, P.-H Ho, and X. Zhao. Coverage estimation for symbolic model checking. In *Proc. 36th Design Automation Conf.*, pages 300–305, 1999.
- 25 S. Katz, D. Geist, and O. Grumberg. “Have I written enough properties ?” A method of comparison between specification and implementation. In *Proc. 10th Conf. on Correct Hardware Design and Verification Methods*, volume 1703 of *Lecture Notes in Computer Science*, pages 280–297. Springer, 1999.
- 26 M.W. Krentel. The complexity of optimization problems. *Journal of Computer and Systems Science*, 36:490–509, 1988.
- 27 O. Kupferman. Sanity Checks in Formal Verification. In *Proc. 17th Int. Conf. on Concurrency Theory*, volume 4137 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2006.
- 28 O. Kupferman. Examining classical graph-theory problems from the viewpoint of formal-verification methods. In *Proc. 49th ACM Symp. on Theory of Computing*, page 6, 2017.
- 29 O. Kupferman, W. Li, and S.A. Seshia. A Theory of Mutations with Applications to Vacuity, Coverage, and Fault Tolerance. In *Proc. 8th Int. Conf. on Formal Methods in Computer-Aided Design*, pages 1–9, 2008.
- 30 O. Kupferman and T. Tamir. Hierarchical Network Formation Games. In *Proc. 22nd Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 10205 of *Lecture Notes in Computer Science*, pages 229–246. Springer, 2017.
- 31 O. Kupferman and M.Y. Vardi. Vacuity detection in temporal model checking. *Software Tools for Technology Transfer*, 4(2):224–233, 2003.
- 32 F. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi. Reasoning About Strategies: On the Model-Checking Problem. *ACM Transactions on Computational Logic*, 15(4):34:1–34:47, 2014.
- 33 O. Padon, N. Immerman, A. Karbyshev, O. Lahav, M. Sagiv, and S. Shoham. Decentralizing SDN Policies. In *Proc. 42nd ACM Symp. on Principles of Programming Languages*, pages 663–676, 2015.
- 34 A. Panda, K. Argyraki, M. Sagiv, M. Schapira, and S. Shenker. New Directions for Network Verification. In *1st Summit on Advances in Programming Languages*, volume 32 of *LIPICs*, pages 209–220, 2015.
- 35 C.H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). In *Proc. 14th ACM Symp. on Theory of Computing*, pages 255–260, 1982.
- 36 F. Souza and L.C. Rego. Mixed equilibrium in 2×2 normal form games: when burning money is rational. *Pesquisa Operacional*, 36:81–99, April 2016.

10:18 Coverage and Vacuity in Network Formation Games

- 37 E. Tardos and T. Wexler. *Algorithmic Game Theory*. Cambridge University Press, 2007. Chapter 19: Network Formation Games and the Potential Function Method.
- 38 M. Wooldridge, J. Gutierrez, P. Harrenstein, E. Marchioni, G. Perelli, and A. Toumi. Rational Verification: From Model Checking to Equilibrium Checking. In *Proc. of 30th National Conf. on Artificial Intelligence*, pages 4184–4190, 2016.

A Complete Axiomatisation of a Fragment of Language Algebra

Paul Brunet 

University College London, United Kingdom

<http://paul.brunet-zamansky.fr>

paul@brunet-zamansky.fr

Abstract

We consider algebras of languages over the signature of reversible Kleene lattices, that is the regular operations (empty and unit languages, union, concatenation and Kleene star) together with intersection and mirror image. We provide a complete set of axioms for the equational theory of these algebras. This proof was developed in the proof assistant Coq.

2012 ACM Subject Classification Theory of computation → Algebraic language theory

Keywords and phrases Kleene algebra, language algebra, completeness theorem, axiomatisation

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.11

Supplement Material Coq formalisation: <https://github.com/monstrencage/LangAlg>

Funding This work was funded by the EPSRC grant IRIS (reference EP/R006865/1).

Acknowledgements I want to thank the anonymous referees who provided valuable comments, and Amina Doumane for her kind assistance.

1 Introduction

We are interested in algebras of languages, equipped with the constants empty language (0), unit language (1, the language containing only the empty word), the binary operations of union (+), intersection (\cap), and concatenation (\cdot), and the unary operations of Kleene star ($(-)^*$) and mirror image ($\overline{(-)}$). It is convenient in this paper to see the Kleene star as a derived operator $e^* := 1 + e^+$ with the operator e^+ representing the non-zero iteration. We call these algebras *reversible Kleene lattices*. Given a finite set of variables X , and two terms e, f built from variables and the above operations, we say that the equation $e \simeq f$ is *valid* if the corresponding equality holds universally.

In a previous paper [3] we have presented an algorithm to test the validity of such equations, and shown this problem to be EXSPACE-complete. However, we had left open the question of the axiomatisation of these algebras. We address it now, by providing in the current paper a set of axioms from which every valid equation can be derived.

Several fragments of this algebra have been studied:

Kleene algebra (KA): if we restrict ourselves to the operators of regular expressions (0, 1, +, \cdot , and $(-)^+$), then several axiomatisations have been proposed by Conway[4], before being shown to be complete by Krob [8] and Kozen [6].

Kleene algebra with converse: if we add to KA the mirror operation, then the previous theorem can be extended by switching to a duplicated alphabet, with a letter a' denoting the mirror of the letter a . A small number of identities may be added to KA to get a complete axiomatisation [2].

Identity-free Kleene lattices: this algebra stems from the operators 0, +, \cdot , \cap and $(-)^+$. In a recent paper [5] Doumane and Pous provided a complete axiomatisation of this algebra.



© Paul Brunet;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 11; pp. 11:1–11:15

Leibniz International Proceedings in Informatics



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

The present work is then an extension of identity-free Kleene lattices, by adding unit and mirror image. We provide in Table 1 a set of axioms which we prove to be complete for the equational theory of language algebra, by reducing to the completeness theorem of [5]. This proof has been formalised in Coq.

The paper is organised as follows. In Section 2, we introduce some notations and define the various types of expressions used in the paper. We present our axioms and state our main theorem. In Section 3 we deal with a technical lemma having to do with the treatment of the empty word. We proceed in Section 4 to extend the theorem of [5] with the mirror image operator. Section 5 studies in detail terms of the algebra that are below the constant 1, as those play a crucial role in the main proof. We present the proof of our main result in Section 6. We conclude in Section 7 by a discussion on an operator that is missing from our signature, namely constant \top denoting the full language.

On the Coq formalisation

As we have mentioned already, the proofs in this paper have been formalised and checked using the proof assistant Coq. This has several consequences for the present article.

Since Coq offers a very high level of confidence in the proofs it validates, the summary we give here is not meant to convince the reader of our result's validity. Instead we focus on the precise statement of the theorems we proved, and the strategy we employed to establish those. If the reader has doubts as to the validity of some of our claims we refer them to the Coq proof, available on GitHub.

The source of most mistakes when dealing with formal proofs is the correspondence between the statement we want to prove and the one we actually prove. In other words, the main task when assessing the validity of a Coq proof consists in checking that the definitions and assertions in the Coq file match those we have in mind. To that effect, we tried in the present document to remain as close as possible to the Coq script. This might sometimes lead to slightly pedantic definitions, and less than intuitive proofs. We feel however that this is better than the alternative: we use the claims we checked instead of making more intuitive but imprecise arguments.

2 Preliminaries

2.1 Sets, words, and languages

Given a set X , we write $\mathcal{P}(X)$ for the powerset of X and $\mathcal{P}_f(X)$ for the set of finite subsets of X . We will denote the two-elements boolean set as 2. For two sets X, Y , we write $X \times Y$ for their Cartesian product, $X \cup Y$ for their union, and $X \cap Y$ for their intersection. The empty set is denoted by \emptyset . We will use the notation $f(A)$ for a set $A \subseteq X$ and a function $f : X \rightarrow Y$ to represent the set $\{y \in Y \mid \exists a \in A : f(a) = y\} = \{f(a) \mid a \in A\}$.

Let Σ be an arbitrary alphabet (set), the words over Σ are finite sequences of elements from Σ . The set of all words is written Σ^* , and the empty word is written ε . The concatenation of two words u, v is simply denoted by uv . The mirror image of a word u , obtained by reading it backwards, is written \bar{u} . For instance \overline{abc} is the word cba .

A language is a set of words, that is an element of $\mathcal{L}(\Sigma) := \mathcal{P}(\Sigma^*)$. We will also use the symbol ε to denote the unit language $\{\varepsilon\}$. The concatenation of two languages L and M , denoted by $L \cdot M$, is obtained by lifting pairwise the concatenation of words: it contains exactly those words that can be obtained as a concatenation uv where $\langle u, v \rangle \in L \times M$. Similarly the mirror image of a language L , denoted by \bar{L} , is the set of mirror images of words

■ **Table 1** Axioms of reversible Kleene lattices.

$e + f = f + e$	(1a.1)	$e \cdot (f \cdot g) = (e \cdot f) \cdot g$	(1b.1)
$e + (f + g) = (e + f) + g$	(1a.2)	$e \cdot 0 = 0 = 0 \cdot e$	(1b.2)
$e + 0 = e$	(1a.3)	$(e + f) \cdot g = e \cdot g + f \cdot g$	(1b.3)
$e \cap f = f \cap e$	(1a.4)	$e \cdot (f + g) = e \cdot f + e \cdot g$	(1b.4)
$e \cap e = e$	(1a.5)	$e^+ = e + e \cdot e^+$	(1b.5)
$e \cap (f \cap g) = (e \cap f) \cap g$	(1a.6)	$e^+ = e + e^+ \cdot e$	(1b.6)
$(e + f) \cap g = e \cap g + f \cap g$	(1a.7)	$e \cdot f + f = f \Rightarrow e^+ \cdot f + f = f$	(1b.7)
$(e \cap f) + e = e$	(1a.8)	$f \cdot e + f = f \Rightarrow f \cdot e^+ + f = f$	(1b.8)
(a) Distributive lattice.		(b) Concatenation and iteration.	
$\bar{\bar{e}} = e$	(1c.1)	$1 \cdot e = e = e \cdot 1$	(1d.1)
$\overline{e + f} = \bar{e} + \bar{f}$	(1c.2)	$1 \cap (e \cdot f) = 1 \cap (e \cap f)$	(1d.2)
$\overline{e \cdot f} = \bar{e} \cdot \bar{f}$	(1c.3)	$1 \cap \bar{e} = 1 \cap e$	(1d.3)
$\overline{e \cap f} = \bar{e} \cap \bar{f}$	(1c.4)	$(1 \cap e) \cdot f = f \cdot (1 \cap e)$	(1d.4)
$\overline{e^+} = \bar{e}^+$	(1c.5)	$((1 \cap e) \cdot f) \cap g = (1 \cap e) \cdot (f \cap g)$	(1d.5)
		$(g + (1 \cap e) \cdot f)^+ = g^+ + (1 \cap e) \cdot (g + f)^+$	(1d.6)
(c) Mirror image.		(d) Unit.	

from L . We write L^n when $L \in \mathcal{L} \langle \Sigma \rangle$ and $n \in \mathbb{N}$ for the iterated concatenation, defined by induction on n by $L^0 := \varepsilon$ and $L^{n+1} := L \cdot L^n$. The language L^+ is the union of all non-zero iterations of L , i.e. $L^+ := \bigcup_{n>0} L^n$.

2.2 Terms: syntax and semantics

Throughout this paper, we will consider expressions over various signatures which we list here. We fix a set of variables X , and let x, y, \dots range over X .

Expressions: $e, f \in \mathbb{E}_X ::= x \mid 0 \mid 1 \mid e + f \mid e \cdot f \mid e \cap f \mid e^+ \mid \bar{e}$;

One-free expressions: $e, f \in \mathbb{E}'_X ::= x \mid 0 \mid e + f \mid e \cdot f \mid e \cap f \mid e^+ \mid \bar{e}$;

Simple expressions: $e, f \in \mathbb{E}^-_X ::= x \mid 0 \mid e + f \mid e \cdot f \mid e \cap f \mid e^+$;

We will use various sets of axioms, depending on the signature. All of the axioms under consideration are listed in Table 1. We use these axioms to generate equivalence relations over terms. For a type of expressions $\mathbb{T}_X \in \{\mathbb{E}_X, \mathbb{E}'_X, \mathbb{E}^-_X\}$, the *axiomatic equivalence relation*, written \equiv is the smallest congruence on \mathbb{T}_X containing those axioms in Table 1 that only use symbols from the signature of \mathbb{T}_X . This means that for \mathbb{E}^-_X we use the axioms from Tables 1a and 1b, for \mathbb{E}'_X we add those from Table 1c and for \mathbb{E}_X we keep all of the axioms of Table 1. We will use the shorthand $e \leq f$ to mean $e + f \equiv f$. This ensures that \leq is a partial order with respect to \equiv . We list in Table 2 some statements that are provable from the axioms.

► **Remark 1.** Axioms in Tables 1a and 1b are borrowed from [5]. We actually omit two axioms from Pous & Doumane: their axiomatisation include (2a.1) and (2a.3), which happen

11:4 A Complete Axiomatisation of a Fragment of Language Algebra

■ **Table 2** Some consequences of the axioms.

$e + e \equiv e$	(2a.1)	$e^+ \cdot e^+ \leq e^+$	(2b.1)
$e \cap 0 \equiv 0$	(2a.2)	$(e^+)^+ \equiv e^+$	(2b.2)
$e \cap (e + f) \equiv e$	(2a.3)	$(1 + e)^+ \equiv 1 + e^+$	(2b.3)
(a) Lattice laws.		(b) Iteration.	
$\bar{0} \equiv 0$	(2c.1)	$e \leq g \Rightarrow f \leq g \Rightarrow e + f \leq g$	(2d.1)
$\bar{1} \equiv 1$	(2c.2)	$g \leq e \Rightarrow g \leq f \Rightarrow g \leq e \cap f$	(2d.2)
$0^+ \equiv 0$	(2c.3)	$e \leq f \Leftrightarrow e \cap f \equiv e$	(2d.3)
$1^+ \equiv 1$	(2c.4)	$1 \leq e \cdot f \Leftrightarrow 1 \leq e \wedge 1 \leq f$	(2d.4)
(c) Constants.		(d) Reasoning rules.	

to be derivable from the other identities. The mirror image identities, presented in Table 1c come from [2] (except (1c.4), which is a trivial extension). In Table 1d, we find (1d.1) which is a standard monoid law, as well as (1d.4) and (1d.5) from [1]. Axiom (1d.6) was also present in that paper, although using the Kleene star instead of the non-zero iteration. As far as we know the identities (1d.2) and (1d.3) are new.

Given an expression $e \in \mathbb{T}_X$, a set Σ , and a map $\sigma : X \rightarrow \mathcal{L}(\Sigma)$, we may interpret e as a language over Σ using the following inductive definition:

$$\begin{array}{lll}
 \llbracket x \rrbracket_\sigma := \sigma(x) & \llbracket e + f \rrbracket_\sigma := \llbracket e \rrbracket_\sigma \cup \llbracket f \rrbracket_\sigma & \llbracket e^+ \rrbracket_\sigma := \llbracket e \rrbracket_\sigma^+ \\
 \llbracket 0 \rrbracket_\sigma := \emptyset & \llbracket e \cdot f \rrbracket_\sigma := \llbracket e \rrbracket_\sigma \cdot \llbracket f \rrbracket_\sigma & \llbracket \bar{e} \rrbracket_\sigma := \overline{\llbracket e \rrbracket_\sigma} \\
 \llbracket 1 \rrbracket_\sigma := \varepsilon & \llbracket e \cap f \rrbracket_\sigma := \llbracket e \rrbracket_\sigma \cap \llbracket f \rrbracket_\sigma &
 \end{array}$$

The *semantic equivalence* and *semantic containment* relations on \mathbb{T}_X , respectively written \simeq and \lesssim , are defined as follows:

$$\begin{array}{l}
 e \simeq f \Leftrightarrow \forall \Sigma, \forall \sigma : X \rightarrow \mathcal{L}(\Sigma), \llbracket e \rrbracket_\sigma = \llbracket f \rrbracket_\sigma. \\
 e \lesssim f \Leftrightarrow \forall \Sigma, \forall \sigma : X \rightarrow \mathcal{L}(\Sigma), \llbracket e \rrbracket_\sigma \subseteq \llbracket f \rrbracket_\sigma.
 \end{array}$$

The main result of this paper is a completeness theorem for reversible Kleene lattices:

► **Theorem 24 (Main result).** $\forall e, f \in \mathbb{E}_X, e \equiv f \Leftrightarrow e \simeq f$.

Since all of the axioms in Table 1 are sound for languages, we know that the implication from left to right holds. This paper will thus focus on the converse implication, and will proceed in several steps. Our starting point will be the recently published completeness theorem for identity-free Kleene lattices [5]:

► **Theorem 2.** $\forall e, f \in \mathbb{E}_X^-, e \equiv f \Leftrightarrow e \simeq f$.

► **Remark 3.** In [5], this theorem is established for interpretations of terms as binary relations instead of languages. However both semantic equivalences coincide for this signature [1].

■ **Table 3** The two definitions of \sqsubseteq .

$\varepsilon \sqsubseteq_1 \varepsilon$	$\frac{\varepsilon \sqsubseteq_1 v}{\varepsilon \sqsubseteq_1 \bullet v}$	$\frac{u \sqsubseteq_1 v}{\bullet u \sqsubseteq_1 \bullet v}$	$\frac{xu \sqsubseteq_1 v \quad x \in \Sigma}{xu \sqsubseteq_1 \bullet v}$	$\frac{u \sqsubseteq_1 v \quad x \in \Sigma}{xu \sqsubseteq_1 xv}$
$u \sqsubseteq_2 u$	$\varepsilon \sqsubseteq_2 \bullet$	$\frac{u \sqsubseteq_2 v \quad v \sqsubseteq_2 w}{u \sqsubseteq_2 w}$	$\frac{u \sqsubseteq_2 v \quad u' \sqsubseteq_2 v'}{uu' \sqsubseteq_2 vv'}$	

3 A remark about the empty word

In several places in the proof, it makes some difference whether or not the empty word belongs to the language of some one-free expression. We show here one way one might manipulate this property, that will be of use later on. The main technical result of this section is the following lemma:

► **Proposition 4.** *Given an alphabet Σ , a symbol $\bullet \notin \Sigma$, a map $\sigma : X \rightarrow \mathcal{L}\langle \Sigma \rangle$ and a set of variables $\mathcal{X} \subseteq X$, there are maps $\sigma' : X \rightarrow \mathcal{L}\langle \Sigma \cup \{\bullet\} \rangle$ and $\phi : (\Sigma \cup \{\bullet\})^* \rightarrow \Sigma^*$ such that:*

$$\forall a \in \mathcal{X}, \varepsilon \notin \sigma'(a) \qquad \forall e \in \mathbb{E}'_X, \llbracket e \rrbracket_\sigma = \phi(\llbracket e \rrbracket_{\sigma'} \setminus \varepsilon).$$

Before we can prove it, we need to introduce a few definitions and intermediate lemmas. Let us fix for the remainder of the section an alphabet Σ , and a new symbol $\bullet \notin \Sigma$. We write $\Sigma' := \Sigma \cup \{\bullet\}$. The monoid homomorphism $\phi : \Sigma'^* \rightarrow \Sigma^*$ is generated by

$$\phi(\bullet) := \varepsilon \qquad \forall x \in \Sigma, \phi(x) := x.$$

We will need an ordering \sqsubseteq between words over Σ' , that corresponds intuitively to “ $u \sqsubseteq v$ if u can be obtained by removing some \bullet s from v ”. To define this relation, we provide two deduction systems in Table 3. The definition \sqsubseteq_1 can be thought of as being more algorithmic: it is syntax directed (given a pair of words u, v , there is at most one rule with conclusion $u \sqsubseteq_1 v$), and progressing from bottom to top it removes the superfluous \bullet s from the right hand side. The other definition is more algebraic. It can be summarised as “the smallest precongruence containing $\varepsilon \sqsubseteq_2 \bullet$ ”. It turns out both definitions are equivalent, and we will simply write \sqsubseteq instead of \sqsubseteq_i .

► **Lemma 5.** $\sqsubseteq_1 = \sqsubseteq_2$.

Proof. First we prove that $\sqsubseteq_2 \subseteq \sqsubseteq_1$. By proceeding by induction on the derivation $u \sqsubseteq_2 v$, we see that it amounts to showing that \sqsubseteq_1 1) is a preorder (i.e. reflexive and transitive) 2) contains $\varepsilon \sqsubseteq_1 \bullet$ and 3) satisfies the rule $u \sqsubseteq_1 v$ and $u' \sqsubseteq_1 v'$ implies $uu' \sqsubseteq_1 vv'$.

1. Reflexivity and transitivity can be shown by a simple induction on words.
2. By induction on u we can show that $u \sqsubseteq_1 \bullet u$ (which implies $\varepsilon \sqsubseteq_1 \bullet$).
3. Then we may prove:
 - by induction on u that for any v_1, v_2 we have $v_1 \sqsubseteq_1 v_2 \Rightarrow uv_1 \sqsubseteq_1 uv_2$ and
 - by induction on the derivation $u \sqsubseteq_1 v$ that for every w , we get $uw \sqsubseteq_1 vw$.

These two properties, together with transitivity give us that \sqsubseteq_1 is a precongruence.

For the other containment, we show that $u \sqsubseteq_1 v \Rightarrow u \sqsubseteq_2 v$ by a straightforward induction on the derivation $u \sqsubseteq_1 v$. ◀

11:6 A Complete Axiomatisation of a Fragment of Language Algebra

By induction on the derivation $u \sqsubseteq_2 v$ we may prove the following properties:

$$u \sqsubseteq v \Rightarrow \phi(u) = \phi(v) \quad (3.1)$$

$$u \sqsubseteq v \Rightarrow \bar{u} \sqsubseteq \bar{v} \quad (3.2)$$

By induction on v , and using the definition \sqsubseteq_1 , we get the following decomposition property:

$$u_1 u_2 \sqsubseteq v \Rightarrow \exists v_1, v_2 : v = v_1 v_2 \wedge u_1 \sqsubseteq v_1 \wedge u_2 \sqsubseteq v_2 \quad (3.3)$$

We make the following observations about words greater than ε :

► **Lemma 6.** *For any words $u, v \in \Sigma'$:*

1. $\varepsilon \sqsubseteq u \Leftrightarrow \phi(u) = \varepsilon$.
2. If $\varepsilon \sqsubseteq u, v$ then either $u \sqsubseteq v$ or $v \sqsubseteq u$.

Proof. 1. By (3.1), we only need to check the right to left implication. We do so by induction on u . $\phi(u) = \varepsilon$ means that u is only composed of \bullet s, so there are two case, both being straightforward instances of \sqsubseteq_1 .

2. This second observation is a consequence of the following statement: if $\varepsilon \sqsubseteq u, v$ then $u \sqsubseteq v$ if and only if the length of u is smaller than the length of v . By a simple induction on \sqsubseteq_2 one can show that for any u, v we have the left-to-right implication. For the converse implication we perform the induction on v . ◀

We now arrive at the key property of this ordering:

► **Lemma 7.** *Any words u, v such that $\phi(u) = \phi(v)$ have a least upper bound, i.e. a word $u \sqcup v$ such that $u \sqsubseteq u \sqcup v, v \sqsubseteq u \sqcup v$ and for any word t such that $u \sqsubseteq t$ and $v \sqsubseteq t$, we have $u \sqcup v \sqsubseteq t$.*

Proof. We pose a word $w = \phi(u) = \phi(v)$, and proceed by induction on w . If $w = \varepsilon$, then by the remark we made earlier u and v are ordered, so the least upper bound is the maximum of the two.

Otherwise, we have $\phi(u) = \phi(v) = aw$. By (yet another) induction, we show that this means we can decompose u and v as follows:

$$u = u_1 a u_2 \quad v = v_1 a v_2 \quad \varepsilon \sqsubseteq u_1, v_1 \quad w = \phi(u_2) = \phi(v_2).$$

So we may use our induction hypothesis to get a least upper bound for u_2 and v_2 . Since u_1 and v_1 are both greater than ε , they are ordered. Without loss of generality, let us assume $u_1 \sqsubseteq v_1$. In this case, we claim that $u \sqcup v = v_1 a (u_2 \sqcup v_2)$. It is straightforward to check that $u \sqsubseteq u \sqcup v$ and $v \sqsubseteq u \sqcup v$.

For the remaining property, let t be a word such that $u \sqsubseteq t$ and $v \sqsubseteq t$. We use another decomposition lemma (omitted here), to decompose t as

$$t = t_1 a t_2 \quad u_1 \sqsubseteq t_1 \quad u_2 \sqsubseteq t_2 \quad v_1 \sqsubseteq t_1 \quad v_2 \sqsubseteq t_2.$$

This allows us to conclude: since $u_2 \sqsubseteq t_2$ and $v_2 \sqsubseteq t_2$, then $u_2 \sqcup v_2 \sqsubseteq t_2$, so:

$$u \sqcup v = v_1 a (u_2 \sqcup v_2) \sqsubseteq t_1 a t_2 = t. \quad \blacktriangleleft$$

Notice that by (3.1) and Lemma (7) we get that each equivalence class of the relation $\{\langle u, v \rangle \mid \phi(u) = \phi(v)\}$ forms a join-semilattice.

We may now prove Proposition 4:

Proof of Proposition 4. We fix Σ , σ , and \mathcal{X} as in the statement, and define Σ' and $\phi()$ as in the rest of this section. Finally, σ' is defined as $\sigma'(x) := \{u \mid \phi(u) \in \sigma(x) \wedge (x \in \mathcal{X} \Rightarrow u \neq \varepsilon)\}$.

It is straightforward to check that $\phi(\sigma'(x)) = \sigma(x)$ for any variable x . Therefore we only need to check that this property is preserved by the operators of one-free expressions. For any languages L, M , the following distributivity laws hold:

$$\begin{aligned} \phi(\overline{L}) &= \overline{\phi(L)} & \phi(L \cdot M) &= \phi(L) \cdot \phi(M) \\ \phi(L^+) &= \phi(L)^+ & \phi(L \cup M) &= \phi(L) \cup \phi(M) \end{aligned}$$

However, it is not the case in general that $\phi(L \cap M) = \phi(L) \cap \phi(M)$. To make the induction go through, we will need to show that this identity holds for all the languages generated from the languages $\sigma'(x)$ by the operations $0, \cdot, +, \cap, (-)^+, \overline{(-)}$. This is achieved by identifying some sufficient condition for $\phi(L \cap M) = \phi(L) \cap \phi(M)$, and showing that this condition is satisfied by every language of the shape $\llbracket e \rrbracket_{\sigma'}$.

A good choice for such a condition is the property “being upwards-closed with respect to \sqsubseteq ”, i.e. languages L such that whenever $u \in L$ and $u \sqsubseteq v$, then $v \in L$. Clearly $\sigma'(x)$ is closed for any variable x . Since the property “being closed” is preserved by each operation in the signature of \mathbb{E}'_X , we deduce that for any expression $e \in \mathbb{E}'_X$ the language $\llbracket e \rrbracket_{\sigma'}$ is closed.

Thankfully, for closed languages the missing identity $\phi(L \cap M) = \phi(L) \cap \phi(M)$ holds, thanks to Lemma 7. Thus we may conclude by induction on the expressions that $\llbracket e \rrbracket_{\sigma} = \phi(\llbracket e \rrbracket_{\sigma'})$. For the last step, notice that $\varepsilon \sqsubseteq \bullet$ and $\phi(\varepsilon) = \phi(\bullet)$. Since $\llbracket e \rrbracket_{\sigma'}$ is closed, if $\varepsilon \in \llbracket e \rrbracket_{\sigma'}$, then $\bullet \in \llbracket e \rrbracket_{\sigma'}$, thus $\phi(\llbracket e \rrbracket_{\sigma'} \setminus \varepsilon) = \phi(\llbracket e \rrbracket_{\sigma'}) = \llbracket e \rrbracket_{\sigma}$. ◀

By setting the set \mathcal{X} in the previous proposition to the full set X , we get the straightforward corollary, which will prove useful in the next section.

► **Corollary 8.** *Let e be a one-free expression, then for any expression $f \in \mathbb{E}_X$ we have*

$$e \lesssim f \Leftrightarrow \forall \Sigma, \forall \sigma : X \rightarrow \mathcal{L}(\Sigma), \varepsilon \notin \bigcup_{x \in X} \sigma(x) \Rightarrow \llbracket e \rrbracket_{\sigma} \subseteq \llbracket f \rrbracket_{\sigma}.$$

4 Mirror image

In this section, we show a completeness theorem for one-free expressions. In order to get this result we will use translations between \mathbb{E}'_X and $\mathbb{E}'_{X \times 2}$. An expression $e \in \mathbb{E}'_X$ is *clean*, written $e \in \mathbb{C}_X$, if the mirror operator is only applied to variables. First, notice that we may restrict ourselves to clean expressions thanks to the following inductive function:

$$\begin{aligned} \Upsilon : \mathbb{E}'_X \times 2 &\rightarrow \mathbb{E}'_X \\ \langle 0, b \rangle &\mapsto 0 & \langle e^+, b \rangle &\mapsto \Upsilon \langle e, b \rangle^+ \\ \langle x, \top \rangle &\mapsto x & \langle \bar{e}, \top \rangle &\mapsto \Upsilon \langle e, \perp \rangle \\ \langle x, \perp \rangle &\mapsto \bar{x} & \langle \bar{e}, \perp \rangle &\mapsto \Upsilon \langle e, \top \rangle \\ \langle e + f, b \rangle &\mapsto \Upsilon \langle e, b \rangle + \Upsilon \langle f, b \rangle & \langle e \cdot f, \top \rangle &\mapsto \Upsilon \langle e, \top \rangle \cdot \Upsilon \langle f, \top \rangle \\ \langle e \cap f, b \rangle &\mapsto \Upsilon \langle e, b \rangle \cap \Upsilon \langle f, b \rangle & \langle e \cdot f, \perp \rangle &\mapsto \Upsilon \langle f, \perp \rangle \cdot \Upsilon \langle e, \perp \rangle. \end{aligned}$$

We can show by induction on terms the following properties of Υ :

$$\forall \langle e, b \rangle \in \mathbb{E}'_X \times 2, \Upsilon \langle e, b \rangle \in \mathbb{C}_X. \quad (4.1)$$

$$\forall e \in \mathbb{E}'_X, \Upsilon \langle e, \top \rangle \equiv e \text{ and } \Upsilon \langle e, \perp \rangle \equiv \bar{e}. \quad (4.2)$$

We now define translations between clean expressions and simple expressions:

11:8 A Complete Axiomatisation of a Fragment of Language Algebra

- $\uparrow(-) : \mathbb{C}_X \rightarrow \mathbb{E}_{X \times 2}^-$ replaces mirrored variables \bar{x} with $\langle x, \perp \rangle$ and variables x with $\langle x, \top \rangle$;
- $\downarrow(-) : \mathbb{E}_{X \times 2}^- \rightarrow \mathbb{C}_X$ replaces $\langle x, \top \rangle$ with x and $\langle x, \perp \rangle$ with \bar{x} .

We can easily show by induction the following properties:

$$\forall e \in \mathbb{C}_X, \downarrow \uparrow e = e. \quad (4.3)$$

$$\forall e, f \in \mathbb{E}_{X \times 2}^-, e \equiv f \Rightarrow \downarrow e \equiv \downarrow f. \quad (4.4)$$

The last step to obtain the completeness theorem for \mathbb{E}'_X is the following claim:

▷ **Claim 9.** $\forall e, f \in \mathbb{C}_X, e \simeq f \Rightarrow \uparrow e \simeq \uparrow f$.

► **Lemma 10.** *If Claim 9 holds, then $\forall e, f \in \mathbb{E}'_X, e \equiv f \Leftrightarrow e \simeq f$.*

Proof. By soundness, we know that $e \equiv f \Rightarrow e \simeq f$. For the converse implication:

$$\begin{aligned} e \simeq f &\Rightarrow \Upsilon \langle e, \top \rangle \simeq \Upsilon \langle f, \top \rangle && \text{By soundness and Equation (4.2).} \\ &\Rightarrow \uparrow \Upsilon \langle e, \top \rangle \simeq \uparrow \Upsilon \langle f, \top \rangle && \text{By Claim 9.} \\ &\Rightarrow \uparrow \Upsilon \langle e, \top \rangle \equiv \uparrow \Upsilon \langle f, \top \rangle && \text{By Theorem 2.} \\ &\Rightarrow \downarrow \uparrow \Upsilon \langle e, \top \rangle \equiv \downarrow \uparrow \Upsilon \langle f, \top \rangle && \text{By Equation (4.4).} \\ &\Rightarrow \Upsilon \langle e, \top \rangle \equiv \Upsilon \langle f, \top \rangle && \text{By Equation (4.3).} \\ &\Rightarrow e \equiv f && \text{By Equation (4.2).} \end{aligned}$$

◀

Hence, we only need to show Claim 9 to conclude. To that end, we show that for any clean expression e , any interpretation of $\uparrow e$ can be obtained by applying some transformation to some interpretation of e . Thanks to Corollary 8, we may restrict our attention to interpretation that avoid the empty word. This seemingly mundane restriction turns out to be of significant importance: if the empty word is allowed, the proof of Lemma 11 becomes much more involved. More precisely, we prove the following lemma:

► **Lemma 11.** *Let Σ be some set and $\sigma : X \times 2 \rightarrow \mathcal{L} \langle \Sigma \rangle$ some interpretation such that $\forall x, \varepsilon \notin \sigma(x)$. There exists an alphabet Σ' , an interpretation $\sigma' : X \rightarrow \mathcal{L} \langle \Sigma' \rangle$ and a function $\psi : \mathcal{L} \langle \Sigma' \rangle \rightarrow \mathcal{L} \langle \Sigma \rangle$ such that: $\forall e \in \mathbb{C}_X, \llbracket \uparrow e \rrbracket_\sigma = \psi(\llbracket e \rrbracket_{\sigma'})$.*

Proof. We fix Σ and $\sigma : X \times 2 \rightarrow \mathcal{L} \langle \Sigma \rangle$ as in the statement. Like in the proof of Proposition 4, we set $\Sigma' = \Sigma \cup \{\bullet\}$, with \bullet a fresh letter, and write $\phi(u)$ for the word obtained from $u \in \Sigma'^*$ by erasing every occurrence of \bullet . Additionally we define the function $\eta : \Sigma^* \rightarrow \Sigma'^*$ as follows:

$$\eta(\varepsilon) := \varepsilon \qquad \eta(au) := \bullet a \eta(u) \quad (\langle a, u \rangle \in \Sigma \times \Sigma^*).$$

Clearly, $\phi(\eta(u)) = u$ and $\eta(uv) = \eta(u)\eta(v)$. We may now define σ' and ψ :

$$\sigma'(x) := \{\eta(u) \mid u \in \sigma \langle x, \top \rangle\} \cup \{\overline{\eta(u)} \mid u \in \sigma \langle x, \perp \rangle\} \qquad \psi(L) := \{u \mid \eta(u) \in L\}.$$

This is where the restriction $\varepsilon \notin \sigma(x)$ comes in. Indeed a word w cannot be written both as $w = \eta(u_1)$ and as $w = \overline{\eta(u_2)}$ unless $w = u_1 = u_2 = \varepsilon$. Since σ does not contain the empty word, we may show that $\psi(\sigma'(x)) = \sigma \langle x, \top \rangle$ and $\psi(\overline{\sigma'(x)}) = \sigma \langle x, \perp \rangle$.

ψ distributes over the union and intersection operators. However, it does not hold in general that $\psi(L \cdot M) = \psi(L) \cdot \psi(M)$. Like in the proof of Proposition 4 we will therefore identify a predicate on languages that is sufficient for this identity to hold, is satisfied by

$\sigma'(x)$, and is stable by $\cdot, \cap, +, (-)^+, \overline{(-)}$. In this case we find that an adequate candidate is “ L contains only valid words”, where the set \mathbb{V} of valid words is defined as follows:

$$\frac{u \in \Sigma^+}{\eta(u) \in \mathbb{V}} \qquad \frac{u \in \mathbb{V}}{\bar{u} \in \mathbb{V}} \qquad \frac{u \in \mathbb{V} \quad v \in \mathbb{V}}{uv \in \mathbb{V}}$$

Alternatively, the elements of \mathbb{V} are words over Σ' that can be written as a product $\alpha_1 \dots \alpha_n$ with $1 \leq n$ and each $\alpha_i \in (\Sigma \cdot \bullet) \cup (\bullet \cdot \Sigma)$. One may see from the definitions that $\sigma'(x) \subseteq \mathbb{V}$. \mathbb{V} can also be seen to be trivially closed by concatenation and mirror image. Since the remaining operators are either idempotent (union and intersection) or derived (iteration), we get that $\llbracket e \rrbracket_{\sigma'} \subseteq \mathbb{V}$. This enables us to conclude thanks to the following property:

$$\forall u_1, u_2 \in \mathbb{V}, \eta(u) = u_1 u_2 \Rightarrow \exists v_1, v_2 : u_1 = \eta(v_1) \wedge u_2 = \eta(v_2) \wedge u = v_1 v_2. \quad (4.5)$$

This property enables us to show that $\psi(L \cdot M) = \psi(L) \cdot \psi(M)$ and $\psi(L^+) = \psi(L)^+$, for languages of valid words L, M . Hence we obtain by induction on expressions that for any term $e \in \mathbb{C}_X$, it holds that $\llbracket \uparrow e \rrbracket_{\sigma} = \llbracket e \rrbracket_{\sigma'}$. \blacktriangleleft

► **Theorem 12.** $\forall e, f \in \mathbb{E}'_X, e \equiv f \Leftrightarrow e \simeq f$.

Proof. Thanks to Lemma 10, we only need to check Claim 9. Let e, f be two clean expressions such that $e \simeq f$, we want to prove $\uparrow e \simeq \uparrow f$. According to Corollary 8, we need to compare $\llbracket \uparrow e \rrbracket_{\sigma}$ and $\llbracket \uparrow f \rrbracket_{\sigma}$ for some $\sigma : X \times 2 \rightarrow \mathcal{L} \langle \Sigma \rangle$ such that $\varepsilon \notin \bigcup_{x \in X \times 2} \sigma(x)$. By Lemma 11, we may express these languages as respectively $\psi(\llbracket e \rrbracket_{\sigma'})$ and $\psi(\llbracket f \rrbracket_{\sigma'})$. Since $e \simeq f$, we get that $\llbracket e \rrbracket_{\sigma'} = \llbracket f \rrbracket_{\sigma'}$, thus proving the desired identity and concluding the proof. \blacktriangleleft

5 Interlude: tests

Before we start with the main proof, we define *tests* and establish a few result about them. Given a list of variables $u \in X^*$, we define the term θ_u by induction on u as $\theta_\varepsilon := 1$ and $\theta_{au} := a \cap \theta_u$. Thanks to the following remark, we will hereafter consider θ_A for $A \in \mathcal{P}_f(X)$:

► **Remark 13.** Let u, v be two lists of variables containing the same letters (meaning a variable appears in u if and only if it appears in v). Then $\theta_u \equiv \theta_v$.

The following property explains our choice of terminology: the function $\lambda \sigma. \llbracket \theta_A \rrbracket_{\sigma}$ can be seen as a boolean predicate testing whether the empty word is in each of the $\sigma(a)$ for $a \in A$.

► **Lemma 14.** Let Σ be some alphabet and $\sigma : X \rightarrow \mathcal{L} \langle \Sigma \rangle$. Then either $\forall a \in A, \varepsilon \in \sigma(a)$, in which case $\llbracket \theta_A \rrbracket_{\sigma} = \varepsilon$, or $\llbracket \theta_A \rrbracket_{\sigma} = \emptyset$.

Tests satisfy the following universal identities, with $A, B \in \mathcal{P}_f(X)$ and $e, f \in \mathbb{E}_X$:

$$\theta_A \leq 1 \quad (5.1)$$

$$\theta_A \cap \theta_B \equiv \theta_A \cdot \theta_B \equiv \theta_{A \cup B} \quad (5.2)$$

$$\theta_A \equiv \theta_A \cdot \theta_A \quad (5.3)$$

$$a \in A \Rightarrow \theta_A \leq a \quad (5.4)$$

$$\theta_A \cdot e \equiv e \cdot \theta_A \quad (5.5)$$

$$(\theta_A \cdot e) \cap (\theta_B \cdot f) \equiv \theta_{A \cup B} \cdot (e \cap f) \quad (5.6)$$

$$\theta_A^+ \equiv \overline{\theta_A} \equiv \theta_A. \quad (5.7)$$

11:10 A Complete Axiomatisation of a Fragment of Language Algebra

We now want to compare tests with other tests or with expressions. Let us define the following interpretation for any finite set $A \in \mathcal{P}_f(X)$.

$$\sigma_A : X \rightarrow \mathcal{L}(\emptyset)$$

$$x \mapsto \begin{cases} \varepsilon & \text{if } x \in A \\ \emptyset & \text{otherwise.} \end{cases}$$

Note that the alphabet here does not matter, since we only want the unit language and the empty language. This interpretation enables us to establish the following lemma:

► **Lemma 15.** *For any $A, B \in \mathcal{P}_f(X)$, the following are equivalent:*

$$(i) \ \varepsilon \in \llbracket \theta_B \rrbracket_{\sigma_A} \quad (ii) \ B \subseteq A \quad (iii) \ \theta_A \leq \theta_B \quad (iv) \ \theta_A \lesssim \theta_B.$$

Proof. Assume (i) holds, i.e. $\varepsilon \in \llbracket \theta_B \rrbracket_{\sigma_A}$. By Lemma 14 this means that for every $a \in B$ we have $\varepsilon \in \sigma_A(a)$ which by definition of σ_A ensures that $a \in A$. Thus we have shown that (ii) holds. We show that (ii) implies (iii) by induction on the size of B :

- if $B = \emptyset$, by Equation (5.1) $\theta_A \leq 1 = \theta_\emptyset$.
- if $B = \{a\} \cup B'$ with $a \notin B'$, since $B \subseteq A$ we have $a \in A$ and $B' \subseteq A$. By induction hypothesis we know that $\theta_A \leq \theta_{B'}$. By Remark 13 we get that $\theta_A \equiv a \cap \theta_{B'}$. Hence we get:

$$\theta_A \equiv a \cap \theta_{B'} \leq a \cap \theta_{B'} = \theta_B.$$

Thanks to soundness we have that (iii) implies (iv). For the last implication, notice that by construction of σ_A we have $\varepsilon \in \llbracket \theta_A \rrbracket_{\sigma_A}$. Therefore if $\theta_A \lesssim \theta_B$ then we can conclude that $\varepsilon \in \llbracket \theta_A \rrbracket_{\sigma_A} \subseteq \llbracket \theta_B \rrbracket_{\sigma_A}$. ◀

We now define a function $\mathbb{I} : \mathbb{E}_X \rightarrow \mathcal{P}_f(\mathcal{P}_f(X))$, whose purpose is to represent as a sum of tests the intersection of an arbitrary expression with 1:

$$\mathbb{I}(0) := \emptyset \quad \mathbb{I}(1) := \{\emptyset\} \quad \mathbb{I}(x) := \{\{x\}\} \quad \mathbb{I}(e + f) := \mathbb{I}(e) \cup \mathbb{I}(f)$$

$$\mathbb{I}(e \cdot f) = \mathbb{I}(e \cap f) := \{A \cup B \mid \langle A, B \rangle \in \mathbb{I}(e) \times \mathbb{I}(f)\} \quad \mathbb{I}(e^+) = \mathbb{I}(\bar{e}) := \mathbb{I}(e).$$

► **Lemma 16.** $\forall e \in \mathbb{E}_X, 1 \cap e \equiv \sum_{C \in \mathbb{I}(e)} \theta_C$.

► **Corollary 17.** $\forall e \in \mathbb{E}_X, \forall A \in \mathcal{P}_f(X), \theta_A \leq e \Leftrightarrow \theta_A \lesssim e$.

Proof. We only need to show the implication from right to left. Assume $\theta_A \lesssim e$. This implies $1 \cap \theta_A \lesssim 1 \cap e$, and since $\theta_A \leq 1$ we know that $1 \cap \theta_A \equiv \theta_A$ which by soundness implies $\theta_A \simeq 1 \cap \theta_A$. Combining this with Lemma 16, we get that $\theta_A \simeq 1 \cap \theta_A \lesssim 1 \cap e \simeq \sum_{C \in \mathbb{I}(e)} \theta_C$. By Lemma 15, we know that $\varepsilon \in \llbracket \theta_A \rrbracket_{\sigma_A}$, which means that $\varepsilon \in \left[\left[\sum_{C \in \mathbb{I}(e)} \theta_C \right]_{\sigma_A} \right] = \bigcup_{C \in \mathbb{I}(e)} \llbracket \theta_C \rrbracket_{\sigma_A}$. Therefore there must be some $B \in \mathbb{I}(e)$ such that $\varepsilon \in \llbracket \theta_B \rrbracket_{\sigma_A}$ which by Lemma 15 tells us that $\theta_A \leq \theta_B$. We may now conclude:

$$\theta_A \leq \theta_B \leq \sum_{C \in \mathbb{I}(e)} \theta_C \equiv 1 \cap e \leq e. \quad \blacktriangleleft$$

► **Remark 18.** The word “test” is reminiscent of Kleene algebra with tests (KAT)[7]. Indeed according to Equation (5.1) our tests are sub-units, like in KAT. However unlike in KAT, there are non-test terms t such that $t \leq 1$. In general such terms are sums of tests, as can be inferred from Lemma 16 (because for every sub-unit $e \leq 1$, we have $e \equiv 1 \cap e \equiv \sum_{C \in \mathbb{I}(e)} \theta_C$).

6 Completeness of reversible Kleene lattices

To tackle this completeness proof, we will proceed in three steps. Since we already proved soundness, and since an equality can be equivalently expressed as a pair of containments, we start from the following statement:

$$\forall e, f \in \mathbb{E}_X, e \lesssim f \Rightarrow e \leq f.$$

First, we will show that any expression in \mathbb{E}_X can be equivalently written as a sum of terms that are either tests or products $\theta_A \cdot e$ of a test and a one-free expression. The case of tests having been dispatched already (Corollary 17), this reduces the problem to:

$$\forall e \in \mathbb{E}'_X, \forall A \in \mathcal{P}_f(X), \forall f \in \mathbb{E}_X, \theta_A \cdot e \lesssim f \Rightarrow \theta_A \cdot e \leq f.$$

Second, we will show that for any pair $\langle A, f \rangle \in \mathcal{P}_f(X) \times \mathbb{E}_X$, there exists an expression $\langle f \rangle_A \in \mathbb{E}_X$ such that $\theta_A \cdot \langle f \rangle_A \leq f$ and whenever $\theta_A \cdot e \lesssim f$ we have $e \lesssim \langle f \rangle_A$. This further reduces the problem into:

$$\forall e \in \mathbb{E}'_X, \forall f \in \mathbb{E}_X, e \lesssim f \Rightarrow e \leq f.$$

For the third and last step, we show that for any expression $f \in \mathbb{E}_X$, there is an expression $[f] \in \mathbb{E}'_X$ such that $[f] \leq f$ and whenever $e \lesssim f$ for $e \in \mathbb{E}'_X$ we have $e \lesssim [f]$. This is enough to conclude thanks to Theorem 12.

In the next three subsections, we introduce constructions and prove lemmas necessary for each step. Then, in Section 6.4 we put them all together to show the main result.

6.1 First step: normal forms

A normal form is either an expression of the shape θ_A or of the shape $\theta_A \cdot e$ with $e \in \mathbb{E}'_X$. We denote by \mathbb{NF} the set of normal forms. The main result of this section is the following:

► **Lemma 19.** *For any $e \in \mathbb{E}_X$ there exists a finite set $\mathcal{N}(e) \subseteq \mathbb{NF}$ such that $e \equiv \sum_{\eta \in \mathcal{N}(e)} \eta$.*

Proof. We show by induction on e how to build $\mathcal{N}(e)$. The correctness of the construction is fairly straightforward, and is left as an exercise : we will only state the relevant proof obligations when appropriate.

For constants, variables, and unions, the choice is rather obvious:

$$\mathcal{N}(0) := \emptyset \quad \mathcal{N}(1) := \{\theta_\emptyset\} \quad \mathcal{N}(x) := \{\theta_\emptyset \cdot x\} \quad \mathcal{N}(e + f) := \mathcal{N}(e) \cup \mathcal{N}(f).$$

The case of mirror image is also rather straightforward:

$$\mathcal{N}(\bar{e}) := \{\theta_A \mid \theta_A \in \mathcal{N}(e)\} \cup \{\theta_A \cdot \bar{e}' \mid \theta_A \cdot e' \in \mathcal{N}(e)\}.$$

For concatenations, we define the product $\eta \odot \gamma$ of two normal forms $\eta, \gamma \in \mathbb{NF}$ as:

$$\theta_A \odot \theta_B := \theta_{A \cup B} \quad \theta_A \odot \theta_B \cdot e := \theta_A \cdot e \odot \theta_B := \theta_{A \cup B} \cdot e \quad \theta_A \cdot e \odot \theta_B \cdot f := \theta_{A \cup B} \cdot (e \cdot f).$$

We then define $\mathcal{N}(e \cdot f) := \{\eta \odot \gamma \mid \langle \eta, \gamma \rangle \in \mathcal{N}(e) \times \mathcal{N}(f)\}$. For correctness of the construction, we would have to prove that $\forall \eta, \gamma \in \mathbb{NF}, \eta \cdot \gamma \equiv \eta \odot \gamma$.

For intersections, we define $\otimes : \mathbb{NF} \times \mathbb{NF} \rightarrow \mathcal{P}_f(\mathbb{NF})$:

$$\begin{aligned} \theta_A \otimes \theta_B &:= \{\theta_{A \cup B}\} & \theta_A \otimes \theta_B \cdot e &:= \theta_A \cdot e \otimes \theta_B := \{\theta_{A \cup B \cup C} \mid C \in \mathbb{I}(e)\} \\ \theta_A \cdot e \otimes \theta_B \cdot f &:= \theta_{A \cup B} \cdot (e \cap f). \end{aligned}$$

11:12 A Complete Axiomatisation of a Fragment of Language Algebra

We then define $\mathcal{N}(e \cap f) := \bigcup_{\langle \eta, \gamma \rangle \in \mathcal{N}(e) \times \mathcal{N}(f)} \eta \otimes \gamma$.

Finally, for iterations we use the following definition:

$$\mathcal{N}(e^+) := \left\{ \theta_A \mid \theta_A \in \mathcal{N}(e) \right\} \cup \left\{ \theta_{\cup_i A_i} \cdot \left(\sum_i e_i \right)^+ \mid \left\{ \theta_{A_i} \cdot e_i \mid i \leq n \right\} \subseteq \mathcal{N}(e) \right\}. \quad \blacktriangleleft$$

► **Remark.** In [1], a similar lemma was proved (Lemma 3.4). However, the proof in that paper is slightly wrong, as it fails to consider the cases $\theta_A \cap \theta_B$ (easy) and $\theta_A \cap \theta_B \cdot e$ (more involved).

6.2 Second step: removing tests on the left

Here we want to transform an inequation $\theta_A \cdot e \lesssim f$, into one of the shape $e \lesssim \langle f \rangle_A$, while maintaining that $\theta_A \cdot \langle f \rangle_A \leq f$. The construction of $\langle f \rangle_A$ is fairly straightforward, the intuition being that θ_A forces us to only consider interpretations such that $a \in A \Rightarrow \varepsilon \in \llbracket a \rrbracket_\sigma$. Therefore, for any $a \in A$ we replace in f every occurrence of a with $1 + a$.

► **Lemma 20.** $\theta_A \cdot \langle f \rangle_A \leq f \leq \langle f \rangle_A$.

Proof. Since $a \leq 1 + a$, we can show by induction that $f \leq \langle f \rangle_A$. Also, if $a \in A$:

$$\begin{aligned} \theta_A \cdot (1 + a) &\equiv \theta_A + \theta_A \cdot a && \text{By (1d.1) and (1b.4)} \\ &\equiv \theta_A \cdot \theta_A + \theta_A \cdot a && \text{By (5.3)} \\ &\equiv \theta_A \cdot (\theta_A + a) && \text{By (1b.4)} \\ &\equiv \theta_A \cdot a. && \text{By (5.4)} \end{aligned}$$

This proves for the case of variables that $\theta_A \cdot \langle f \rangle_A \leq f$, and can be generalised to arbitrary expressions by a simple induction. \blacktriangleleft

For the other property, we rely on the following lemma:

► **Lemma 21.** *Let Ξ be some alphabet, and $\sigma : X \rightarrow \mathcal{L}(\Xi)$ be an interpretation such that $\forall x \in X, \varepsilon \notin \sigma(x)$. Then $\llbracket \langle f \rangle_A \rrbracket_\sigma = \llbracket \langle f \rangle_A \rrbracket_\tau$, where $\tau : X \rightarrow \mathcal{L}(\Xi)$
 $x \mapsto \sigma(x) \cup \{\varepsilon \mid x \in A\}$.*

Proof. The result follows from a straightforward induction, the only interesting case being that of variables $x \in A$. This case is a simple consequence of our definitions:

$$\llbracket 1 + a \rrbracket_\tau = \varepsilon \cup \tau(a) = \varepsilon \cup \sigma(a) \cup \varepsilon = \varepsilon \cup \sigma(a) = \llbracket 1 + a \rrbracket_\sigma. \quad \blacktriangleleft$$

► **Corollary 22.** *Let $\langle A, e \rangle \in \mathcal{P}_f(X) \times \mathbb{E}'_X$ such that $\theta_A \cdot e \lesssim f$, then $e \lesssim \langle f \rangle_A$.*

Proof. Since by Lemma 20 we have $f \leq \langle f \rangle_A$ by soundness and transitivity of \lesssim we have $\theta_A \cdot e \lesssim \langle f \rangle_A$. We want to show that $e \lesssim \langle f \rangle_A$, so by Corollary 8 we only need to check that for any interpretation $\sigma : X \rightarrow \mathcal{L}(\Sigma)$ such that $\varepsilon \notin \bigcup_{x \in X} \sigma(x)$ we have $\llbracket e \rrbracket_\sigma \subseteq \llbracket \langle f \rangle_A \rrbracket_\sigma$. If we take τ like in Lemma 21, we get that 1) since for every variable $\sigma(x) \subseteq \tau(x)$, $\llbracket e \rrbracket_\sigma \subseteq \llbracket e \rrbracket_\tau$ and 2) since for every $a \in A$ we have $\varepsilon \in \tau(a)$, we get $\llbracket \theta_A \rrbracket_\tau = \varepsilon$. Together these tell us that $\llbracket e \rrbracket_\sigma \subseteq \llbracket e \rrbracket_\tau = \varepsilon \cdot \llbracket e \rrbracket_\tau = \llbracket \theta_A \cdot e \rrbracket_\tau$. Since $\theta_A \cdot e \lesssim \langle f \rangle_A$ we know that $\llbracket \theta_A \cdot e \rrbracket_\tau \subseteq \llbracket \langle f \rangle_A \rrbracket_\tau$, and by Lemma 21 we know $\llbracket \langle f \rangle_A \rrbracket_\sigma = \llbracket \langle f \rangle_A \rrbracket_\tau$. We may therefore conclude that $\llbracket e \rrbracket_\sigma \subseteq \llbracket \theta_A \cdot e \rrbracket_\tau \subseteq \llbracket \langle f \rangle_A \rrbracket_\tau = \llbracket \langle f \rangle_A \rrbracket_\sigma$. \blacktriangleleft

6.3 Third step: removing tests on the right

This last step relies on Proposition 4 and Lemma 19.

► **Lemma 23.** *For any expression $f \in \mathbb{E}_X$, there exists a one-free expression $[f] \in \mathbb{E}'_X$ such that $[f] \leq f$ and for any one-free expression $e \in \mathbb{E}'_X$ such that $e \lesssim f$ we have $e \lesssim [f]$. In other words, $[f]$ is the maximum of the set $\{e \in \mathbb{E}'_X \mid e \leq f\}$.*

Proof. We define $[f] := \sum_{\theta_\emptyset \cdot f' \in \mathcal{N}(f)} f'$. We can easily check that $[f] \leq f$:

$$[f] \equiv 1 \cdot [f] = \theta_\emptyset \cdot \sum_{\theta_\emptyset \cdot f' \in \mathcal{N}(f)} f' \equiv \sum_{\theta_\emptyset \cdot f' \in \mathcal{N}(f)} \theta_\emptyset \cdot f' \leq \sum_{\eta \in \mathcal{N}(f)} \eta \equiv f.$$

For the other property, we rely on Proposition 4. Assume $e \lesssim f$, we want to show that $e \lesssim [f]$. By Corollary 8, it is enough to check that $\llbracket e \rrbracket_\sigma \subseteq \llbracket [f] \rrbracket_\sigma$ for interpretations σ such that $\forall x \in X, \varepsilon \notin \sigma(x)$. Let σ be such an interpretation, and u some word such that $u \in \llbracket e \rrbracket_\sigma$. Notice that the condition on σ ensures that $\forall x \in X, 1 \cap \sigma(x) = \emptyset$, hence $\llbracket \theta_A \rrbracket_\sigma \neq \emptyset$ implies that $A = \emptyset$ by Lemma 14. Also, because $\sigma(x)$ never contains the empty word and e does not feature the constant 1, u must be different from ε . Since $e \lesssim f$, we already know that $u \in \llbracket f \rrbracket_\sigma$. By Lemma 19 and soundness, we know that there is a normal form $\eta \in \mathcal{N}(f)$ such that $u \in \llbracket \eta \rrbracket_\sigma$. Since $u \neq \varepsilon$, η cannot be a test: that would imply by (5.1) that $\eta \leq 1$, hence $\llbracket \eta \rrbracket_\sigma \subseteq \llbracket 1 \rrbracket_\sigma = \varepsilon$. Therefore we know that there is a term $\theta_A \cdot f' \in \mathcal{N}(f)$ such that $u \in \llbracket \theta_A \cdot f' \rrbracket_\sigma$. This means that $u \in \llbracket f' \rrbracket_\sigma$ and $\varepsilon \in \llbracket \theta_A \rrbracket_\sigma$. As we have noticed before, this means that $A = \emptyset$. Thus we get $u \in \llbracket f' \rrbracket_\sigma$ and $\theta_\emptyset \cdot f' \in \mathcal{N}(f)$, which ensures that $u \in \llbracket [f] \rrbracket_\sigma$. ◀

6.4 Main theorem

We may now prove the main result of this paper:

► **Theorem 24 (Main result).** $\forall e, f \in \mathbb{E}_X, e \equiv f \Leftrightarrow e \simeq f$.

Proof. Since $e \equiv f \Leftrightarrow e \leq f \wedge f \leq e$ and $e \simeq f \Leftrightarrow e \lesssim f \wedge f \lesssim e$, we focus instead on proving that $e \leq f \Leftrightarrow e \lesssim f$. By soundness we know that $e \leq f \Rightarrow e \lesssim f$, so we only need to show the converse implication.

Let $e, f \in \mathbb{E}_X$ such that $e \lesssim f$. By Lemma 19 we can show that $e \equiv \sum_{\eta \in \mathcal{N}(e)} \eta$. Let $\eta \in \mathcal{N}(e)$. Thanks to the properties of \lesssim we have that $\eta \lesssim f$. There are two cases for η :

- either $\eta = \theta_A$ for some $A \in \mathcal{P}_f(X)$, in which case we have $\eta \leq f$ by Corollary 17;
- or $\eta = \theta_A \cdot e'$ with $A \in \mathcal{P}_f(X)$ and $e' \in \mathbb{E}'_X$. In that case, by Corollary 22 we have $e' \lesssim \langle f \rangle_A$, and by Lemma 23 we get $e' \lesssim \llbracket \langle f \rangle_A \rrbracket$. Since both e' and $\llbracket \langle f \rangle_A \rrbracket$ are one-free, we may apply Theorem 12 to get a proof that $e' \leq \llbracket \langle f \rangle_A \rrbracket$. Therefore

$$\begin{aligned} \eta = \theta_A \cdot e' &\leq \theta_A \cdot \llbracket \langle f \rangle_A \rrbracket \leq \theta_A \cdot \langle f \rangle_A && \text{By Lemma 23.} \\ &\leq f && \text{By Lemma 20.} \end{aligned}$$

In both cases we have established that $\eta \leq f$, so by monotonicity we show that

$$e \equiv \sum_{\eta \in \mathcal{N}(e)} \eta \leq \sum_{\eta \in \mathcal{N}(e)} f \leq f. \quad \blacktriangleleft$$

7 The “top” problem

In reversible Kleene lattices, union and intersection form a distributive lattice, and 0 acts as both the unit of union and the annihilator of intersection. All that is missing to get a bounded distributive lattice is the unit of intersection and annihilator of union, namely the constant \top , to be interpreted as the full language. However, this turns out to be more complicated than one might think.

The first idea that comes to mind is to add the sole axiom $\top + e = \top$. This axiom just says that for any expression $e \leq \top$, and is enough to show that $e \cap \top \equiv \top \cap e \equiv e$. It is obviously sound, so we get soundness of the resulting axiomatic equivalence. This axiomatic equivalence can be reduced without too much difficulty to that of reversible Kleene lattices, thanks to the following remark:

► **Remark 25.** If we write \mathbb{E}_X^\top for expressions with \top , let $\phi : \mathbb{E}_X^\top \rightarrow \mathbb{E}_{X+1}$ be the function that replaces every occurrence of \top with $(\sum_{a \in X+1} (a + \bar{a}))^*$. Then the following identity holds: $\forall e, f \in \mathbb{E}_X^\top, e \equiv f \Leftrightarrow \phi(e) \equiv \phi(f)$.

This same construction, when applied to expressions without intersections, yields a completeness proof. In the presence of intersection however it is not complete. We illustrate this with two examples.

► **Example 26 (Levi’s lemma).** Levi’s lemma for strings [9] states that whenever we have two factorisations of the same word, i.e. $u_1 u_2 = v_1 v_2$, then either $\exists w, u_1 = v_1 w \wedge v_2 = w u_2$ or $\exists w, v_1 = u_1 w \wedge u_2 = w v_2$. If we now move from words to languages, it means that every word that can be obtained simultaneously as $L_1 \cdot L_2$ and $M_1 \cdot M_2$ also belongs to either $L_1 \cdot \top \cdot M_2$ or $M_1 \cdot \top \cdot L_2$. In other words, the following inequation holds:

$$(e_1 \cdot e_2) \cap (f_1 \cdot f_2) \lesssim (e_1 \cdot \top \cdot f_2) + (f_1 \cdot \top \cdot e_2).$$

However this equation is not derivable. This law also contrasts with the properties we can observe in every fragment of this algebra that we have studied: in every case, if a term without \star or $+$ is smaller than a term $e + f$, then it must be smaller than either e or f . One can plainly see that it is not the case here.

► **Example 27 (Factorisation).** Another troubling example is the following:

$$(a \cdot b) \cap (a \cdot c) \lesssim a \cdot ((\top \cdot b) \cap (\top \cdot c)).$$

As before, this inequation is valid, but it is not derivable, and it does not involve unions. This suggests that the (in-)equational theory of languages with just the signature $\langle \cdot, \cap, \top \rangle$ is already non-trivial. We believe that the key to adding \top to Kleene lattices lies with a better understanding of the theory of this smaller signature.

References

- 1 Hajnal Andréka, Szabolcs Mikulás, and István Németi. The Equational Theory of Kleene Lattices. *Theor. Comput. Sci.*, 412(52):7099–7108, 2011. doi:10.1016/j.tcs.2011.09.024.
- 2 S. L. Bloom, Z. Ésik, and Gh. Stefanescu. Notes on Equational Theories of Relations. *algebra universalis*, 33(1):98–126, March 1995. doi:10.1007/BF01190768.
- 3 Paul Brunet. Reversible Kleene Lattices. In Kim G. Larsen, Hans L. Bodlaender, and Jean-Francois Raskin, editors, *42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017)*, volume 83 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 66:1–66:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.MFCS.2017.66.
- 4 John H. Conway. *Regular algebra and finite machines*. Chapman and Hall Mathematics Series, 2012.
- 5 Amina Doumane and Damien Pous. Completeness for Identity-Free Kleene Lattices. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory (CONCUR 2018)*, volume 118 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CONCUR.2018.18.
- 6 D. Kozen. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. *Information and Computation*, 110(2):366–390, May 1994. doi:10.1006/inco.1994.1037.
- 7 Dexter Kozen. Kleene Algebra with Tests. *ACM Trans. Program. Lang. Syst.*, 19(3):427–443, 1997. doi:10.1145/256167.256195.
- 8 Daniel Kroh. Complete Systems of B-Rational Identities. *Theoretical Computer Science*, 89(2):207–343, October 1991. doi:10.1016/0304-3975(91)90395-I.
- 9 Frank W. Levi. On semigroups. *Bull. Calcutta Math. Soc.*, 36(141-146):82, 1944.

Proof Complexity of Systems of (Non-Deterministic) Decision Trees and Branching Programs

Sam Buss

Dept. of Mathematics, UC San Diego, USA

<https://www.math.ucsd.edu/~sbuss/>

sbuss@ucsd.edu

Anupam Das

Dept. of Computer Science, University of Copenhagen, Denmark

anupam.das@di.ku.dk

Alexander Knop

Dept. of Mathematics, UC San Diego, USA

<https://alexanderknop.github.io/>

aknop@ucsd.edu

Abstract

This paper studies propositional proof systems in which lines are sequents of decision trees or branching programs, deterministic or non-deterministic. Decision trees (DTs) are represented by a natural term syntax, inducing the system LDT, and non-determinism is modelled by including disjunction, \vee , as primitive (system LNDDT). Branching programs generalise DTs to dag-like structures and are duly handled by extension variables in our setting, as is common in proof complexity (systems eLDT and eLNDDT).

Deterministic and non-deterministic branching programs are natural nonuniform analogues of log-space (L) and nondeterministic log-space (NL), respectively. Thus eLDT and eLNDDT serve as natural systems of reasoning corresponding to L and NL, respectively.

The main results of the paper are simulation and non-simulation results for tree-like and dag-like proofs in LDT, LNDDT, eLDT and eLNDDT. We also compare them with Frege systems, constant-depth Frege systems and extended Frege systems.

2012 ACM Subject Classification Theory of computation \rightarrow Computational complexity and cryptography

Keywords and phrases proof complexity, decision trees, branching programs, logspace, sequent calculus, non-determinism, low-depth complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.12

Related Version A full version of the paper is available at [5], <https://arxiv.org/abs/1910.08503>.

Funding *Sam Buss*: This work supported in part by Simons Foundation grant 578919.

Anupam Das: This work was supported by a Marie Skłodowska-Curie fellowship, *Monotonicity in Logic and Complexity*, ERC project 753431.

1 Introduction

Propositional proof systems are widely studied because of their connections to feasible complexity classes and their usefulness for computer-based reasoning. The first connections to computational complexity arose largely from the work of Cook and Reckhow [11, 16, 17], showing a connection to the NP-coNP question. These results, building on the work of Tseitin [33] initiated the study of the relative efficiency of propositional proof systems. The present paper introduces propositional proof systems that are closely connected to log-space (L) and nondeterministic log-space (NL).



© Sam Buss, Anupam Das, and Alexander Knop;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 12; pp. 12:1–12:17

Leibniz International Proceedings in Informatics



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

Our original motivation for this study was to investigate propositional proof systems corresponding to the first-order bounded arithmetic theories VL and VNL for L and NL, see [15]. This follows a long line of work defining formal theories of bounded arithmetic that correspond to computational complexity classes, as well as to provability in propositional proof systems. The first results of this type were due (independently) to Paris and Wilkie [30] who gave a translation from $I\Delta_0$ to constant-depth Frege (AC^0 -Frege) proofs and to Cook [11] who gave a translation from PV to extended Frege ($e\mathcal{F}$) proofs. Since the first-order bounded arithmetic theory S_2^1 is conservative over the equational theory PV, Cook’s translation also applies to the bounded arithmetic theory S_2^1 [7]. As shown in the table below, similar propositional translations have since been given for a range of other theories, including first-order, second-order and equational theories.

Formal Theories	Propositional Proof Systems	Complexity Class	
PV, S_2^1	$e\mathcal{F}$	P	[11, 7]
PSA, U_2^1	QBF	PSPACE	[18, 7]
T_2^i, S_2^{i+1}	G_i, G_{i+1}^*	$P^{\Sigma_i^P}$	[27, 28, 7]
VNC^0	Frege (\mathcal{F})	ALogTime	[14, 15, 1]
VL	GL^*	L	[31, 15]
VNL	GNL^*	NL	[32, 15]

For an introduction to these and related results, see the books [7, 15, 25, 26]. A hallmark of the table above is that the lines in the propositional proofs express (nonuniform) properties in the corresponding complexity class. For instance, lines in a Frege proof are propositional formulas, for which the evaluation problem is complete for alternating log-time (ALogTime), cf. [8]. Likewise, lines in an $e\mathcal{F}$ proof are (implicitly) Boolean circuits, for which the evaluation problem is complete for P, cf. [29].

This paper’s main goal is to define alternatives for the proof systems GL^* and GNL^* corresponding to log-space and nondeterministic log-space (see [31, 32, 12, 13]). GL^* restricts cut formulas to be “ $\Sigma CNF(2)$ ” formulas; the subformula property then implies that proofs contain only $\Sigma CNF(2)$ formulas when proving $\Sigma CNF(2)$ theorems. GNL^* similarly restricts cut formulas to be “ $\Sigma Krom$ ” formulas.¹ $\Sigma CNF(2)$ and $\Sigma Krom$ do have expressive power equivalent to nonuniform L and NL respectively [22, 19], but they are somewhat ad hoc classes of quantified formulas, and their connections to L and NL are indirect. In this paper, we propose new proof systems, $eLDT$ and $eLNDDT$, as alternatives for GL^* and GNL^* respectively. The lines in $eLDT$ and $eLNDDT$ proofs are sequents of formulas expressing *branching programs* and *nondeterministic branching programs*, respectively. This follows an earlier unpublished suggestion of S. Cook [10], who gave a system for L based on branching programs via “Prover-Liar” games (see [9]). The advantage of our systems is that deterministic and nondeterministic branching programs correspond directly to nonuniform L and NL respectively and do not require the use of quantified formulas. (See [34] for a comprehensive introduction to branching programs.)

¹ A $\Sigma Krom$ formula has the form if it has the form $\exists z\phi(z, \vec{x})$, where ϕ is a conjunction $C_1 \wedge C_2 \wedge \dots \wedge C_n$ with each C_i a disjunction of any number of x -literals and at most two z -literals.

To design the proof systems eLDT and eLNDT, we need to choose representations for branching programs. For this, we use a formula-based representation, as this fits well into the customary frameworks for proof systems. Since formulas only represent tree-structures, we first define the systems LDT and LNNT for decision trees and non-deterministic decision trees, respectively. From here dag-like structures are described using extension variables, allowing us to abbreviate complex formulas by fresh variables, yielding the systems eLDT and eLNNT. An example this is given in Figure 2 on page 12. This is similar to the way the extension variables in extended Frege proofs allow circuits to be expressed by small formulas.

We start in Section 2 describing proof systems LDT and LNNT that work with just deterministic and nondeterministic decision trees (without extension variables). Deterministic decision trees are represented by formulas using a single “case” or “if-then-else” connective, written in infix notation ApB , which means “if p is false, then A , else B ”. The condition p is required to be a literal, but A and B are arbitrary formulas. The system LDT is a sequent calculus system in which all formulas are decision trees. Nondeterministic decision trees may further be composed by disjunctions, allowing formulas of the form $(A \vee B)$. The system LNNT is a sequent calculus in which all formulas are nondeterministic decision trees. LDT and LNNT are weak systems; in fact, they are both polynomially simulated by depth-2 LK that is, by the sequent calculus LK with all formulas are depth two, allowing proofs to be dag-like. Figure 1 shows the equivalences between systems as currently established; those that concern LDT and LNNT are given in Section 4. Section 5 introduces the proof systems eLDT and eLNNT for branching programs and nondeterministic branching programs.

One issue in designing these proof systems is the treatment of isomorphic or bisimilar branching programs. One approach is to allow proofs to freely replace any branching program with any isomorphic or bisimilar branching program by means of additional axioms, e.g. as done by Jeřábek [21] for the reformulation of extended Frege using Boolean circuits as lines. The problem with using isomorphism or bisimilarity axioms is that these problems (for branching programs) are in NL but not known to be in L. Such axioms are thus undesirable, at least for eLDT, as it is a proof system for log-space. We instead adopt a more conservative approach: the equivalence of bisimilar branching programs must be proved explicitly.

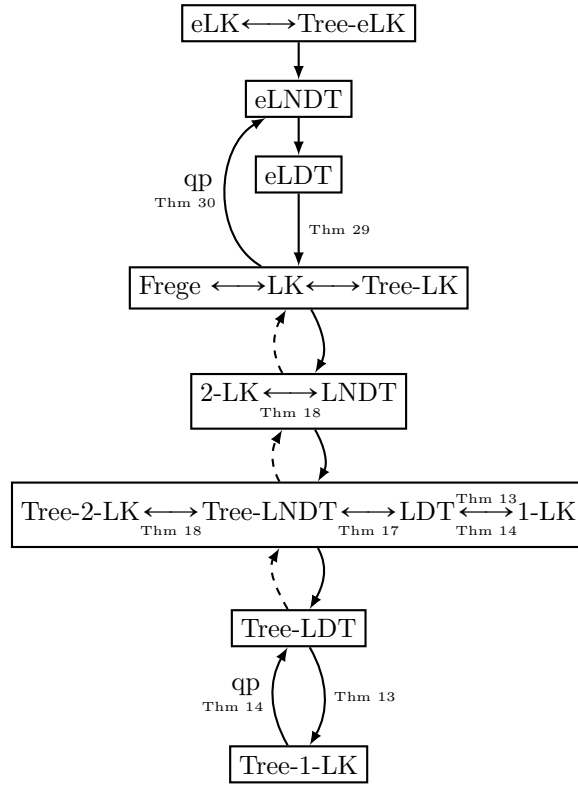
Since formulas in eLDT and eLNNT proofs express nonuniform L and NL properties, respectively, they are intermediate in expressive power between Boolean formulas (expressing NC^1 properties) and Boolean circuits (expressing nonuniform P properties). Thus it is not surprising that, as shown in Figure 1, these two systems are between Frege and extended Frege in strength. In addition, since NL properties can be expressed by quasipolynomial size formulas, it is not unexpected that Frege proofs can quasipolynomially simulate eLNNT, and hence eLDT. These results are given in Section 6.

We include only brief proof sketches in this paper, due to space constraints, but full proofs may be found in [5].

2 Decision tree formulas and LDT proofs

This section describes decision tree (DT) formulas, and the associated sequent calculus proof system LDT. All our proof systems are *propositional* proof systems with variables $x, y, z \dots$ intended to range over the Boolean values *False* and *True*. We use 0 and 1 to denote the constants *False* and *True*, respectively. A *literal* is either a propositional variable x or a negated propositional variable \bar{x} . We use variables p, q, r, \dots to range over literals.

The only connective for forming decision tree formulas (DT formulas) is the 3-ary “case” function, written in infix notation as (ApB) where A and B are formulas and p is required to be a literal. This informally means “if p is false, then A , else B ”. Formally:



■ **Figure 1** Relations between proof systems. \rightarrow means “polynomially simulates”; \rightarrow_{qp} means “quasipolynomially simulates”; \dashrightarrow means “exponentially separated from”. d -LK is the system of dag-like LK proofs with only depth d formulae occurring (atomic formulae have depth 0) By default, all proof systems allow dag-like proofs, unless they are labeled as “Tree”.

► **Definition 1.** Decision tree formulas, or DT formulas, are inductively defined as follows:

1. any literal p is a DT formula, and
2. if A and B are DT formulas and p is a literal, then (ApB) is a DT formula. We call p the decision literal of this formula.

The size of a DT formula A is the number of occurrences of atomic formulas in A .

Suppose α is a 0-1-truth assignment to the variables; the semantics of DT formulas is defined by extending α to be a truth assignment to all DT formulas by inductively defining:

$$\alpha(\bar{x}) = 1 - \alpha(x) \tag{1}$$

$$\alpha(ApB) = \begin{cases} \alpha(A) & \text{if } \alpha(p) = 0 \\ \alpha(B) & \text{otherwise.} \end{cases}$$

It is important that only *literals* p serve as the case distinctions in DT formulas. Notably, for C a complex formula, an expression (ACB) , which evaluates to A if C is true and to B if C is false, would in general denote a decision *diagram* rather than a decision tree.

Although there is no explicit negation of DT formulas, we informally define the negation \bar{A} of a DT formula inductively by letting \bar{x} denote x , and letting \overline{ApB} denote the formula $\bar{A}p\bar{B}$. Of course \bar{A} is a DT formula whenever A is, and \bar{A} correctly expresses the negation of A . Notice also that negative decision literals are “syntactic sugar”, since \overline{ApB} is equivalent to BpA . Nonetheless the notation is useful for making later definitions more intuitive.

Our definition of DT formulas is somewhat different from the usual definition of decision trees. The more common definition would allow 0 and 1 as atomic formulas instead of literals p as in condition 1 of Definition 1. We call such formulas 0/1-DT formulas; they are equivalent to DT formulas in expressive power. The constants 0 and 1 are equivalent to $pp\bar{p}$ and $\bar{p}pp$, for any literal p . More generally, $0pA$, $1pA$, $Ap0$ or $Ap1$ are equivalent to ppA , $\bar{p}pA$, $Ap\bar{p}$, or App , respectively. Conversely, a literal p , when used as atom, is equivalent to $0p1$.

► **Remark 2 (Expressive power of decision trees).** It is easy to decide the validity or satisfiability of a DT formula with a log-space algorithm. To check satisfiability, for example, one examines each leaf in the formula tree (each atomic subformula p) and verifies whether the path of literals from the root to the leaf is consistent with some truth assignment.

A DT formula A of size n can be expressed as a DNF formula of size $O(n^2)$ with at most n disjuncts, defined formally in Section 3. Informally this DNF is formed by taking the disjunction of terms (a.k.a conjunctions of literals) corresponding to paths from the root to a leaf. A dual construction expresses a DT formula A as a CNF formula of size $O(n^2)$ with at most n conjuncts. It is folklore that the construction can be partially reversed: namely any Boolean function that is equivalently expressed by a DNF φ and a CNF ψ can be represented by a DT formula of size quasipolynomial in the sizes of φ and ψ . This bound is optimal, as [23] proves a quasipolynomial lower bound.

We next define the proof system LDT for reasoning about DT formulas. Lines in an LDT proof are sequents, hence they express disjunctions of DT's. Thus lines in LDT proofs can express DNF properties, whose validity problem is non-trivial, indeed coNP-complete.

► **Definition 3.** A cedent, denoted Γ , Δ etc., is a multiset of formulas; we often use commas for multiset union, and write Γ, A for the multiset $\Gamma, \{A\}$. A sequent is an expression $\Gamma \rightarrow \Delta$ where Γ and Δ are cedents. Γ and Δ are called the antecedent and succedent, respectively.

The intended meaning of $\Gamma \rightarrow \Delta$ is that if every formula in Γ is true, then some formula in Δ is true. Accordingly, $\Gamma \rightarrow \Delta$ is true under a truth assignment α iff $\alpha(A) = 0$ for some $A \in \Gamma$ or $\alpha(A) = 1$ for some $A \in \Delta$. A sequent is *valid* iff it is true for every truth assignment.

► **Definition 4.** The sequent calculus LDT is a proof system in which lines are sequents of DT formulas. The valid initial sequents (axioms) are, for p any literal,

$$p \rightarrow p \quad p, \bar{p} \rightarrow \quad \rightarrow p, \bar{p}.$$

The rules of inference are:

$$\text{Contraction rules:} \quad \text{c-l: } \frac{A, A, \Gamma \rightarrow \Delta}{A, \Gamma \rightarrow \Delta} \quad \text{c-r: } \frac{\Gamma \rightarrow \Delta, A, A}{\Gamma \rightarrow \Delta, A}$$

$$\text{Weakening rules:} \quad \text{w-l: } \frac{\Gamma \rightarrow \Delta}{A, \Gamma \rightarrow \Delta} \quad \text{w-r: } \frac{\Gamma \rightarrow \Delta}{\Gamma \rightarrow \Delta, A}$$

$$\text{Cut rule:} \quad \text{cut: } \frac{\Gamma \rightarrow \Delta, A \quad A, \Gamma \rightarrow \Delta}{\Gamma \rightarrow \Delta}$$

$$\text{Decision rules:} \quad \text{dec-l: } \frac{\Gamma, A \rightarrow p, \Delta \quad \Gamma, p, B \rightarrow \Delta}{\Gamma, ApB \rightarrow \Delta}$$

$$\text{dec-r: } \frac{\Gamma \rightarrow A, p, \Delta \quad \Gamma, p \rightarrow B, \Delta}{\Gamma \rightarrow ApB, \Delta}$$

12:6 Proof Systems of Decision Trees and Branching Programs

Proofs are, by default, dag-like. I.e. a proof of a sequent S in LDT is a sequence (S_0, \dots, S_n) such that S is S_n and each S_k is either an initial sequent or is the conclusion of an inference step whose premises occur amongst $(S_i)_{i < k}$. The subsystem where proofs are restricted to be tree-like (i.e. trees of sequents composed by inference steps) is denoted Tree-LDT.

The size of a proof is the sum of the sizes of the formula occurrences in the proof.

The inference rules that are new to LDT are the two decision rules, *dec-l* and *dec-r*. Since ApB is equivalent to $(A \vee p) \wedge (\bar{p} \vee B)$, the lower sequent of a *dec-r* is true (under some fixed truth assignment) iff both upper sequents are true under the same assignment, i.e. the rule is sound and invertible. Similarly, since ApB is also equivalent to $(A \wedge \bar{p}) \vee (p \wedge B)$, the *dec-l* rule is also sound and invertible.

► **Remark 5 (Cut-free completeness).** The invertibility properties also imply that the cut-free fragment of LDT is complete. To prove this by induction on the complexity of sequents, start with a valid sequent $\Gamma \rightarrow \Delta$; choose any non-atomic formula ApB in Γ or Δ , and apply the appropriate decision rule *dec-l* or *dec-r* that introduces this formula. The upper sequents of this inference are also valid and, furthermore, they have logical complexity strictly less than the logical complexity of $\Gamma \rightarrow \Delta$. The base case of the induction is when $\Gamma \rightarrow \Delta$ contains only atomic formulas; in this case, it can be inferred from an initial sequent with weakenings. Note that this shows in fact, that any valid sequent can be proved in LDT using only decision rules, weakenings, and initial sequents. The system also enjoys a “local” cut-elimination procedure, via standard techniques, but that is beyond the scope of this work.

► **Proposition 6.** *The following have polynomial size, cut-free, Tree-LDT proofs:*

- | | | | |
|------------------------------|------------------------------|----------------------------|----------------------------|
| (a) $A \rightarrow A$ | (c) $A, \bar{A} \rightarrow$ | (e) $p, B \rightarrow ApB$ | (g) $ApB, p \rightarrow B$ |
| (b) $\rightarrow A, \bar{A}$ | (d) $A \rightarrow p, ApB$ | (f) $ApB \rightarrow A, p$ | |

3 Comparing DT proof systems and LK proof systems

LK is the usual Gentzen sequent calculus for Boolean formulas over the basis \wedge and \vee . The *Boolean formulas* are defined inductively by

- Any literal p is a Boolean formula, and
- If A and B are Boolean formulas, then so are $(A \vee B)$ and $(A \wedge B)$.

The proof system LK has the same initial sequents (axioms) as LDT, its inference rules are the contraction rules *c-l* and *c-r*, the weakening rules *w-l* and *w-r*, the cut rule, and the following Boolean rules:

$$\wedge\text{-l: } \frac{A, B, \Gamma \rightarrow \Delta}{A \wedge B, \Gamma \rightarrow \Delta} \qquad \wedge\text{-r: } \frac{\Gamma \rightarrow \Delta, A \quad \Gamma \rightarrow \Delta, B}{\Gamma \rightarrow \Delta, A \wedge B}$$

$$\vee\text{-l: } \frac{A, \Gamma \rightarrow \Delta \quad B, \Gamma \rightarrow \Delta}{A \vee B, \Gamma \rightarrow \Delta} \qquad \vee\text{-r: } \frac{\Gamma \rightarrow \Delta, A, B}{\Gamma \rightarrow \Delta, A \vee B}$$

Recall that a *clause* is a disjunction of literals and a *term* is a conjunction of literals.

► **Definition 7.** *A Boolean formula is depth one if it is either a clause or a term. 1-LK is the fragment of LK in which all formulas appearing in sequents are depth one formulas. Tree-1-LK is the same system with the restriction that proofs are tree-like.*

If \vec{p} is a vector of literals, we write $\bigvee \vec{p}$ to denote a disjunction of the literals \vec{p} , taken in the indicated order. The notation $\bigwedge \vec{p}$ is defined similarly. The nesting of disjunctions and conjunctions can be arbitrary, so $\bigvee \vec{p}$ denotes any formula of the form $(\bigvee \vec{p}') \vee (\bigvee \vec{p}'')$ where

\vec{p}' and \vec{p}'' denote p_1, \dots, p_k and p_{k-1}, \dots, p_ℓ for some $1 \leq k \leq \ell$. Although these notations are ambiguous about the nesting of disjunctions or conjunctions, this makes no difference in this work, since if A and B are both of the form $\bigvee \vec{p}$ but with different orders of applications of \vee 's, then there are polynomial size, cut-free Tree-1-LK proofs of $A \rightarrow B$ and $B \rightarrow A$.

Later theorems will compare the proof theoretic strengths of various fragments and extensions of LDT to fragments of LK. Since these theories use different languages, we need to establish translations between cedents of DT formulas and (depth one) Boolean formulas.

► **Definition 8.** For a (nonempty) sequence of literals \vec{p} we define the DT formulas $\text{Conj}(\vec{p})$ and $\text{Disj}(\vec{p})$ by induction on the length of \vec{p} as follows:

$$\begin{aligned} \text{Conj}(p) &:= p & \text{Disj}(p) &:= p \\ \text{Conj}(p, \vec{p}) &:= (pp\text{Conj}(\vec{p})) & \text{Disj}(p, \vec{p}) &:= (\text{Disj}(\vec{p})pp) \end{aligned}$$

In other words, if $\vec{p} = (p_1, \dots, p_\ell)$, for $\ell > 1$, we have:

$$\begin{aligned} \text{Conj}(\vec{p}) &= (p_1 p_1 (p_2 p_2 (\dots (p_{\ell-2} p_{\ell-2} (p_{\ell-1} p_{\ell-1} p_\ell)) \dots))) \\ \text{Disj}(\vec{p}) &= (((\dots ((p_\ell p_{\ell-1} p_{\ell-1}) p_{\ell-2} p_{\ell-2}) \dots) p_2 p_2) p_1 p_1). \end{aligned}$$

It is not hard to verify that Conj and Disj correctly express the conjunction and disjunction of the literals \vec{p} . This is borne out by the next proposition.

► **Proposition 9.** The following sequents have polynomial size, cut-free Tree-LDT proofs.

- | | |
|--|--|
| (a) $\text{Conj}(\vec{p}, \vec{q}) \rightarrow \text{Conj}(\vec{p})$ | (d) $\text{Disj}(\vec{p}) \rightarrow \text{Disj}(\vec{p}, \vec{q})$ |
| (b) $\text{Conj}(\vec{p}, \vec{q}) \rightarrow \text{Conj}(\vec{q})$ | (e) $\text{Disj}(\vec{q}) \rightarrow \text{Disj}(\vec{p}, \vec{q})$ |
| (c) $\text{Conj}(\vec{p}), \text{Conj}(\vec{q}) \rightarrow \text{Conj}(\vec{p}, \vec{q})$ | (f) $\text{Disj}(\vec{p}, \vec{q}) \rightarrow \text{Disj}(\vec{p}), \text{Disj}(\vec{q})$ |

For the converse direction of simulating LDT (and its supersystems) by LK, we need to express DT formulas A as Boolean formulas in both CNF and DNF forms. For this we define $\text{Tms}(A)$ as a multiset of terms (i.e., a multiset of conjunctions) and $\text{Cls}(A)$ as a multiset of clauses (i.e., a multiset of disjunctions) so that A is equivalent to both the DNF $\bigvee \text{Tms}(A)$ and the CNF $\bigwedge \text{Cls}(A)$.

► **Definition 10.** Let A be a DT-formula. The terms and clauses of A are the multisets $\text{Tms}(A)$ and $\text{Cls}(A)$ inductively defined by letting $\text{Tms}(p)$ and $\text{Cls}(p)$ both equal p , and letting

$$\text{Tms}(BpC) := \{(\vec{p} \wedge D) : D \in \text{Tms}(B)\} \cup \{(p \wedge D) : D \in \text{Tms}(C)\} \quad (2)$$

$$\text{Cls}(BpC) := \{(p \vee D) : D \in \text{Cls}(B)\} \cup \{(\vec{p} \vee D) : D \in \text{Cls}(C)\}. \quad (3)$$

For example, if A is $p_1 p_2 (p_3 p_4 p_5)$ then $\text{Tms}(A)$ is $\{\overline{p_2} \wedge p_1, p_2 \wedge \overline{p_4} \wedge p_3, p_2 \wedge p_4 \wedge p_5\}$, and $\text{Cls}(A)$ is equal to $\{p_2 \vee p_1, \overline{p_2} \vee p_4 \vee p_3, \overline{p_2} \vee \overline{p_4} \vee p_5\}$.

The equivalence between A , $\bigvee \text{Tms}(A)$ and $\bigwedge \text{Cls}(A)$ is witnessed by simple proofs:

► **Proposition 11.** There are polynomial size, cut-free Tree-LK-proofs of:

- | |
|--|
| (a) $C \rightarrow D$, for each $C \in \text{Tms}(A)$ and $D \in \text{Cls}(A)$. |
| (b) (i) $\text{Cls}(ApB) \rightarrow D, p$, for each $D \in \text{Cls}(A)$; |
| (ii) $p, \text{Cls}(ApB) \rightarrow D$, for each $D \in \text{Cls}(B)$. |
| (iii) $\text{Cls}(A) \rightarrow D, p$, for each $D \in \text{Cls}(ApB)$. |
| (iv) $p, \text{Cls}(B) \rightarrow D$, for each $D \in \text{Cls}(ApB)$. |
| (c) (i) $C \rightarrow p, \text{Tms}(ApB)$, for each $C \in \text{Tms}(A)$; |
| (ii) $p, C \rightarrow \text{Tms}(ApB)$, for each $C \in \text{Tms}(B)$. |

- (iii) $C \rightarrow p, \text{Tms}(A)$, for each $C \in \text{Tms}(ApB)$.
- (iv) $p, C \rightarrow \text{Tms}(B)$, for each $C \in \text{Tms}(ApB)$.

Proof sketch. Part (a) of the lemma is proved by induction on the complexity of A . Parts (b) and (c) are trivial once the definitions are unwound. For example, (b.i) follows from the fact that $\text{Cls}(ApB)$ contains the formula $p \vee D$. This allows (b.i) to be derived from the two sequents $p \rightarrow p$ and $D \rightarrow D$. The former is an axiom, and the latter has a tree-like cut-free proof by Proposition 6a. The other cases are similar. ◀

The next definition shows how to compare proof complexity between proof systems that work with DT formulas and ones that work with Boolean formulas.

► **Definition 12.** Let P be a proof system for sequents of Boolean formulas (or at least, sequents of depth one Boolean formulas), and Q be a proof system for sequents of DT formulas. We say that P polynomially simulates Q if there is a polynomial time procedure which, given a Q -proof of

$$A_0, \dots, A_{m-1} \rightarrow B_0, \dots, B_{n-1}, \quad (4)$$

where the A_i 's and B_i 's are DT-formulas, produces a P -proof of

$$\text{Cls}(A_0), \dots, \text{Cls}(A_{m-1}) \rightarrow \text{Tms}(B_0), \dots, \text{Tms}(B_{n-1}). \quad (5)$$

The system Q polynomially simulates P if there is a polynomial time procedure which, given a P -proof of

$$\bigvee \vec{a}_0, \dots, \bigvee \vec{a}_{m-1} \rightarrow \bigwedge \vec{b}_0, \dots, \bigwedge \vec{b}_{n-1}, \quad (6)$$

where the \vec{a}_i 's and \vec{b}_i 's are sequences of literals, produces a Q -proof of

$$\text{Disj}(\vec{a}_0), \dots, \text{Disj}(\vec{a}_{m-1}) \rightarrow \text{Conj}(\vec{b}_0), \dots, \text{Conj}(\vec{b}_{n-1}). \quad (7)$$

The systems P and Q are polynomially equivalent if they polynomially simulate each other. (5) is called the Boolean translation of (4). (7) is called the DT-translation of (6). Quasipolynomial simulation and equivalence are defined in the same way, but using quasipolynomial time (time $2^{\log^{O(1)} n}$) procedures.²

3.1 1-LK and LDT

Our first results compare the weakest systems considered in this work, operating with just DT formulas or with just terms and clauses.

► **Theorem 13.** LDT polynomially simulates 1-LK. Tree-LDT polynomially simulates Tree-1-LK.

Proof sketch. We may replace terms $\bigwedge \vec{a}$ and clauses $\bigvee \vec{a}$ occurring in a 1-LK proof by DT-formulas $\text{Conj}(\vec{a})$ or $\text{Disj}(\vec{a})$ respectively. The result can be adapted into a correct LDT proof using cuts against proofs from Proposition 9. ◀

² It turns out that all stated quasipolynomial simulations in this work (Theorems 14 and 30) take time $n^{O(\log n)} = 2^{O(\log^2 n)}$.

A converse result holds too, but we have only a quasipolynomial simulation in the tree-like case. It is open whether this can be improved to a polynomial simulation.

► **Theorem 14.** *1-LK polynomially simulates LDT. Tree-1-LK quasipolynomially simulates Tree-LDT.*

Proof sketch. In a given LDT proof, we may replace every DT A in an antecedent by the multiset $\text{Cls}(A)$ and every DT A in a succedent by $\text{Tms}(A)$. The result can be adapted into a correct 1-LK proof using cuts against proofs of the truth conditions from Proposition 11.

In the tree-like case, when simulating the cut rule we must copy one subproof polynomially many times (such copying is unnecessary when proofs are dag-like). However it turns out we may freely choose which of the two subproofs to duplicate, so we may just take the smaller one, which has size at most half that of the original proof. Doing this recursively yields a $n^{O(\log n)} = 2^{O(\log^2 n)}$ bound on the size of the resulting Tree-1-LK proof. ◀

4 Nondeterministic decision tree formulas and LNDDT proofs

This section defines nondeterministic decision tree (NDT) formulas, and the associated sequent calculus LNDDT. The NDT formulas have two kinds of connectives; the 3-ary case function ApB and the Boolean OR-gate (\vee). Formally:

► **Definition 15.** *The nondeterministic decision tree formulas, or NDT formulas for short, are inductively defined by*

- Any literal p is a NDT formula;
- If A and B are NDT formulas and p is a literal, then (ApB) is a NDT formula;
- If A and B are NDT formulas, then $(A \vee B)$ is an NDT formula.

A nondeterministic gate in a decision tree accepts just when at least one of its children is accepting. This corresponds exactly to an \vee gate, which yields *True* exactly when at least one input is *True*. One of our motivations in defining LNDDT that is will serve as a foundation for our later definition eLNDDT, which will capture a logic for nondeterministic branching programs, and hence a logic for nonuniform NL.

► **Definition 16.** *The sequent calculus LNDDT is a proof system in which lines are sequents of NDT formulas. Its initial sequents (axioms) and rules are the same as those of LDT, along with the two \vee inferences, \vee -l and \vee -r, of LK as described on page 6.*

For α a 0-1-truth assignment, the semantics of NDT formulas is defined extending the definition of the semantics of DT formulas, in equations 1, to include

$$\alpha(A \vee B) = \begin{cases} 1 & \text{if } \alpha(A) = 1 \text{ or } \alpha(B) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

It is straightforward to verify that LNDDT is sound and complete for sequents of NDT formulas, by a similar argument to that of Remark 5.

4.1 LDT and tree-like LNDDT are equivalent

Next we turn to the relative complexity of LDT and LNDDT. Naturally the latter subsumes the former, but this can be strengthened as follows:

► **Theorem 17.** *Tree-LNDDT is polynomially equivalent to LDT over DT-sequents.*

12:10 Proof Systems of Decision Trees and Branching Programs

We will soon see that this also refines the known polynomial equivalence between 1-LK and Tree-2-LK (see [2, 3]), by virtue of Theorems 14 and 18.

Proof sketch. To show that Tree-LNDDT polynomially simulates LDT we notice that lines of an LDT proof (i.e. sequents of DT formulas) may be expressed as NDT formulas. From here one may use an adaptation of a standard technique for showing that tree-like LK is equivalent to dag-like LK, carefully managing the complexity of formulas occurring.

To show that LDT polynomially simulates Tree-LNDDT, we first notice that each NDT formula may be written as a disjunction of DT formulas (“normal form”), and furthermore that LNDDT proofs may be written in a way that operates with only such formulas with only polynomial blowup. Now we convert a normal form Tree-LNDDT proof π of $\bigvee \Pi_1, \dots, \bigvee \Pi_k \rightarrow \bigvee \Lambda_1, \dots, \bigvee \Lambda_l$ to a (dag-like) LDT derivation π' of the sequent $\rightarrow \Lambda_1, \dots, \Lambda_l$ from extra *hypotheses* $\{\rightarrow \Pi_i\}_{i=1}^k$. This is proved by induction on the structure of the proof tree and takes polynomial time. Now, when π derives a DT sequent, notice that π' is just a LDT proof of the same sequent. ◀

4.2 Equivalence of LNDDT and 2-LK

A Boolean formula is *depth two* if it is depth one, or if it is a conjunction of clauses or a disjunction of terms. 2-LK is the fragment of LK in which all formulas occurring are depth two formulas. Tree-2-LK is the same system with the restriction that proofs are tree-like.

► **Theorem 18.** LNDDT and 2-LK are polynomially equivalent. Tree-LNDDT and Tree-2-LK are polynomially equivalent.

This is not so surprising a result, since NDTs have equivalent expressive power to DNFs, so depth two sequents may be written as NDT sequents and vice-versa.

Proof sketch. A (two-sided) 2-LK proof is simulated in LNDDT by simply replacing every DNF $\bigvee_i \bigwedge \vec{p}_i$ with the NDT $\bigvee_i \text{Conj}(\vec{p}_i)$ and locally repairing the proof using cuts against proofs from Proposition 8. In the other direction we work with “normal form” LNDDT proofs (as in the proof of Theorem 17). From here the translation to DNFs is straightforward, since DT formulas already have small DNFs, cf. Definition 10. Again, we use cuts against proofs of the appropriate truth conditions. Both simulations map tree-like proofs to tree-like proofs. ◀

5 Proof systems for branching programs

5.1 Formulas and proofs with extension variables

We now describe the systems eLDT and eLNDDT which reason about deterministic and nondeterministic branching programs respectively.³ Formulas can now include *extension variables*, usually denoted by e_1, e_2 , etc. It is important that the extension variables are explicitly distinguished from the propositional variables we have thus far used.

The purpose of extension variables is to serve as abbreviations for more complex formulas. Thus, proofs that use extension variables will be accompanied by a set of extension axioms $\{e_i \leftrightarrow A_i\}_{i < n}$, where each formula A_i may use any literals p but is restricted to use only the extension variables e_j for $j < i$. The intent is that e_i is an abbreviation for the formula A_i .

³ These systems could equally well be called LBP and LNBP, using “BP” for “branching programs”.

► **Definition 19.** Extended decision tree formulas (eDT formulas) are defined as follows:

- (1) Any literal p is an eDT formula.
- (2) Any extension variable e is an eDT formula.
- (3) If A and B are eDT formulas and p is a literal, then (ApB) is a DT formula.

In particular, a decision literal p in a formula ApB is *not* allowed to be an extension variable. The intuition is that the extension variables may “name” nodes in a branching program.

► **Definition 20.** Extended nondeterministic decision tree formulas (eNDT formulas) are defined by the closure conditions (1)-(3) above (replacing “eDT” by “eNDT”) and:

- (4) If A and B are eNDT formulas, then $(A \vee B)$ is an eNDT formula.

A set of *extension axioms* is a set $\mathcal{A} = \{e_i \leftrightarrow A_i\}_{i < n}$ where e_0, \dots, e_{n-1} are extension variables such that the only extension variables appearing in A_i are e_0, \dots, e_{i-1} , for $i < n$. We identify \mathcal{A} with the set of sequents consisting of $e_i \rightarrow A_i$ and $A_i \rightarrow e_i$, for $i < n$. eDT and eNDT formulas have truth semantics only relative to a set of extension axioms $\{e_i \leftrightarrow A_i\}_{i < n}$. Namely, for α a truth assignment, the definition of truth is extended by setting $\alpha(e_i) = \alpha(A_i)$.

► **Definition 21.** An eLDT proof is a pair (π, \mathcal{A}) where $\mathcal{A} = \{e_i \leftrightarrow A_i\}_{i < n}$ is a set of extension axioms where each A_i is an eDT formula, and π is an LDT derivation which is allowed to use initial sequents from \mathcal{A} . eLNDT proofs are defined similarly, but with eLNDT formulas A_i and eLNDT derivations.

Note that all formulas in an eLDT or eLNDT proof are based on a single set of extension axioms $\{e_i \leftrightarrow A_i\}_{i < n}$.

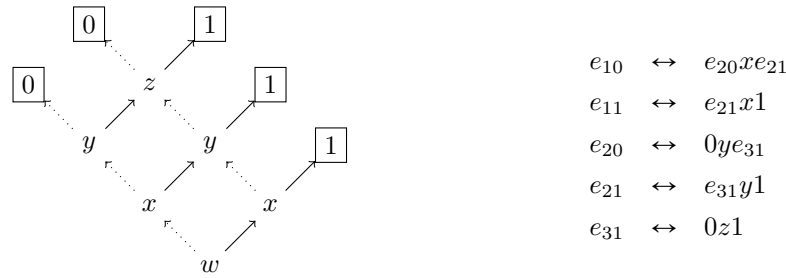
Let us discuss how the extended formulas we have introduced may be used to represent bona fide branching programs. A (deterministic) branching program is a directed acyclic graph G such that (a) G has a unique source node, (b) sink nodes in G are labelled with either 0 or 1, (c) all other nodes are labelled with a literal p and have two outgoing edges, one labelled 0 and the other 1. G can be converted into an equivalent eDT formula with associated extension axioms $\{e_i \leftrightarrow A_i\}_{i < n}$ by introducing an extension variable for every internal node in G . Conversely, as is described in more detail in Section 5.2, any eDT formula A with extension axioms $\{e_i \leftrightarrow A_i\}_{i < n}$ can be straightforwardly transformed into a linear size deterministic branching program. For this, the nodes in the branching program correspond to the extension variables e_i and the subformulas of the formulas A_i .

Nondeterministic branching programs are defined similarly to deterministic branching programs, but further allowing the internal nodes of G to be labelled with “ \vee ” as well as literals (in this case the labelling of its outgoing edges is omitted). The semantics is that an \vee -node is accepting provided at least one of its children is accepting. It is straightforward to convert a nondeterministic branching program into an eLNDT formula with associated extension axioms, and vice versa. A similar construction yields the folklore fact that “extended Boolean formulas” are as expressive as Boolean circuits.

► **Example 22.** Consider the (deterministic) branching program G in Figure 2, on the left, which returns 1 just if at least two out of the four input variables w, x, y, z are 1. Edges labelled with 0 are here dotted (and always left outgoing) while edges labelled 1 are here solid (and always right outgoing). In this particular case, the branching program is *ordered* (or an *OBDD*), i.e. variables occur in the same relative order on each path from the source to a sink. The program also happens to compute a monotone Boolean function.

To represent G in eLDT, we introduce extension variables for each internal node of the program as follows. Write e_{ij} for the j th node of the i th layer, with i, j ranging from 0 onward, and introduce the extension axioms in Figure 2, on the right.⁴ Now G is represented

⁴ Formally, we are writing 0 and 1 as shorthand for $pp\bar{p}$ and $\bar{p}pp$ respectively, for some/any literal p .



■ **Figure 2** A branching program G , on the left, computing the 2-out-of-4 threshold function and an encoding of its (internal) local conditions by extension variables, on the right. Dotted edges are labelled 0 and solid edges are labelled 1. G is equivalent to the eDT formula $e_{10}we_{11}$.

by the eDT formula $e_{10}we_{11}$. Notice that the orderedness of the program is reflected in its eLDT representation: writing (x_0, x_1, x_2, x_3) for (w, x, y, z) , we have that x_i is the root of the formula that any e_{ij} abbreviates.

Other representations of G are possible, for instance by renaming the extension variables or by partially unwinding the graph. In both these two latter cases, the eDT representation obtained will be provably equivalent to the one above, by polynomial-size proofs in eLDT, by virtue of Lemma 28 later.

5.2 Foundational issues and Boolean combinations

The fact that extension variables cannot be used as decision literals is a significant limitation on the expressiveness of DT formulas. Recall for instance that the conjunction of p_1 and p_2 can be expressed with the DT formula $\text{Conj}(p_1, p_2)$, namely $(p_1p_1p_2)$. However, it is not permitted to form $(e_1e_1e_2)$; in fact, it is not possible to express the conjunction $e_1 \wedge e_2$ without taking the extension axioms defining e_1 and e_2 into account. If we could write the conjunction of e_1 and e_2 by a generic formula $A(e_1, e_1)$, then we could introduce a new extension variable representing $A(e_1, e_2)$. This would imply that eDT formulas are as expressive as extended Boolean formulas; in other words, that deterministic branching programs would be as expressive as Boolean circuits. This is a non-uniform analogue of $L = P$ (i.e., log-space equals polynomial time) and, of course, is an open question.

Nonetheless, for any given extension variables e and e' , there is a formula $\text{AND}(e, e')$ expressing the conjunction of e and e' by changing the underlying set of extension axioms. The intuition is that we start with the branching program G for e , but now with sink nodes labelled with 0 or 1 instead of with variables. To form the branching program for $e \wedge e'$, we take (an isomorphic copy) of the branching program G' for e' , and modify G by replacing each sink node labelled with 1 with the source node of G' (in other words, each edge directed into a sink “1” is modified to instead point to the root of G'). Since we do not actually have 0 and 1 in the language, we work modulo their encodings by literals:

► **Definition 23.** Let C be an eDT or eNNT formula. $C[0/B]$ is the formula obtained by replacing (in parallel) each occurrence of a literal p as a leaf in C with the formula (Bpp) . Similarly, $C[1/B]$ is the formula obtained by replacing each occurrence of a literal p as a leaf in C with the formula (ppB) .

The point of $C[0/B]$ is that (Bpp) evaluates to 1 if p is true, and to B otherwise. Thus, the intent is that $C[0/B]$ is equivalent $C \vee B$. Likewise, we want $C[1/B]$ to be equivalent

$C \wedge B$. However, these equivalences hold only if the substitutions are applied not just in C but instead throughout the definitions of the extension axioms used in C . This is done with the following definition.

► **Definition 24.** Let \mathcal{A} be a set of extension axioms $\{e_i \leftrightarrow A_i\}_{i < n}$. Another set of extension axioms $\mathcal{A}[1/B]$ is defined as follows. First, let $\{e'_i\}_i$ be a set of new extension variables. Define $A_i[\bar{e}'/\bar{e}]$ to be the result of replacing each e_j in A_i with e'_j . Let A'_i be $(A_i[\bar{e}'/\bar{e}])[1/B]$. Then $\mathcal{A}[1/B]$ is the set of extension axioms $\{e'_i \leftrightarrow A'_i\}_{i < n} \cup \mathcal{A}$. The set $\mathcal{A}[0/B]$ is defined similarly: letting \bar{e}'' be another set of new extension variables, defining A''_i to be $(A_i[\bar{e}''/\bar{e}])[0/B]$, and letting $\mathcal{A}[0/B]$ be the set of extension axioms $\{e''_i \leftrightarrow A''_i\}_{i < n} \cup \mathcal{A}$.

Finally, if A and B are eDT or eNDT formulas defined using extension axioms \mathcal{A} , then $\text{AND}(A, B)$ is by definition $A[1/B]$ relative to the extension axioms $\mathcal{A}[1/B]$. The formula $\text{OR}(A, B)$ for disjunction is defined similarly, namely, it is equal to $A[0/B]$ relative to the extension axioms $\mathcal{A}[0/B]$.

Note the two formulas $\text{AND}(A, B)$ and $\text{OR}(A, B)$ introduced *different* sets of new extension variables, so we may use both $\text{AND}(A, B)$ and $\text{OR}(A, B)$ without any clashes between extension variables. More generally, we adopt the convention that the new extension variables are uniquely determined by the Boolean combination being constructed. For instance, e'_i could have instead been designated $e_{i, (A \wedge B)}$. When measuring proof size, we also need to count the sizes of the subscripts on the extension variables. This clearly however only increases proof size polynomially.

There are two other sources of growth of size in forming $\text{AND}(A, B)$ and $\text{OR}(A, B)$. The first is that formula sizes increase since copies of B is substituted in at many places in A and \mathcal{A} : this potentially gives a quadratic blowup in proof size. We avoid this quadratic blowup in proof size, by always taking B to be a single variable (namely, an extension variable). The construction of $\text{AND}(A, B)$ or $\text{OR}(A, B)$ also introduces many new extension variables, namely it potentially doubles the number of variables. To control this, we will ensure that the constructions of $\text{AND}(\cdot, \cdot)$ and $\text{OR}(\cdot, \cdot)$ are nested only logarithmically.

► **Example 25.** Consider the formula $\text{AND}(p_1, \text{AND}(p_2, p_3))$, which is a translation of the Boolean formula $p_1 \wedge (p_2 \wedge p_3)$ to a DT formula. To form $\text{AND}(p_2, p_3)$, start with $(p_2 p_2 1)$ and substitute p_3 for “1”, to obtain $(p_2 p_2 p_3)$. Then $\text{AND}(p_1, \text{AND}(p_2, p_3))$ is obtained by forming $(p_1 p_1 1)$ and replacing “1” with $\text{AND}(p_2, p_3)$ to obtain $(p_1 p_1 (p_2 p_2 p_3))$. It is also the same as $\text{Conj}(p_1, p_2, p_3)$. A similar construction shows that $\text{OR}(p_1, \text{OR}(p_2, p_3))$ is equal to $((p_3 p_2 p_2) p_1 p_1)$. This is a translation of the Boolean formula $p_1 \vee (p_2 \vee p_3)$ to a DT formula, and is equal to $\text{Disj}(p_1, p_2, p_3)$.

► **Example 26.** Let A be the formula $(p_1 p_2 (e_1 p_3 e_2))$ and B be the formula $(q_1 q_2 e_2)$ in the context of the extension axioms \mathcal{A}

$$e_1 \leftrightarrow (r_1 \bar{r}_2 e_2) \quad e_2 \leftrightarrow (\bar{s}_1 s_2 s_3), \quad (8)$$

where p_i, q_i, r_i, s_i are literals. The formula $A[0/B]$ is formed as follows. First $\mathcal{A}[\bar{e}'/\bar{e}]$ is $e'_1 \leftrightarrow (r_1 \bar{r}_2 e'_2)$, $e'_2 \leftrightarrow (\bar{s}_1 s_2 s_3)$. Then $\mathcal{A}[0/B]$ contains the extension axioms of \mathcal{A} as shown in (8) plus the extension axioms $e'_1 \leftrightarrow ((B r_1 r_1) \bar{r}_2 e'_2)$, $e'_2 \leftrightarrow ((B \bar{s}_1 \bar{s}_1) s_2 (B s_3 s_3))$. Finally, $A[0/B]$ is the DT formula $((B p_1 p_1) p_2 (e'_1 p e'_2))$, namely, $((q_1 q_2 e_2) p_1 p_1) p_2 (e'_1 p e'_2)$, relative to the four extension axioms in $\mathcal{A}[0/B]$.

5.3 Truth conditions and renaming of extension variables

We show that, despite the delicate renaming of variables required for notions such as $A[0/B]$ and $\text{AND}(A, B)$, for DT (respectively eNDT) formulas A, B , we may nonetheless realise their basic truth conditions by small eLDT (respectively eLNDT) proofs:

12:14 Proof Systems of Decision Trees and Branching Programs

► **Lemma 27.** *Let A and B be eDT formulas (respectively, eNDT formulas) relative to extension axioms \mathcal{A} . Then, the sequents (a)-(c) below have polynomial size, cut free eLDT proofs (respectively, eLNNT proofs) relative to the extension axioms $\mathcal{A}[0/B]$. The same holds for the sequents (d)-(f) relative to $\mathcal{A}[1/B]$.*

$$\begin{array}{lll} \text{(a)} & B \rightarrow A[0/B] & \text{(c)} & A[0/B] \rightarrow A, B & \text{(e)} & A[1/B] \rightarrow A \\ \text{(b)} & A \rightarrow A[0/B] & \text{(d)} & A[1/B] \rightarrow B & \text{(f)} & A, B \rightarrow A[1/B] \end{array}$$

Proof sketch. Parts (a)-(c) are proved by showing inductively that if C is a subformula of $A[0/B]$ or a subformula of any A'_i in $\mathcal{A}[0/B]$, then $C \rightarrow A, B$ and $B \rightarrow C$ and $A \rightarrow C$ have short eLDT (resp., eLNNT) proofs. The base cases are just the cases where C is the form (Bpp) . The inductive cases are trivial. A similar argument proves cases (d)-(f). ◀

The proofs of Lemma 27 seem to be inherently dag-like, and we do not know if there are polynomial-size Tree-eLDT proofs for those sequents.

As discussed above, we assume that the choice of new extension variables \vec{e}' or \vec{e}'' depends explicitly on what formula $\text{AND}(A, B)$ and $\text{OR}(A, B)$ is being formed. In other words, each e'_i or e''_i is a variable $e_{i, \text{AND}(A, B)}$ or $e_{i, \text{OR}(A, B)}$. In the proof of Theorem 29 later, this means that the translations of distinct occurrences of the same Boolean formula use the same extension variables. However, this is not strictly necessary, as eLDT can prove the equivalence of formulas after renaming extension variables:

► **Lemma 28.** *Suppose A is a DT formula w.r.t. extension axioms $\mathcal{A} = \{e_i \leftrightarrow A_i\}_i$, and that the extension variables \vec{f} are distinct from the extension variables \vec{e} . Let B equal $A[\vec{f}/\vec{e}]$ w.r.t. the extension axioms $\mathcal{B} = \{f_i \leftrightarrow A_i[\vec{f}/\vec{e}]\}_i$. Then eLDT has a polynomial size, cut free (dag-like) proofs of $A \rightarrow B$ and $B \rightarrow A$ relative to the extension axioms $\mathcal{A} \cup \mathcal{B}$.*

Lemma 28 has a straightforward proof that proceeds inductively through all subformulas of the formulas A_i and A .

6 Simulations for eLDT, eLNNT and LK

We compare the systems eLDT and eLNNT with LK, showing that they are all quasi-polynomially related in terms of proof size, constituting the upper half of Figure 1.

6.1 eLDT polynomially simulates LK

The intuition for the next simulation is that the formulas in an LK proof are Boolean and may be evaluated in log-space. Thus they may be expressed by polynomial-size eDT formulas (under appropriate extension axioms).

► **Theorem 29.** *eLDT (and so also eLNNT) polynomially simulates LK.*

Proof sketch. We assume the given LK proof is written in *balanced* form, i.e. with only $O(\log n)$ -depth Boolean formulas occurring. Once again we proceed by replacing each formula occurrence by an eDT formula representing it, by virtue of the constructions of AND and OR from Definition 24. (We appeal to the logarithmic depth of Boolean formula occurrences in order to control the complexity of this translation). From here we locally simulate each step of the LK proof by cutting against the truth conditions from Lemma 27. ◀

6.2 LK quasipolynomially simulates eLNDT

The intuition for the next simulation is that eNDT formulas define nondeterministic logspace properties, and these are expressible with quasipolynomial size Boolean formulas.

► **Theorem 30.** *LK quasipolynomially simulates eLNDT (and so also eLDT).*

Proof sketch. We work from the observation that NL predicates have quasipolynomial-size (in fact $n^{O(\log n)}$ -size) Boolean formulas. Moreover, there is an *evaluator* for non-deterministic branching programs with quasipolynomial-size Boolean formulas for *st*-connectivity in graphs, whose basic properties were shown to have quasipolynomial-size LK proofs in [4]. Once the basic truth conditions of this evaluator are given appropriate LK proofs, we may proceed by duly replacing every eNDT formula occurrence in an eLNDT proof π by the corresponding Boolean formula evaluating the non-deterministic branching program it represents. We cut against proofs of the truth conditions to locally simulate each step of π . ◀

7 Conclusions

We presented sequent-style systems LDT, LNDT, eLDT and eLNDT that manipulate decision trees, nondeterministic decision trees, branching programs (via extension) and nondeterministic Branching Programs (via extension) respectively. The systems eLDT and eLNDT serve as natural systems for log-space and nondeterministic log-space reasoning, respectively. We examined their relative proof complexity and also compared them to (low depth) Frege systems (more precisely their representations in the sequent calculus LK).

We did not compare the proof complexity theoretic strength of our systems eLDT and eLNDT with the systems GL^* for L and GNL^* for NL in [31, 32]. In future work we intend to show that our systems correspond to the bounded arithmetic theories VL and VNL in the usual way. Namely, proofs of Π_1 formulas in VL translate to families of small eLDT proofs of each instance, and, conversely, VL proves the soundness of eLDT. (Similarly for VNL and eLNDT.) This would render our systems polynomially equivalent to GL^* and GNL^* , respectively, by the analogous results from [31, 32], though this remains work in progress.

Two natural open questions arise from this work.

► **Question 31.** *Does Tree-1-LK polynomially simulate Tree-LDT, or is there a quasipolynomial separation between the two?*

► **Question 32.** *Does Tree-eLDT polynomially simulate eLDT? Similarly for eLNDT.*

While well-defined, the systems Tree-eLDT and Tree-eLNDT do not seem very robust, in the sense that it is not immediate how to witness branching program isomorphisms with short proofs. Nonetheless, it would be good to settle their proof complexity theoretic status.

There has been much recent work on the proof complexity of systems that may manipulate OBDDs [24, 6, 20], branching programs where propositional variables must occur in the same relative order on each path through the dag. In fact, we could also define an “OBDD fragment” of eLDT by restricting lines to eDT formulas expressing OBDDs, as alluded to in Example 22. It would be interesting to examine such systems from the point of view of proof complexity in the future, in particular comparing them to existing OBDD systems.

References


- 1 Toshiyasu Arai. A Bounded Arithmetic AID for Frege Systems. *Annals of Pure and Applied Logic*, 103:155–199, 2000.
- 2 Arnold Beckmann and Samuel R. Buss. Separation Results for the Size of Constant-Depth Propositional Proofs. *Annals of Pure and Applied Logic*, 136:30–55, 2005.
- 3 Arnold Beckmann and Samuel R. Buss. On Transformations of Constant Depth Propositional Proofs. *Annals of Pure and Applied Logic*, ??:??–???, 2019. To appear. doi:10.1016/j.apal.2019.05.002.
- 4 Sam Buss. Quasipolynomial Size Proofs of the Propositional Pigeonhole Principle. *Theoretical Computer Science*, 576(C):77–84, 2015. doi:10.1016/j.tcs.2015.02.005.
- 5 Sam Buss, Anupam Das, and Alexander Knop. Proof complexity of systems of (non-deterministic) decision trees and branching programs, 2019. arXiv:1910.08503.
- 6 Sam Buss, Dmitry Itsykson, Alexander Knop, and Dmitry Sokolov. Reordering Rule Makes OBDD Proof Systems Stronger. In *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, pages 16:1–16:24, 2018. doi:10.4230/LIPIcs.CCC.2018.16.
- 7 Samuel R. Buss. *Bounded Arithmetic*. Bibliopolis, Naples, Italy, 1986. Revision of 1985 Princeton University Ph.D. thesis.
- 8 Samuel R. Buss. The Boolean Formula Value Problem is in ALOGTIME. In *Proceedings of the 19-th Annual ACM Symposium on Theory of Computing*, pages 123–131, May 1987.
- 9 Samuel R. Buss and Pavel Pudlák. How to Lie Without Being (Easily) Convicted and the Lengths of Proofs in Propositional Calculus. In L. Pacholski and J. Tiuryn, editors, *Proceedings of the 8th Workshop on Computer Science Logic, Kazimierz, Poland, September 1994*, Lecture Notes in Computer Science #933, pages 151–162, Berlin, 1995. Springer-Verlag.
- 10 Stephen A. Cook. A Survey of Complexity Classes and Their Associated Propositional Proof Systems and Theories, and a Proof System for Log Space. Talk presented at the ICMS Workshop on Circuit and Proof Complexity, Edinburgh, October 2001. <http://www.cs.toronto.edu/sa-cook/>.
- 11 Stephen A. Cook. Feasibly Constructive Proofs and the Propositional Calculus. In *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*, pages 83–97. Association for Computing Machinery, 1975.
- 12 Stephen A. Cook and Antonina Kolokolova. A Second-Order System for Polytime Reasoning based on Grädel’s Theorem. *Annals of Pure and Applied Logic*, 124:193–231, 2003.
- 13 Stephen A. Cook and Antonina Kolokolova. A Second-Order Theory for NL. In *Proc. 19th IEEE Symp. on Logic in Computer Science (LICS’04)*, pages 398–407, 2004.
- 14 Stephen A. Cook and Tsuyoshi Morioka. Quantified Propositional Calculus and A Second-Order Theory for NC¹. *Archive for Mathematical Logic*, 44:711–749, 2005.
- 15 Stephen A. Cook and Phuong Nguyen. *Foundations of Proof Complexity: Bounded Arithmetic and Propositional Translations*. ASL and Cambridge University Press, 2010. 496 pages.
- 16 Stephen A. Cook and Robert A. Reckhow. On the Lengths of Proofs in the Propositional Calculus, Preliminary Version. In *Proceedings of the Sixth Annual ACM Symposium on the Theory of Computing*, pages 135–148, 1974.
- 17 Stephen A. Cook and Robert A. Reckhow. The Relative Efficiency of Propositional Proof Systems. *Journal of Symbolic Logic*, 44:36–50, 1979.
- 18 Martin Dowd. Propositional Representation of Arithmetic Proofs. In *Proceedings of the 10th ACM Symposium on Theory of Computing (STOC)*, pages 246–252, 1978. doi:10.1145/800133.804354.
- 19 Erich Grädel. Capturing Complexity Classes by Fragments of Second Order Logic. *Theoretical Computer Science*, 101:35–57, 1992.
- 20 Dmitry Itsykson, Alexander Knop, Andrei E. Romashchenko, and Dmitry Sokolov. On OBDD-Based Algorithms and Proof Systems That Dynamically Change Order of Variables. In *34th*

- Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, pages 43:1–43:14, 2017. doi:10.4230/LIPIcs.STACS.2017.43.
- 21 Emil Jeřábek. Dual Weak Pigeonhole Principle, Boolean Complexity, and Derandomization. *Annals of Pure and Applied Logic*, 124:1–37, 2004. doi:10.1016/j.apal.2003.12.003.
 - 22 Jan Johannsen. Satisfiability Problem Complete for Deterministic Logarithmic Space. In *Proc. 21st Symp. on Theoretical Aspects of Computer Science (STACS)*, Lecture Notes in Computer Science 2996, pages 317–325. Springer, 2004.
 - 23 Stasys Jukna, Alexander A. Razborov, Petr Savický, and Ingo Wegener. On P versus $NP \cap co-NP$ for Decision Trees and Read-Once Branching Programs. *Computational Complexity*, 8(4):357–370, 1999. doi:10.1007/s000370050005.
 - 24 Alexander Knop. IPS-like proof systems based on binary decision diagrams. Typeset manuscript, June 2017.
 - 25 Jan Krajíček. *Bounded Arithmetic, Propositional Calculus and Complexity Theory*. Cambridge University Press, Heidelberg, 1995.
 - 26 Jan Krajíček. *Proof Complexity*. Cambridge University Press, 2019.
 - 27 Jan Krajíček and Pavel Pudlák. Quantified Propositional Calculi and Fragments of Bounded Arithmetic. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 36:29–46, 1990.
 - 28 Jan Krajíček and Gaisi Takeuti. On Induction-Free Provability. *Annals of Mathematics and Artificial Intelligence*, pages 107–126, 1992.
 - 29 Richard E. Ladner. The Circuit Value Problem is Log Space Complete for P. *SIGACT News*, 7:18–20, 1975.
 - 30 Jeff B. Paris and Alex J. Wilkie. Counting Problems in Bounded Arithmetic. In *Methods in Mathematical Logic, Lecture Notes in Mathematics #1130*, pages 317–340. Springer-Verlag, 1985.
 - 31 Steven Perron. A Propositional Proof System for Log Space. In *Proc. 14th Annual Conf. Computer Science Logic (CSL)*, Springer Verlag Lecture Notes in Computer Science 3634, pages 509–524, 2005.
 - 32 Steven Perron. *Power of Non-Uniformity in Proof Complexity*. PhD thesis, Department of Computer Science, University of Toronto, 2009.
 - 33 G. S. Tsejtin. On the Complexity of Derivation in Propositional Logic. *Studies in Constructive Mathematics and Mathematical Logic*, 2:115–125, 1968.
 - 34 Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000. URL: <http://ls2-www.cs.uni-dortmund.de/monographs/bdd/>.

Internal Parametricity for Cubical Type Theory

Evan Cavallo 

Carnegie Mellon University, Pittsburgh, PA, USA
ecavallo@cs.cmu.edu

Robert Harper 

Carnegie Mellon University, Pittsburgh, PA, USA
rwh@cs.cmu.edu

Abstract

We define a computational type theory combining the contentful equality structure of cartesian cubical type theory with internal parametricity primitives. The combined theory supports both univalence and its relational equivalent, which we call *relativity*. We demonstrate the use of the theory by analyzing polymorphic functions between higher inductive types, and we give an account of the identity extension lemma for internal parametricity.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases parametricity, cubical type theory, higher inductive types

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.13

Related Version <https://arxiv.org/abs/1901.00489>

Funding We gratefully acknowledge the support of the Air Force Office of Scientific Research through MURI grant FA9550-15-1-0053. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the AFOSR.

Acknowledgements We thank Carlo Angiuli, Steve Awodey, Daniel Gratzer, Kuen-Bang Hou (Favonia), Dan Licata, Anders Mörtberg, Emily Riehl, Christian Sattler, and Jonathan Sterling for their comments and insights.

1 Introduction

Cubical type theory [17, 3, 2] is a recent extension of type theory with *contentful equality* (or *paths*). The central concept of cubical type theory, inherited from homotopy type theory [30], is that terms can be equal in multiple ways, each of which is a method of translating results between them. The motivating example of contentful equality is *structured isomorphism*. In informal mathematical practice, it is common to treat isomorphic objects as if they were “the same,” because any interesting property of one will also hold of the other. Cubical type theory makes this informal practice formal: equality of types *is* isomorphism, which is to say that we have Voevodsky’s *univalence axiom* [32]. For this to be possible, it is essential that equality be contentful, because two objects can be isomorphic in many ways. The key feature of cubical type theory, in comparison with homotopy type theory, is that it is a programming language, not just a logical formalism; in particular, uses of contentful equality *compute*.

A user of cubical type theory can also define types with custom equality structure using *higher inductive types* (HITs) [18, 14]. A simultaneous generalization of inductive types and quotient types, a HIT is freely generated by a collection of constructors, each of which may introduce not only elements but also *paths between elements*. For example, the following specification defines a type $\mathbb{Z}/2\mathbb{Z}$ of integers modulo 2 from a type \mathbb{Z} of integers.

```
data  $\mathbb{Z}/2\mathbb{Z} : \mathcal{U}$  where
| in( $n : \mathbb{Z}$ ) :  $\mathbb{Z}/2\mathbb{Z}$ 
| mod( $n : \mathbb{Z}, x : \mathbb{I}$ ) :  $\mathbb{Z}/2\mathbb{Z}$  [ $x = 0 \leftrightarrow \text{in}(n) \mid x = 1 \leftrightarrow \text{in}(n + 2)$ ]
```



© Evan Cavallo and Robert Harper;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 13; pp. 13:1–13:17



Leibniz International Proceedings in Informatics

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

This type has two constructors, `in` and `mod`. The first introduces an element of $\mathbb{Z}/2\mathbb{Z}$ for each element of \mathbb{Z} , while the second identifies `in`(n) and `in`($n + 2$) for every $n : \mathbb{Z}$. To construct a function out of $\mathbb{Z}/2\mathbb{Z}$, we simply explain where to send `in` and `mod`, in direct analogy with the induction principle of an ordinary inductive type. In addition to quotients, HITs permit the definition of higher-dimensional objects, enabling the use of type theory as a domain-specific language for formalizing homotopy-theoretic mathematics [30].

While contentful equality creates new possibilities, it also introduces new obligations. The type of equalities between any pair of objects has its own equality structure, which means that every type actually contains an infinite tower of structure: paths, paths between paths, and so on. A user of HITs is often forced to wrestle with this higher-dimensional structure. A particularly vicious example is provided by the *smash product* [30, §6.8], a key construction in homotopy theory. The smash product is a binary operator $-\wedge- : \mathcal{U}_* \rightarrow \mathcal{U}_* \rightarrow \mathcal{U}_*$ on the universe $\mathcal{U}_* := (X : \mathcal{U}) \times X$ of *pointed types*. Defined as a HIT (see Section 4.3), it is the natural notion of tensor product for \mathcal{U}_* , being left adjoint to the pointed function space. We thus expect properties such as commutativity and associativity: for any $X, Y, Z : \mathcal{U}_*$, we hope that $X \wedge Y \simeq Y \wedge X$ and $(X \wedge Y) \wedge Z \simeq X \wedge (Y \wedge Z)$. However, these laws are not so simple to prove. The associator, in particular, involves two stacked applications of \wedge ; this means the programmer must wrangle with two-dimensional structure to define functions back and forth, then three-dimensional structure to prove they are inverses. Worse yet, these are only the first level of an infinite hierarchy of laws satisfied by \wedge . For example, Mac Lane’s pentagon relates the two different ways of re-associating the product of four types; to prove it requires building *four*-dimensional terms. Despite concerted effort [31, 12], a complete formal proof that the smash product is a symmetric monoidal product has yet to be produced.

For all the suffering, these properties seem “obvious” in a way familiar to computer scientists: they look like consequences of *parametricity* [25]. Parametricity is Reynolds’ crystallization of a property enjoyed by many type theories: programs behave uniformly in their type variables. Reynolds captures this uniformity in the existence of a *relational interpretation* of type theory that expresses the invariance of type constructions under a broad class of relations. With parametricity, it is often possible to derive what Wadler dubs “free theorems” [33], naturality properties enjoyed by any term of a given type. In our case, we can hope that there are only so many functions $\alpha : (X, Y, Z : \mathcal{U}_*) \rightarrow (X \wedge Y) \wedge Z \rightarrow X \wedge (Y \wedge Z)$. Perhaps *any* definable function of this type satisfies Mac Lane’s pentagon?

Contributions

We present a cubical type theory with *internal parametricity*, a further extension to type theory introduced by Bernardy and Moulin [8, 9, 6, 21, 23, 22]. Just as cubical type theory makes the fact that all constructions act on isomorphisms available to the user, internally parametric type theory exposes that all constructions act on relations, providing an *operational* account of Reynolds’ denotational presentation of parametricity. In fact, the two are based on the same design principle: the use of *dimension variables*.

Our contribution can be viewed in two ways. On the one hand, we bring parametricity to bear on problems in cubical type theory, giving in particular a characterization of maps between smash products. On the other, we provide an internally parametric theory that enjoys the extensionality principles of cubical type theory. A lack of function extensionality, for one, is acutely visible when working with Church encodings. By relying on univalence, we are able to eliminate the technical device of *I-sets* used by Bernardy, Coquand, and Moulin in their presheaf model [6]. We also explore the use of internal parametricity beyond the initial forays of Bernardy et al., internally developing the sub-universe of *bridge-discrete*

types (which is closed under all type formers except the universe) as a substitute for the traditional identity extension lemma. This is a departure from previous approaches [4, 23] where the goal is typically to *externally* restrict the universe to bridge-discrete types from the start. We also show by example that the relational interpretations of inductive types can be characterized. Finally, we use this paper as an opportunity to compare and contrast the mechanisms underlying cubical and parametric type theory.

We begin in Section 2 by introducing cubical type theory. In Section 3, we mix in the parametricity primitives. With the theory complete, we make use of it in Section 4, proving results about the smash product and probing the status of the identity extension lemma. In Section 5, we go into more detail on the meaning of the judgments and canonicity. In Section 6, we briefly sketch a presheaf model. We close in Section 7 with a discussion of related work. Complete proofs of our results, and in particular a detailed development of the computational interpretation, can be found in our companion technical report [15].

2 Cubical type theory

Cubical type theory, in its various incarnations, is a means of organizing the data of a type equipped with path structure. Homotopy theory suggests various ways of doing this; empirically, *cubical* structure is most convenient for the design of type theories, because n -dimensional elements of a type can be represented by terms in a context of n *dimension variables*. In this section, we recall cartesian cubical type theory [3, 2]; however, one can substitute another cubical type theory (e.g., [17, 24]) in the remainder of this paper without difficulty.

2.1 Path dimensions

Like ordinary type theory, cubical type theory is based on four judgment forms, expressing typehood, type equality, elementhood, and element equality.

$$\Gamma \gg A \text{ type} \quad \Gamma \gg A = B \text{ type} \quad \Gamma \gg M \in A \quad \Gamma \gg M = N \in A$$

For us, these judgments are *behavioral specifications* on terms of an untyped programming language. Roughly, a program A is a type when, for any instantiation of its hypotheses, it computes to a name in some prescribed set of *value types*, while M is in A when its instantiations compute to values in the type named by A . Types and elements are equal when they compute to the same values, where value equality is again prescribed in advance. We use the notation \gg for the behavioral counterpart of the formal \vdash . To more quickly give the reader a feel for the system, however, we will defer precise definitions of the judgments to Section 5, and instead first present a collection of rules they satisfy.

Cubical type theory is distinguished by the addition of *path dimensions*, for which we write r, s . These are specified by judgments $\Gamma \gg r \text{ pdim}$ and $\Gamma \gg r = s \text{ pdim}$ and populated by variables and two distinguished constants.

$$\Gamma, x : \mathbb{I}, \Gamma' \gg x \text{ pdim} \quad \Gamma \gg 0 \text{ pdim} \quad \Gamma \gg 1 \text{ pdim}$$

In the context we write dimension variable assumptions in the form $x : \mathbb{I}$, but this is merely suggestive notation: “ \mathbb{I} ” is not the name of a type. We also allow dimension *equality* assumptions. (Note that dimension equality is decidable.)

13:4 Internal Parametricity for Cubical Type Theory

$$\begin{array}{c}
\frac{\Gamma, x : \mathbb{I} \gg A \text{ type} \quad \Gamma \gg M_0 \in A\langle 0/x \rangle \quad \Gamma \gg M_1 \in A\langle 1/x \rangle}{\Gamma \gg \text{Path}_{x.A}(M_0, M_1) \text{ type}} \\
\\
\frac{\Gamma, x : \mathbb{I} \gg P \in A \quad \Gamma \gg P\langle 0/x \rangle = M_0 \in A\langle 0/x \rangle \quad \Gamma \gg P\langle 1/x \rangle = M_1 \in A\langle 1/x \rangle}{\Gamma \gg \lambda^{\mathbb{I}}x.P \in \text{Path}_{x.A}(M_0, M_1)} \\
\\
\frac{\Gamma \gg Q \in \text{Path}_{x.A}(M_0, M_1) \quad \Gamma \gg r \text{ pdim}}{\Gamma \gg Q@r \in A\langle r/x \rangle} \qquad \frac{\Gamma, x : \mathbb{I} \gg P \in A}{\Gamma \gg (\lambda^{\mathbb{I}}x.P)@r = P\langle r/x \rangle \in A\langle r/x \rangle} \\
\\
\frac{\Gamma \gg Q \in \text{Path}_{x.A}(M_0, M_1) \quad \varepsilon \in \{0, 1\}}{\Gamma \gg Q@{\varepsilon} = M_{\varepsilon} \in A\langle \varepsilon/x \rangle} \qquad \frac{\Gamma \gg Q \in \text{Path}_{x.A}(M_0, M_1)}{\Gamma \gg Q = \lambda^{\mathbb{I}}x.Q@x \in \text{Path}_{x.A}(M_0, M_1)}
\end{array}$$

■ **Figure 1** Rules for Path-types.

$$\frac{}{\cdot \text{ ctx}} \qquad \frac{\Gamma \text{ ctx} \quad \Gamma \gg A \text{ type}}{\Gamma, a : A \text{ ctx}} \qquad \frac{\Gamma \text{ ctx}}{\Gamma, x : \mathbb{I} \text{ ctx}} \qquad \frac{\Gamma \text{ ctx} \quad \Gamma \gg r \text{ pdim} \quad \Gamma \gg s \text{ pdim}}{\Gamma, r = s \text{ ctx}}$$

A path dimension variable can be pictured as varying in the real unit interval $[0, 1]$: as x varies in a term $x : \mathbb{I} \gg M \in A$, the term M draws out a “line” in A . The line’s “endpoints” are obtained by substituting 0 and 1 for x : writing $\langle r/x \rangle$ for path dimension substitution, we have $M\langle 0/x \rangle \in A\langle 0/x \rangle$ and $M\langle 1/x \rangle \in A\langle 1/x \rangle$. Path dimension variables support all of the structural rules (weakening, contraction, and exchange) enjoyed by ordinary variables. In contrast to ordinary variables, however, there is more to M than its closed instantiations $M\langle 0/x \rangle$ and $M\langle 1/x \rangle$: there can be many distinct terms with the same endpoints.

Path dimensions provide a judgmental notion of contentful equality: a path between $M_0 \in A$ and $M_1 \in A$ is a term $x : \mathbb{I} \gg P \in A$ such that $P\langle 0/x \rangle = M_0$ and $P\langle 1/x \rangle = M_1$. The judgmental notion is then internalized via *Path-types*, shown in Figure 1. Aside from the indices M_0 and M_1 , they behave as “functions out of \mathbb{I} ”: they are introduced by abstraction, eliminated by application, and satisfy β - and η -rules. In general, Path-types are heterogeneous, meaning that they are *dependent* functions: they take the form $\text{Path}_{x.A}(M_0, M_1)$ where $x : \mathbb{I} \gg A \text{ type}$, and applying $Q \in \text{Path}_{x.A}(M_0, M_1)$ at r yields $Q@r \in A\langle r/x \rangle$. When A does *not* depend on x , we simply write $\text{Path}_A(M_0, M_1)$.

Using just these few principles of cubical type theory, we can already observe that path types validate function extensionality. Given $F_0, F_1 \in (a:A) \rightarrow B$, a pointwise path between them is a term $H \in (a:A) \rightarrow \text{Path}_B(F_0a, F_1a)$. Given such an H , we obtain a path between F_0 and F_1 by simply flipping the order of abstraction: $\lambda^{\mathbb{I}}x.\lambda a.Ha@x \in \text{Path}_{(a:A) \rightarrow B}(F_0, F_1)$.

2.2 Coercion and composition

The path apparatus equips each type with some kind of infinite-dimensional relation. It is reflexive: given any $M \in A$, we have $\lambda^{\mathbb{I}}_.M \in \text{Path}_A(M, M)$. However, there is as yet no reason for it to be symmetric or transitive, nor for all constructions to respect it. These properties are ensured by adding two operations: *coercion* and *composition*.

The coercion operation turns paths between types into isomorphisms,¹ implementing one direction of the correspondence required by the univalence axiom. Given a line $x.A$ and an element $M \in A\langle r/x \rangle$ at some index r , coercion produces an element at any other index s .

$$\frac{\Gamma, x : \mathbb{I} \gg A \text{ type} \quad \Gamma \gg r, s \text{ pdim} \quad \Gamma \gg M \in A\langle r/x \rangle}{\Gamma \gg \text{coe}_{x.A}^{r \rightsquigarrow s}(M) \in A\langle s/x \rangle}$$

We moreover impose the equation $\text{coe}_{x.A}^{r \rightsquigarrow r}(M) = M \in A\langle r/x \rangle$. From this, one can show that $\lambda a. \text{coe}_{x.A}^{r \rightsquigarrow s}(a) \in A\langle r/x \rangle \rightarrow A\langle s/x \rangle$ is in fact an isomorphism. Using coercion, we can see that all constructions respect paths, in the following sense: if we have some property $B \in A \rightarrow \mathcal{U}$, a proof $N \in BM_0$ that B holds of some $M_0 \in A_0$, and a path $Q \in \text{Path}_A(M_0, M_1)$, we get a proof $\text{coe}_{x.B(Q @ x)}^{0 \rightsquigarrow 1}(N) \in BM_1$ that B holds of M_1 . This fact can be used to invert and compose paths, establishing that path equality is symmetric and transitive.

While coercion gives us all we need of equality, it is not a strong enough “induction hypothesis.” Operationally, the evaluation of a coercion term is guided by the outermost constructor of type line. To explain the reduction of coercion for `Path`-types, a second operation, (*homogeneous*) *composition*, is required to obtain a term with the correct endpoints. As the purpose of this operation is essentially technical, however, we defer to [3, 2] for details.

2.3 Paths in the universe: V-types and univalence

Coercion converts paths of types into isomorphisms, but we still need a way to convert isomorphisms into paths. This is accomplished by a new type former: *Glue-types* in [17, 2], *V-types* in [3], and *G-types* in [11]. We use *V-types*, a sufficient special case of *Glue-types*.

The *V* constructor does not, strictly speaking, convert an isomorphism into a path. Rather, it takes an isomorphism and a path as input, and produces a *second* path that is the concatenation of the two inputs. Precisely, it takes a dimension r `pdim`, some $r = 0 \gg A$ type defined at its 0 endpoint, some B type, and an isomorphism $r = 0 \gg E \in A \simeq B$. These inputs form a *V*-shape, hence the name.

$$\begin{array}{ccc} A & & \\ E \downarrow & \searrow \text{V}_r(A, B, E) & \\ B_0 & \xrightarrow{B} & B_1 \\ r \rightarrow & & \end{array}$$

The output is a type $\text{V}_r(A, B, E)$ satisfying the equations $r = 0 \gg \text{V}_r(A, B, E) = A$ type and $r = 1 \gg \text{V}_r(A, B, E) = B$ type.

To transform an isomorphism $E \in A \simeq B$ of types $A, B \in \mathcal{U}$ into a path, we simply take $\lambda^{\mathbb{I}x}. \text{V}_x(A, B, E) \in \text{Path}_{\mathcal{U}}(A, B)$. This is the special case of the *V*-type where B is constant in the direction of the output (here x).

2.4 Higher inductive types

As described in the introduction, cubical type theory also supports *higher inductive types*, types generated by constructors that may take dimension arguments and be attached at their boundaries to other elements. We refer to [18, 14] for formal treatments of HITs in cubical type theory; for this paper, we will only need an intuitive understanding.

¹ For us, an *isomorphism* is a function with a left and right inverse up to path equality. That is, we define $A \simeq B$ to be the following type.

$$(f : A \rightarrow B) \times (l : B \rightarrow A) \times (r : B \rightarrow A) \times ((a : A) \rightarrow \text{Path}_A(l(fa), a)) \times ((b : B) \rightarrow \text{Path}_B(f(rb), b))$$

These are often called *equivalences* in the literature, and have several equivalent definitions [30, §4].

3 Internalizing parametricity

We now add *internal parametricity* primitives, closely following Bernardy, Coquand, and Moulin [6]. (Our notation, however, differs substantially from theirs; see [15, Figure 7] for a translation dictionary.) Reynolds’ parametricity captures the vague concept of “uniformity in type variables” by the precise concept of *acting on relations*. To say that all constructions act on relations is essentially to say that type theory has a relational semantics. With internal parametricity, we make that semantics visible inside the theory.

With cubical type theory, the goal was to ensure that all constructions act on isomorphisms; the solution was to equip each type with equality structure via dimension variables, then to identify lines *between* types with isomorphisms. For internal parametricity, we ensure that constructions act on relations with the same technique, but now identifying lines between types with *type-valued relations*. Where we use the word *path* in cubical type theory, we will use *bridge* in parametric type theory, following Nuyts et al. [23].

3.1 Bridge dimensions

Second verse, same as the first: we introduce *bridge dimensions* $\mathbf{r}, \mathbf{s}, \dots$ by judgments $\Gamma \gg \mathbf{r} \text{ bdim}$ and $\Gamma \gg \mathbf{r} = \mathbf{s} \text{ bdim}$ with two constants $\Gamma \gg \mathbf{0}, \mathbf{1} \text{ bdim}$. We use **bold type** to distinguish bridge from path dimensions. We likewise add bridge dimension and equality assumptions – but this time, only equations where one side is a constant.

$$\frac{\Gamma \text{ ctx}}{\Gamma, \mathbf{x} : \mathbf{2} \text{ ctx}} \qquad \frac{\Gamma \text{ ctx} \quad \Gamma \gg \mathbf{r} \text{ bdim} \quad \varepsilon \in \{\mathbf{0}, \mathbf{1}\}}{\Gamma, \mathbf{r} = \varepsilon \text{ ctx}}$$

The distinguishing feature of bridge dimensions is that they are *substructural*, specifically *affine*: they do not support contraction. For $\Gamma \text{ ctx}$ and $(\mathbf{x} : \mathbf{2}) \in \Gamma$, write $\Gamma \setminus^{\mathbf{x}}$ for the result of deleting \mathbf{x} and all term variables that occur beyond it from the context.²

$$(\Gamma, \mathbf{y} : \mathbb{I}) \setminus^{\mathbf{x}} := (\Gamma \setminus^{\mathbf{x}}, \mathbf{y} : \mathbb{I}) \quad (\Gamma, \mathbf{a} : A) \setminus^{\mathbf{x}} := \Gamma \setminus^{\mathbf{x}} \quad (\Gamma, \mathbf{y} : \mathbf{2}) \setminus^{\mathbf{x}} := \begin{cases} \Gamma & \text{if } \mathbf{x} = \mathbf{y} \\ (\Gamma \setminus^{\mathbf{x}}, \mathbf{y} : \mathbf{2}) & \text{if } \mathbf{x} \neq \mathbf{y} \end{cases}$$

Set $\Gamma \setminus^{\varepsilon} = \Gamma$. We then have the following structural rules for bridge dimension variables.

$$\frac{}{\Gamma, \mathbf{x} : \mathbf{2}, \Gamma' \gg \mathbf{x} \text{ bdim}} \text{BHYP} \qquad \frac{\Gamma' \gg \mathcal{J}}{\Gamma, \mathbf{x} : \mathbf{2}, \Gamma' \gg \mathcal{J}} \text{BWEAK}$$

$$\frac{\Gamma \gg \mathbf{r} \text{ bdim} \quad \Gamma \setminus^{\mathbf{r}}, \mathbf{x} : \mathbf{2} \gg \Gamma' \text{ ctx} \quad \Gamma \setminus^{\mathbf{r}}, \mathbf{x} : \mathbf{2}, \Gamma' \gg \mathcal{J}}{\Gamma(\Gamma' \langle \mathbf{r}/\mathbf{x} \rangle) \gg \mathcal{J} \langle \mathbf{r}/\mathbf{x} \rangle} \text{BCUT}$$

The first two rules are unsurprising, but the third contains an essential restriction. To substitute \mathbf{r} for \mathbf{x} in a judgment $\Gamma, \mathbf{x} : \mathbf{2}, \Gamma' \gg \mathcal{J}$, we must know that neither Γ' or \mathcal{J} refers to \mathbf{r} (if it is a variable). In other words, \mathbf{r} must be *fresh* for Γ' and \mathcal{J} .

The Bezem-Coquand-Huber (BCH) cubical sets model [10, 11] also uses affine dimension variables, but recent work on cubical type theories has focused on structural variables. The primary motivation for the shift is to support HITs, whose development remains an open problem in the affine setting; ease of implementation is another factor. For internal

² Here we follow the approach developed by Cheney for nominal dependent type theory [16].

$$\begin{array}{c}
\Gamma \gg \mathbf{r} \text{ bdim} \quad \Gamma \setminus^{\mathbf{r}}, \mathbf{x} : \mathbf{2} \gg A \text{ type} \quad \Gamma \setminus^{\mathbf{r}}, \mathbf{x} : \mathbf{2}, a : A \gg B \text{ type} \quad \Gamma \gg M \in A \langle \mathbf{r}/\mathbf{x} \rangle \\
\Gamma \setminus^{\mathbf{r}}, a_0 : A \langle \mathbf{0}/\mathbf{x} \rangle \gg N_0 \in B \langle \mathbf{0}/\mathbf{x} \rangle [a_0/a] \quad \Gamma \setminus^{\mathbf{r}}, a_1 : A \langle \mathbf{1}/\mathbf{x} \rangle \gg N_1 \in B \langle \mathbf{1}/\mathbf{x} \rangle [a_1/a] \\
\Gamma \setminus^{\mathbf{r}}, a_0 : A \langle \mathbf{0}/\mathbf{x} \rangle, a_1 : A \langle \mathbf{1}/\mathbf{x} \rangle, \bar{a} : \text{Bridge}_{\mathbf{x}.A}(a_0, a_1) \gg \bar{N} \in \text{Bridge}_{\mathbf{x}.B[\bar{a}@\mathbf{x}/a]}(N_0, N_1) \\
\hline
\Gamma \gg \text{extent}_{\mathbf{r}}(M; a_0.N_0, a_1.N_1, a_0.a_1.\bar{a}.\bar{N}) \in B \langle \mathbf{r}/\mathbf{x} \rangle [M/a] \\
\text{extent}_{\varepsilon}(M; \dots) = N_{\varepsilon}[M/a_{\varepsilon}] \in B \langle \varepsilon/\mathbf{x} \rangle [M/a] \\
\Gamma \setminus^{\mathbf{r}}, \mathbf{x} : \mathbf{2} \gg M \in A \\
\hline
\Gamma \gg \text{extent}_{\mathbf{r}}(M \langle \mathbf{r}/\mathbf{x} \rangle; \dots) = \bar{N}[M \langle \mathbf{0}/\mathbf{x} \rangle / a_0][M \langle \mathbf{1}/\mathbf{x} \rangle / a_1][\lambda^{\mathbf{2}}\mathbf{x}.M/\bar{a}]@_{\mathbf{r}} \in B \langle \mathbf{r}/\mathbf{x} \rangle [M/a]
\end{array}$$

■ **Figure 2** The extent operator. We omit straightforwardly inferrable premises for readability.

parametricity, however, affine dimensions are essential to ensure the correct characterization of bridges in function types (Section 3.2) and in the universe (Section 3.3).

As with paths, we introduce types $\text{Bridge}_{\mathbf{x}.A}(M_0, M_1)$ of bridges over $\mathbf{x}.A$ from M_0 to M_1 . We write $\lambda^{\mathbf{2}}\mathbf{x}.P$ for the values of these types. In accordance with the judgmental structure, a bridge can only be applied to a fresh variable: if $\Gamma \setminus^{\mathbf{r}} \gg Q \in \text{Bridge}_{\mathbf{x}.A}(M_0, M_1)$, then $\Gamma \gg Q@_{\mathbf{r}} \in A \langle \mathbf{r}/\mathbf{x} \rangle$. Otherwise, they are exactly like path types; see [15, §8.3] for rules. We now have one direction of our desired correspondence between bridges of types and binary relations: for any $\mathbf{x}.A$, we have the relation $\text{Bridge}_{\mathbf{x}.A}(-, -)$ on $A \langle \mathbf{0}/\mathbf{x} \rangle$ and $A \langle \mathbf{1}/\mathbf{x} \rangle$.

3.2 Bridges at function type: extent

In the standard relational interpretation of type theory [7, 4], two functions are related when they take related arguments to related results. As such, we expect $\text{Bridge}_{\mathbf{x}.(a:A) \rightarrow B}(F_0, F_1)$ to be isomorphic to the following.

$$(a_0 : A \langle \mathbf{0}/\mathbf{x} \rangle)(a_1 : A \langle \mathbf{1}/\mathbf{x} \rangle)(q : \text{Bridge}_{\mathbf{x}.A}(a_0, a_1)) \rightarrow \text{Bridge}_{\mathbf{x}.B[q@\mathbf{x}/a]}(F_0 a_0, F_1 a_1) \quad (1)$$

Although it is simple to define a map from $\text{Bridge}_{\mathbf{x}.(a:A) \rightarrow B}(F_0, F_1)$ to the type (1), the converse is more delicate. In fact, the first role of substructurality is in enabling this principle. As we have seen, structural dimensions give rise to a *different* principle: $\text{Path}_{\mathbf{x}.(a:A) \rightarrow B}(F_0, F_1) \simeq (a:A) \rightarrow \text{Path}_{\mathbf{x}.B}(F_0 a, F_1 a)$ when A does not depend on \mathbf{x} . The proof, sketched in Section 2.1, uses the interchangeability of terms $\lambda a.\lambda^{\mathbb{1}}\mathbf{x}.P$ and $\lambda^{\mathbb{1}}\mathbf{x}.\lambda a.P$. However, $\lambda a.\lambda^{\mathbf{2}}\mathbf{x}.P$ and $\lambda^{\mathbf{2}}\mathbf{x}.\lambda a.P$ are *not* interchangeable: in the former, \mathbf{x} ranges over dimensions that are *fresh for* a , whereas no such restriction is in play in the latter. Conversely, structural dimensions would not allow us to prove that the map from $\text{Bridge}_{\mathbf{x}.(a:A) \rightarrow B}(F_0, F_1)$ to (1) is an isomorphism, as having both principles at once would lead to a contradiction (in the presence of Gel-types, defined below). The two principles *are* both derivable for paths, but only thanks to the presence of coercion – an operation with no equivalent for bridges.

To see how we can get from (1) to $\text{Bridge}_{\mathbf{x}.(a:A) \rightarrow B}(F_0, F_1)$ with affine dimensions, let H in (1) be given. Abstracting \mathbf{x} and a , our goal is to exhibit a term in B that is equal to $F_0 a$ when $\mathbf{x} = \mathbf{0}$ and $F_1 a$ when $\mathbf{x} = \mathbf{1}$. Recall that a , being abstracted when \mathbf{x} is in scope, ranges over terms that may mention \mathbf{x} . We would like, then, to think of a as a *bridge* in direction \mathbf{x} , and to supply that bridge to H . In syntax, we would like to write “ $H(a \langle \mathbf{0}/\mathbf{x} \rangle)(a \langle \mathbf{1}/\mathbf{x} \rangle)(\lambda^{\mathbf{2}}\mathbf{x}.a)@_{\mathbf{x}}$ ”. This does not quite make sense: a is a variable, so $a \langle \mathbf{0}/\mathbf{x} \rangle = a$.

We instead introduce an operator, $\text{extent}_{\mathbf{r}}(M; a_0.N_0, a_1.N_1, a_0.a_1.\bar{a}.\bar{N})$, that performs the substitutions and capture once a (now M) is instantiated. This operator satisfies the rules shown in Figure 2; we call it *extent* because it reveals the extent of the term M as a

$$\begin{array}{c}
 \frac{\Gamma \gg \mathbf{r} \text{ bdim} \quad \Gamma \setminus^{\mathbf{r}} \gg A \text{ type} \quad \Gamma \setminus^{\mathbf{r}} \gg B \text{ type} \quad \Gamma \setminus^{\mathbf{r}}, a : A, b : B \gg R \text{ type}}{\Gamma \gg \text{Gel}_{\mathbf{r}}(A, B, a.b.R) \text{ type}} \\
 \\
 \frac{\Gamma \setminus^{\mathbf{r}} \gg M \in A \quad \Gamma \setminus^{\mathbf{r}} \gg N \in B \quad \Gamma \setminus^{\mathbf{r}} \gg P \in R[M, N/a, b]}{\Gamma \gg \text{gel}_{\mathbf{r}}(M, N, P) \in \text{Gel}_{\mathbf{r}}(A, B, R)} \\
 \\
 \frac{\Gamma, \mathbf{x} : \mathbf{2} \gg Q \in \text{Gel}_{\mathbf{x}}(A, B, E)}{\Gamma \gg \text{ungel}(\mathbf{x}.Q) \in R[Q\langle \mathbf{0}/\mathbf{x} \rangle, Q\langle \mathbf{1}/\mathbf{x} \rangle/a, b]} \\
 \\
 \text{Gel}_{\mathbf{0}}(A, B, a.b.R) = A \quad \text{Gel}_{\mathbf{1}}(A, B, a.b.R) = B \quad \text{gel}_{\mathbf{0}}(M, N, P) = M : A \\
 \text{gel}_{\mathbf{1}}(M, N, P) = N : B \quad \text{ungel}(\mathbf{x}.\text{gel}_{\mathbf{x}}(M, N, P)) = P : R[M, N/a, b] \\
 Q\langle \mathbf{r}/\mathbf{x} \rangle = \text{gel}_{\mathbf{r}}(Q\langle \mathbf{0}/\mathbf{x} \rangle, Q\langle \mathbf{1}/\mathbf{x} \rangle, \mathbf{x}.Q) : \text{Gel}_{\mathbf{r}}(A, B, a.b.R)
 \end{array}$$

■ **Figure 3** Rules for Gel-types.

line in direction \mathbf{r} . There are three cases: either \mathbf{r} is $\mathbf{0}$ or $\mathbf{1}$, in which case M is simply a point, or \mathbf{r} is a variable \mathbf{x} , in which case M is a bridge in direction \mathbf{x} . The operator takes an argument for each case, here N_0 , N_1 , and \bar{N} . The last of these takes as input endpoints a_0, a_1 and a bridge \bar{a} between them and produces a bridge between N_0 and N_1 ; when $\text{extent}_{\mathbf{x}}$ executes, it supplies $M\langle \mathbf{0}/\mathbf{x} \rangle$, $M\langle \mathbf{1}/\mathbf{x} \rangle$, and $\lambda^2 \mathbf{x}.M$ to \bar{N} and outputs its value at \mathbf{x} .

Substructurality is essential to extent because of its use of variable capture: the mapping $\lambda^2 : (\mathbf{x}, M) \rightsquigarrow \lambda^2 \mathbf{x}.M$ is not stable under all dimension substitutions. If $M = M'(\mathbf{x}, \mathbf{y})$, for example, then applying λ^2 after substituting $\langle \mathbf{y}/\mathbf{x} \rangle$ results in $\lambda^2 \mathbf{y}.M'(\mathbf{y}, \mathbf{y})$, while applying λ^2 before substituting $\langle \mathbf{y}/\mathbf{x} \rangle$ results in the inequivalent $\lambda^2 \mathbf{x}.M'(\mathbf{x}, \mathbf{y})$. However, variable capture *does* commute with substitution of *fresh* variables. Returning to the original motivation, extent is exactly what is needed to get from (1) to $\text{Bridge}_{\mathbf{x}.(a:A) \rightarrow B}(F_0, F_1)$.

$$H \rightsquigarrow \lambda^2 \mathbf{x}.\lambda a.\text{extent}_{\mathbf{x}}(a; a_0.F_0 a_0, a_1.F_1 a_1, a_0.a_1.\bar{a}.H a_0 a_1 \bar{a}) : \text{Bridge}_{\mathbf{x}.(a:A) \rightarrow B}(F_0, F_1)$$

It is straightforward to show that this map is in fact an isomorphism [15, Theorem 6.9].

3.3 Bridges in the universe: Gel-types and relativity

The final ingredient is the equivalent of univalence: a characterization of bridges in the universe as binary type-valued relations. We call this property *relativity*.

► **Definition 1.** *A universe \mathcal{U} is relativistic when for every pair of types $A, B : \mathcal{U}$, the map $\lambda C.\text{Bridge}_{\mathbf{x}.C @_{\mathbf{x}}}(-, -) \in \text{Bridge}_{\mathcal{U}}(A, B) \rightarrow (A \times B \rightarrow \mathcal{U})$ is an isomorphism.*

As in cubical type theory, we implement the inverse map with a new type constructor: **Gel**, so named because it resembles the **G**-types of the BCH model but applies to **relations** rather than isomorphisms. Rules for **Gel**-types – omitting those for coercion and composition – are displayed in Figure 3. In stark contrast to **V**- or **Glue**-types, coercion and composition in **Gel**-types are simple; this is because the direction of a coercion or composition is always a path dimension and therefore orthogonal to the direction of the **Gel**-type.

Given a dimension \mathbf{r} and a relation $\Gamma \setminus^{\mathbf{r}}, a : A, b : B \gg R \text{ type}$ for which \mathbf{r} is fresh, we obtain a type $\text{Gel}_{\mathbf{r}}(A, B, a.b.R)$ satisfy $\text{Gel}_{\mathbf{0}}(A, B, a.b.R) = A$ and $\text{Gel}_{\mathbf{1}}(A, B, a.b.R) = B$. Its values take the form $\text{gel}_{\mathbf{r}}(M, N, P)$ where $M \in A$, $N \in B$, and P is a proof the

two are related by R ; we have $\text{gel}_0(M, N, P) = M \in A$ and $\text{gel}_1(M, N, P) = N \in B$. Given a bridge $\Gamma, \mathbf{x} : \mathbf{2} \gg Q \in \text{Gel}_{\mathbf{x}}(A, B, a.b.R)$ over a Gel-type, we can project the proof $\text{ungel}(\mathbf{x}.Q) : R[Q\langle \mathbf{0}/\mathbf{x} \rangle, Q\langle \mathbf{1}/\mathbf{x} \rangle/a, b]$ that its endpoints – elements of A and B respectively – are related by R . When we have $A, B \in \mathcal{U}$ and $R \in A \times B \rightarrow \mathcal{U}$, we write $\text{Gel}_{\mathbf{r}}(A, B, R)$ as shorthand for $\text{Gel}_{\mathbf{r}}(A, B, a.b.R\langle a, b \rangle)$.

Whereas \mathbf{V} -types concatenate an isomorphism and a path to produce a path, Gel-types directly convert relations to bridges. That this is possible is a consequence of substructurality. The constructor $\text{Gel}_{\mathbf{x}}$ performs a dimension shift: it takes A, B , and R in some context Γ , and it produces a type in context $\Gamma, \mathbf{x} : \mathbf{2}$. To express a typing rule for Gel, we must be able to specify that \mathbf{x} is fresh for A, B, R . By contrast, $\mathbf{V}_{\mathbf{x}}$ takes a type in context Γ, \mathbf{x} with an isomorphism at one end and produces a type in context Γ, \mathbf{x} ; there is no dimension shift.

Moreover, a \mathbf{V} -like type would be insufficient for internal parametricity. In cubical type theory, we can turn $E : A \simeq B$ into a path by attaching it to the constant path $\lambda^{\mathbb{I}}_.B$, which corresponds to the identity equivalence at B . For bridges, however, this might not give the desired result: the constant bridge $\lambda^{\mathbf{2}}_.B$ corresponds to the relation $\text{Bridge}_B(-, -)$, which may be distinct from the identity relation $\text{Path}_B(-, -)$. In particular, they fail to coincide when B is a universe: we have $\text{Bridge}_{\mathcal{U}}(A, B) \simeq (A \times B \rightarrow \mathcal{U}) \not\simeq (A \simeq B) \simeq \text{Path}_{\mathcal{U}}(A, B)$.

► **Theorem 2.** *Any universe \mathcal{U} closed under Gel types is relativistic.*

Sketch. Given $a : A$ and $b : B$, the gel and ungel operators constitute an isomorphism between $R\langle a, b \rangle$ and $\text{Bridge}_{\mathbf{x}. \text{Gel}_{\mathbf{x}}(A, B, R)}(a, b)$ by virtue of their β - and η -rules. By univalence, this gives a path in \mathcal{U} between the two. By function extensionality, then, we have a path from R to $\text{Bridge}_{\mathbf{x}. \text{Gel}_{\mathbf{x}}(A, B, R)}(-, -)$. Thus $\lambda R. \text{Gel}_{\mathbf{x}}(A, B, R)$ is a left inverse to $\lambda C. \text{Bridge}_{\mathbf{x}. C@_{\mathbf{x}}}(-, -)$, getting us halfway to a proof that \mathcal{U} is relativistic.

For the right inverse, we need a path from $\lambda^{\mathbf{2}}\mathbf{x}. \text{Gel}_{\mathbf{x}}(A, B, \text{Bridge}_{\mathbf{x}. C@_{\mathbf{x}}}(-, -))$ to C for $C : \text{Bridge}_{\mathcal{U}}(A, B)$. We use the fact that bridges between paths correspond to paths between bridges, a correspondence implemented by swapping binders [15, Theorem 6.2]. It thus suffices to exhibit a bridge over $\mathbf{x}. \text{Path}_{\mathcal{U}}(\text{Gel}_{\mathbf{x}}(A, B, \text{Bridge}_{\mathbf{x}. C@_{\mathbf{x}}}(-, -)), C@_{\mathbf{x}})$ between $\lambda^{\mathbb{I}}_.A$ and $\lambda^{\mathbb{I}}_.B$. Next, we apply univalence, reducing the goal to constructing a bridge over $\mathbf{x}. \text{Gel}_{\mathbf{x}}(A, B, \text{Bridge}_{\mathbf{x}. C@_{\mathbf{x}}}(-, -)) \simeq C@_{\mathbf{x}}$ between the identity equivalences on A and B .

Finally, we can show, using the characterization of bridges at function type, that bridges at an isomorphism type correspond to isomorphisms between bridges in the source and target types [15, Corollary 6.10]. That is, it is enough to show that for every $a : A$ and $b : B$, we have an isomorphism between $\text{Bridge}_{\mathbf{x}. \text{Gel}_{\mathbf{x}}(A, B, \text{Bridge}_{\mathbf{x}. C@_{\mathbf{x}}}(-, -))}(a, b)$ and $\text{Bridge}_{\mathbf{x}. C@_{\mathbf{x}}}(a, b)$. This is a special case of the inverse condition already proven. ◀

4 The practice of internal parametricity

We have completed a formulation of internally parametric cubical type theory. Now, we will use it. As a warm-up, we prove that `bool` is isomorphic to its Church encoding. Then we move on to novel results: a characterization of bridges in `bool`, the definition and applications of bridge-discrete types, and finally, a characterization of maps between smash products.

4.1 The basics: booleans

A classic application of parametricity is the characterization of *Church encodings*, definitions of inductive types by their universal properties. Our universes are predicative, so Church encodings likely cannot be used to obtain data types *ex nihilo*. If we know that a data type exists, however, we can show that it is isomorphic to its Church encoding.

13:10 Internal Parametricity for Cubical Type Theory

► **Theorem 3.** *The type $B := (X:\mathcal{U}) \rightarrow X \rightarrow X \rightarrow X$ is isomorphic to `bool`.*

Proof. It is easy to exhibit functions in either direction.

$$\lambda b.(\lambda X.\lambda t.\lambda f.\text{if}_X(b;t,f)) \in \text{bool} \rightarrow B \qquad \lambda g.g(\text{bool})(\text{true})(\text{false}) \in B \rightarrow \text{bool}$$

One inverse condition is immediate. For the other, we must construct a path from $\lambda X.\lambda t.\lambda f.\text{if}_X(g(\text{bool})(\text{true})(\text{false});t,f)$ to g for all $g : B$. Applying function extensionality, we assume $X : \mathcal{U}$ and $t, f : X$ and aim to connect $\text{if}_X(g(\text{bool})(\text{true})(\text{false});t,f)$ to $gXtf$.

Intuitively, we want to say that g is natural in its type argument; specifically, with respect to the map $\text{if}_X(-;t,f) : \text{bool} \rightarrow X$. Accordingly, we define the relation given by the graph of this map: $R : \text{bool} \times X \rightarrow \mathcal{U}$ given by $R\langle b,a \rangle := \text{Path}_X(\text{if}_X(b;t,f),a)$. To obtain the relational interpretation of g at R , we introduce a dimension \mathbf{x} and apply g at its Gel-type.

$$g(\text{Gel}_{\mathbf{x}}(\text{bool}, X, R)) \in \text{Gel}_{\mathbf{x}}(\text{bool}, X, R) \rightarrow \text{Gel}_{\mathbf{x}}(\text{bool}, X, R) \rightarrow \text{Gel}_{\mathbf{x}}(\text{bool}, X, R)$$

Next, we apply this to the related pairs (true, t) and (false, f) , in the form of $\text{gel}_{\mathbf{x}}$ terms.

$$g(\text{Gel}_{\mathbf{x}}(\text{bool}, X, R))(\text{gel}_{\mathbf{x}}(\text{true}, t, \lambda^{\mathbb{I}}_-.t))(\text{gel}_{\mathbf{x}}(\text{false}, f, \lambda^{\mathbb{I}}_-.f)) \in \text{Gel}_{\mathbf{x}}(\text{bool}, X, R)$$

Call this term W . If we substitute $\mathbf{0}$ for \mathbf{x} in W , each Gel or gel term steps to its first argument; thus $W\langle \mathbf{0}/\mathbf{x} \rangle = g(\text{bool})(\text{true})(\text{false})$. Likewise, $W\langle \mathbf{1}/\mathbf{x} \rangle = gXtf$. The term $\text{ungel}(\mathbf{x}.W)$ is then a proof that these are related by R , which is precisely what we need. ◀

We can also characterize the bridges in `bool`. Intuitively, these are all trivial: `bool`'s only elements are the zero-dimensional `true` and `false`. To show this, we use parametricity, an interesting parallel with the use of univalence for characterizing paths in HITs.

► **Theorem 4.** *For any $b_0, b_1 : \text{bool}$, we have $\text{Bridge}_{\text{bool}}(b_0, b_1) \simeq \text{Path}_{\text{bool}}(b_0, b_1)$.*

Sketch. We will only construct the forward map; for a full proof, see [15, Theorem 11.5]. Let $q : \text{Bridge}_{\text{bool}}(b_0, b_1)$. Given \mathbf{x} , we have $P_{\mathbf{x}} := \text{Gel}_{\mathbf{x}}(\text{bool}, \text{bool}, \text{Path}_{\text{bool}}(-, -))$ corresponding to the identity relation on `bool`, as well as a map $F_{\mathbf{x}} : \text{bool} \rightarrow P_{\mathbf{x}}$ defined as follows.

$$F_{\mathbf{x}} := \lambda b.\text{if}_{P_{\mathbf{x}}}(b; \text{gel}_{\mathbf{x}}(\text{true}, \text{true}, \lambda^{\mathbb{I}}_-. \text{true}), \text{gel}_{\mathbf{x}}(\text{false}, \text{false}, \lambda^{\mathbb{I}}_-. \text{false}))$$

Note that $F_{\mathbf{0}} = F_{\mathbf{1}} = \lambda b.\text{if}_{\text{bool}}(b; \text{true}, \text{false})$. By applying F pointwise to q , we thus obtain $\lambda^2 \mathbf{x}.F_{\mathbf{x}}(q@_{\mathbf{x}}) \in \text{Bridge}_{\mathbf{x}.P_{\mathbf{x}}}(\text{if}_{\text{bool}}(b_0; \text{true}, \text{false}), \text{if}_{\text{bool}}(b_1; \text{true}, \text{false}))$. It is easy to show that for any $b : \text{bool}$, there is a path from $\text{if}_{\text{bool}}(b; \text{true}, \text{false})$ to b ; using coercion, we obtain some $T \in \text{Bridge}_{\mathbf{x}.P_{\mathbf{x}}}(b_0, b_1)$. Applying ungel gives an element of $\text{Path}_{\text{bool}}(b_0, b_1)$. ◀

The definition of the forward map uses parametricity; showing that it is an isomorphism uses *iterated parametricity*, which is to say two-bridge-dimensional types. Just as a one-dimensional bridge corresponds to a relation indexed by its boundary (the two endpoint types), relativity and the characterization of bridges at function type suffice to show that two-dimensional bridges satisfy the same characterization, the boundary now being given by four types and four relations in a square shape.

4.2 Bridge-discrete types

The booleans are one example of a *bridge-discrete* type, a type with trivial bridge structure.

► **Definition 5.** A type A is bridge-discrete when for all pairs $a_0, a_1 : A$, the canonical map $\lambda p. \text{coe}_{x. \text{Bridge}_A(a_0, p @ x)}^{0 \rightsquigarrow 1}(\lambda^2 _ . a_0) \in \text{Path}_A(a_0, a_1) \rightarrow \text{Bridge}_A(a_0, a_1)$ is an isomorphism. We write $\text{isBDisc}(A)$ for the type of proofs that A is bridge-discrete.

To show that A is bridge-discrete, it suffices to exhibit *any* family of isomorphisms between Path_A and Bridge_A [15, Corollary 10.7]. The type $\text{isBDisc}(A)$ is a *homotopy proposition* [30, §3.3]: all proofs of $\text{isBDisc}(A)$ are equal up to a path.

Bridge-discrete types are worth identifying because they provide an analogue to the *identity extension lemma*, a standard lemma in parametricity stating that the relational interpretation of a closed type is its identity relation. This does not hold in our theory, in the sense that a homogeneous bridge type $\text{Bridge}_A(M_0, M_1)$ is not necessarily isomorphic to $\text{Path}_A(M_0, M_1)$ (take $A = \mathcal{U}$). However, we can instead work by inserting bridge-discreteness hypotheses where the lemma would be required. For example, in imitation of Abel et al. [1], we can prove that path equality in a bridge-discrete type is isomorphic to Leibniz equality.

► **Proposition 6** ([15, §11.2]). *Let A be bridge-discrete. For any $a_0, a_1 : A$, $\text{Path}_A(a_0, a_1)$ is isomorphic to $(P : A \rightarrow \mathcal{U}) \rightarrow Pa_0 \rightarrow Pa_1$.*

Fortunately, most type constructors preserve bridge-discreteness. We can check that $\mathcal{U}_{\text{BDisc}} := (X : \mathcal{U}) \times \text{isBDisc}(X)$ is closed under almost all the type formers, with the obvious exception of universes.

► **Proposition 7.** *The sub-universe $\mathcal{U}_{\text{BDisc}}$ is closed under product, function, Path-, and Bridge-types. It is univalent and relativistic (i.e., closed under \mathbb{V} - and Gel-types).*

Proof. For the first statement, see [15, Lemma 10.9]. Any sub-universe of a univalent universe carved out by a proposition is univalent (see [30, Lemma 3.5.1]). For the proof of relativity, see [15, Theorem 10.12]. ◀

Per Section 4.1, we also expect that $\mathcal{U}_{\text{BDisc}}$ is closed under inductive types. Proposition 7 implies that we can also use parametricity in $\mathcal{U}_{\text{BDisc}}$: for example, one can repeat the proof of Theorem 3 to show that `bool` is isomorphic to $(X : \mathcal{U}) \rightarrow \text{isBDisc}(X) \rightarrow X \rightarrow X \rightarrow X$.

Finally, we observe as a simple consequence of Theorem 4 that the excluded middle for homotopy propositions [30, §3.4] is refuted by internal parametricity. (The excluded middle for *all* types is already refuted by univalence [30, Corollary 3.27].)

► **Definition 8.** Write $\text{isProp}(A) := (a_0, a_1 : A) \rightarrow \text{Path}_A(a_0, a_1)$. The excluded middle for homotopy propositions is $\text{LEM} := (X : \mathcal{U}) \rightarrow \text{isProp}(X) \rightarrow (b : \text{bool}) \times \text{if}_{\mathcal{U}}(b; X, \neg X)$.

► **Lemma 9.** *If A is bridge-discrete, then any function $f : \mathcal{U} \rightarrow A$ is constant.*

Proof. For any pair of types $X, Y : \mathcal{U}$, we have a bridge $B := \lambda^2 \mathbf{x}. \text{Gel}_{\mathbf{x}}(X, Y, \dots, \perp)$ between them, thus a bridge $\lambda^2 \mathbf{x}. f(B @ \mathbf{x}) : \text{Bridge}_A(fX, fY)$ between their images by f , thus a path from fX to fY by bridge-discreteness of A . ◀

► **Theorem 10.** *There is a term of type $\text{LEM} \rightarrow \perp$.*

Proof. We refute the *weak excluded middle* $\text{WLEM} := (X : \mathcal{U}) \rightarrow (b : \text{bool}) \times \text{if}_{\mathcal{U}}(b; \neg X, \neg \neg X)$, the special case of LEM that decides negated types. (Any negated type is a homotopy proposition.) Let $f : \text{WLEM}$. Then $\lambda X. \text{fst}(fX) \in \mathcal{U} \rightarrow \text{bool}$. By Theorem 4 and Lemma 9, this map is constant. But $\text{fst}(f\perp)$ and $\text{fst}(f\top)$ cannot be equal, so we have a contradiction. ◀

4.3 The smash product

We adopt the convention of writing $A_* : \mathcal{U}_* := (X : \mathcal{U}) \times X$ for a pointed type, $A := \text{fst}(A_*)$ for its underlying type, and $a_0 := \text{snd}(A_*)$ for its basepoint. Given $A_*, B_* \in \mathcal{U}_*$, we write $A_* \rightarrow_* B_* := (f : A \rightarrow B) \times \text{Path}_B(fa_0, b_0)$ for the type of basepoint-preserving functions from A_* to B_* . Given $A_*, B_* : \mathcal{U}_*$, their *smash product* is the following HIT.

```

data  $A_* \wedge B_* : \mathcal{U}$  where
| pair( $a : A, b : B$ ) :  $A_* \wedge B_*$ 
| basel :  $A_* \wedge B_*$ 
| baser :  $A_* \wedge B_*$ 
| gluel( $b : B, x : \mathbb{I}$ ) :  $A_* \wedge B_*$  [ $x = 0 \hookrightarrow \text{basel} \mid x = 1 \hookrightarrow \text{pair}(a_0, b)$ ]
| gluer( $a : A, x : \mathbb{I}$ ) :  $A_* \wedge B_*$  [ $x = 0 \hookrightarrow \text{baser} \mid x = 1 \hookrightarrow \text{pair}(a, b_0)$ ]

```

In words, the smash product of A_* and B_* is the cartesian product of their underlying types modulo the relation equating all pairs of the form (a_0, b) or (a, b_0) . The smash product can itself be made a pointed type $A_* \wedge_* B_*$ with basepoint $\text{pair}(a_0, b_0)$.

Our goal is to characterize the polymorphic pointed endofunctions on n -ary smash products. Here, we will only consider the binary case, and we will only sketch the proof. For the binary case, we give detailed pen-and-paper proofs of the arguments that use parametricity directly in [15, Appendix C], and we have formalized the purely cubical arguments in the `redtt` cubical proof assistant [29, `cool.smash`].

► **Theorem 11.** *Any function $f_* : (X_*, Y_* : \mathcal{U}_*) \rightarrow X_* \wedge_* Y_* \rightarrow_* X_* \wedge_* Y_*$ is connected by a path to either the polymorphic identity or the polymorphic constant function.*

That we cannot squeeze a complete proof into this space may seem to undermine our case for parametricity’s usefulness. However, our argument is not that it is *easy*, but that it *scales*. After establishing the above, we will argue that no combinatorial explosion results from generalizing to n -ary smash products, in contrast to the more direct approaches.

► **Definition 12.** *Given $f : A \rightarrow B$, write $\text{Gr}_r(A, B, f) := \text{Gel}_r(A, B, a.b.\text{Path}_B(fa, b))$. Given $f_* : A_* \rightarrow_* B_*$, define $\text{Gr}_r^*(A_*, B_*, f_*) := \langle \text{Gr}_r(A, B, f), \text{gel}_r(a_0, b_0, f_0) \rangle \in \mathcal{U}_*$.*

The smash product has an action: from $f_* : A_* \rightarrow_* C_*$ and $g_* : B_* \rightarrow_* D_*$, we obtain $f_* \wedge g_* \in A_* \wedge B_* \rightarrow C_* \wedge D_*$. The following extracts results from products of Gr -types.

► **Lemma 13** (Graph Lemma for \wedge). *Let pointed types $A_*, B_*, C_*, D_* : \mathcal{U}_*$ and pointed functions $f_* : A_* \rightarrow_* C_*$, $g_* : B_* \rightarrow_* D_*$ be given. For any \mathbf{x} , there is a map of type*

$$\text{Gr}_{\mathbf{x}}^*(A, C, f) \wedge \text{Gr}_{\mathbf{x}}^*(B, D, g) \rightarrow \text{Gr}_{\mathbf{x}}(A_* \wedge B_*, C_* \wedge D_*, f_* \wedge g_*)$$

equal to the identity function on $A_ \wedge B_*$ when $\mathbf{x} = \mathbf{0}$ and on $C_* \wedge D_*$ when $\mathbf{x} = \mathbf{1}$.*

Sketch. This is analogous to Theorem 4; we define the map by smash product induction. We use `extent` and `ungel` to extract relation witnesses in the `pair`, `gluel`, and `gluer` cases. ◀

► **Proposition 14.** *Any element of $\text{bool}_* \wedge \text{bool}_*$ is either `pair(true, true)` or `pair(false, false)`.*

► **Lemma 15.** *Any $f : (X_*, Y_* : \mathcal{U}_*) \rightarrow X \rightarrow Y \rightarrow X_* \wedge Y_*$ is connected by a path to either*

1. *the pairing function $\lambda\langle X, x_0 \rangle. \lambda\langle Y, y_0 \rangle. \lambda a. \lambda b. \text{pair}(a, b)$, or*
2. *the constant basepoint function $\lambda\langle X, x_0 \rangle. \lambda\langle Y, y_0 \rangle. \lambda_. \lambda_. \text{pair}(x_0, y_0)$.*

Proof. Let $X_*, Y_* : \mathcal{U}_*$, $a : X$, and $b : Y$ be given. We have a function $g_*^X \in \text{bool}_* \rightarrow_* X_*$ taking `true` to x_0 and `false` to a , likewise $g_*^Y \in \text{bool}_* \rightarrow_* Y_*$ taking `true` to y_0 and `false` to b .

Fix a fresh bridge dimension \mathbf{x} . By taking the Gel-types for the graphs of the above functions, we obtain pointed types $G_*^X := \text{Gr}_{\mathbf{x}}^*(\text{bool}, X, g_*^X)$ and $G_*^Y := \text{Gr}_{\mathbf{x}}^*(\text{bool}, Y, g_*^Y)$. We apply f with the elements of G^X and G^Y corresponding to a and b .

$$f(G_*^X)(G_*^Y)(\text{gel}_{\mathbf{x}}(\text{false}, a, \lambda\mathbb{I}-.a))(\text{gel}_{\mathbf{x}}(\text{false}, b, \lambda\mathbb{I}-.b)) \in G_*^X \wedge G_*^Y$$

At $\mathbf{x} = \mathbf{0}$, this is equal to $f(\text{bool}_*)(\text{bool}_*)(\text{false})(\text{false}) \in \text{bool}_* \wedge \text{bool}_*$; at $\mathbf{x} = \mathbf{1}$, it is equal to $fX_*Y_*ab \in X_* \wedge Y_*$. By Lemma 13, we obtain a term in $\text{Gr}_{\mathbf{x}}(\text{bool}_* \wedge \text{bool}_*, X_* \wedge Y_*, g_*^X \wedge g_*^Y)$ with the same endpoints. Applying `ungel`, we get a proof that these endpoints are in the graph of $g_*^X \wedge g_*^Y$, i.e., that $(g_*^X \wedge g_*^Y)(f(\text{bool}_*)(\text{bool}_*)(\text{false})(\text{false}))$ is path-equal to fX_*Y_*ab . By Proposition 14, $f(\text{bool}_*)(\text{bool}_*)(\text{false})(\text{false})$ has two possible values. If it is `pair(true, true)`, then fX_*Y_*ab must be `pair(x0, y0)`; if it is `pair(false, false)`, then fX_*Y_*ab is `pair(a, b)`. ◀

Sketch of Theorem 11. To characterize $f_* : (X_*, Y_* : \mathcal{U}_*) \rightarrow X_* \wedge_* Y_* \rightarrow_* X_* \wedge_* Y_*$, we must characterize its behavior on each constructor, as well as the proof that it preserves the basepoint of $X_* \wedge_* Y_*$. Given X_*, Y_* , write fX_*Y_* for the function underlying $f_*X_*Y_*$.

Write $P := (X_*, Y_* : \mathcal{U}_*) \rightarrow X \rightarrow Y \rightarrow X_* \wedge Y_*$. First, we isolate the behavior of f_* on the pair constructor: $\lambda X_*.\lambda Y_*.\lambda a.\lambda b.fX_*Y_*(\text{pair}(a, b)) \in P$. By Lemma 15, this is one of two functions. We aim to show that this is the only degree of freedom available to f_* .

The values of f on the `basel` and `baser` constructors are uniquely determined up to a path by the fact that $f_*X_*Y_*$ is basepoint-preserving, as `basel` and `baser` are connected to the basepoint of $X_* \wedge_* Y_*$ by `gluel(y0, -)` and `gluer(x0, -)` respectively.

For `gluel`, we consider the term $H := \lambda\mathbb{I}x.\lambda X_*.\lambda Y_*.\lambda a.\lambda b.fX_*Y_*(\text{gluel}(b, x))$, which is a path in P from $\lambda X_*.\lambda Y_*.\lambda a.\lambda b.fX_*Y_*(\text{basel})$ to $\lambda X_*.\lambda Y_*.\lambda a.\lambda b.fX_*Y_*(\text{pair}(x_0, b))$. By Lemma 15, we know that P is isomorphic to `bool`, which means in particular that it is a *homotopy set*: its path types are all homotopy propositions. So H , and therefore the behavior of f_* on `gluel` terms, is uniquely determined (up to a path). The same applies to `gluer`.

Finally, write $f_0 : (X_*, Y_* : \mathcal{U}_*) \rightarrow \text{Path}_{X_* \wedge Y_*}(fX_*Y_*(\text{pair}(x_0, y_0)), \text{pair}(x_0, y_0))$ for the proof that f preserves the basepoint of $X_* \wedge_* Y_*$. As with `gluel`, we prove that f_0 is uniquely determined by recasting it as a path in P , namely the path $\lambda x.\lambda X_*.\lambda Y_*.\lambda a.\lambda b.f_0X_*Y_*@x$ that connects $\lambda X_*.\lambda Y_*.\lambda a.\lambda b.fX_*Y_*(\text{pair}(x_0, y_0))$ to $\lambda X_*.\lambda Y_*.\lambda a.\lambda b.\text{pair}(x_0, y_0)$. ◀

To prove the n -ary generalization of this theorem, we can proceed by an inductive argument, showing for each $i \leq n$ that there are exactly two maps of the following type.

$$(X_{1*}, \dots, X_{n*} : \mathcal{U}_*) \rightarrow X_1 \rightarrow \dots \rightarrow X_{n-i} \rightarrow (X_{n-i+1*} \wedge_* \dots \wedge_* X_{n*}) \rightarrow \bigwedge_{*i} X_{i*}$$

For the base case $i = 0$, we use an easy generalization of Lemma 15; for the inductive step, the argument in the proof of Theorem 11. What is key is that we never use an *iterated* induction argument on the elements of stacked smash products, so we never find ourselves dealing with a two-dimensional case like that of $\text{gluel}(\text{gluel}(c, x), y) \in X_* \wedge (Y_* \wedge Z_*)$.

To see how we can apply the theorem, consider the case of commutativity. If we have $K : (X_*, Y_* : \mathcal{U}) \rightarrow X_* \wedge_* Y_* \rightarrow_* Y_* \wedge_* X_*$, then $\lambda X_*.\lambda Y_* . K_{X_*, Y_*} \circ K_{Y_*, X_*}$ is a polymorphic endofunction on smash products. By Theorem 11, we can show it is the identity simply by showing it is not constant, which we can do by instantiating X_*, Y_* with `bool*`, `bool*` and testing it on `pair(false, false)`. If it is indeed non-constant, we see that K is an isomorphism. Similar techniques show that any non-constant associator is an isomorphism and satisfies Mac Lane's pentagon.

5 Computational meaning of the judgments

We now explain more precisely the meanings of the typing judgments. We follow the work of Angiuli et al. [3]. The central idea is that well-typed terms need not only evaluate to values, but evaluate in a way that is coherent with respect to substitution of dimensions.

First, we have an operational semantics, specified by judgments $M \text{ val}$ and $M \mapsto^* M'$. We write $M \Downarrow V$ when $M \mapsto^* V$ and $V \text{ val}$. The “closed” types and terms – those to which the operational semantics applies – are those whose free variables are all dimensions, i.e., those well-formed in some $\Phi\Psi \text{ ctx}$ where $\Phi = \mathbf{x}_1 : \mathbf{2}, \dots, \mathbf{x}_n : \mathbf{2}$ and $\Psi = y_1 : \mathbb{I}, \dots, y_m : \mathbb{I}$. A *type system* on this language is a five-place relation $\tau(\Phi, \Psi, A_0, A'_0, \varphi)$. It specifies, for each context $\Phi\Psi$, those values A_0 and A'_0 that are equal *value types* in that context, and associates to each such pair a partial equivalence relation (PER) φ on values in context $\Phi\Psi$. To be a type system, τ is required to satisfy laws ensuring symmetry, transitivity, and so forth. To interpret the type formers described in Sections 2 and 3, we can define an appropriate τ by a fixed-point construction populating it with the various constructors.

Two terms A and A' in context $\Phi\Psi$ are then *equal types* when they evaluate to equal type values in a way that commutes with dimension substitution.

► **Definition 16.** *Given τ , define a five-place relation $\text{PTY}(\tau)(\Phi, \Psi, A, A', \alpha)$ on context $\Phi\Psi$, terms A, A' , and families $(\alpha_\psi)_{\psi: \Phi'\Psi' \rightarrow \Phi\Psi}$ indexed by substitutions into $\Phi\Psi$, as follows. $\text{PTY}(\tau)(\Phi, \Psi, A, A', \alpha)$ holds when for all $\psi_1 : \Phi_1\Psi_1 \rightarrow \Phi\Psi$ and $\psi_2 : \Phi_2\Psi_2 \rightarrow \Phi_1\Psi_1$, we have*

$$A\psi_1 \Downarrow A_1 \quad A_1\psi_2 \Downarrow A_2 \quad A\psi_1\psi_2 \Downarrow A_{12} \quad A'\psi_1 \Downarrow A'_1 \quad A'_1\psi_2 \Downarrow A'_2 \quad A'\psi_1\psi_2 \Downarrow A'_{12}$$

with $\tau(\Phi_2, \Psi_2, V, V', \alpha_{\psi_1\psi_2})$ for all $V \in \{A_2, A_{12}\}$ and $V' \in \{A'_2, A'_{12}\}$.

We write $\llbracket A \rrbracket$ for the α such that $\text{PTY}(\tau)(\Phi_2, \Psi_2, A, A, \alpha)$ when it exists; the laws required of type systems ensure its uniqueness. We now say that $\Phi\Psi \gg A = A'$ type when $\text{PTY}(\tau)(\Phi, \Psi, A, A', \alpha)$ for some α (satisfying a certain coherence condition). As a special case, we say that $\Phi\Psi \gg A$ type when $\Phi\Psi \gg A = A$ type. Element equality is defined analogously: M and M' are equal in A when they coherently evaluate to equal values in $\llbracket A \rrbracket$. Finally, the closed judgments are extended to open judgments by functionality: $\Gamma \gg A = A'$ type holds when $\Phi\Psi \gg A\gamma = A'\gamma'$ type for all $\Phi\Psi \gg \gamma = \gamma' \in \Gamma$.

By definition of $\Phi\Psi \gg M \in \text{bool}$, we obtain a canonicity theorem.

► **Theorem 17 (Canonicity).** *If $\Phi\Psi \gg M \in \text{bool}$, then either $M \mapsto^* \text{true}$ or $M \mapsto^* \text{false}$. Additionally, either $\Phi\Psi \gg M = \text{true} \in \text{bool}$ or $\Phi\Psi \gg M = \text{false} \in \text{bool}$.*

6 Presheaf model

As we have said, the rules presented in Sections 2 and 3 can be used as a formalism for reasoning in parametric cubical type theory. Following prior work on cubical type theory [17, 2] and internal parametricity [6], this formalism also supports a presheaf semantics.

The two base categories at play, defined using the notation of Buchholtz and Morehouse [13], are the *BCH cube category* $\mathfrak{CB} := \mathbb{C}_{(\text{we}, \cdot)}$ and the *cartesian cube category* $\mathfrak{CP} := \mathbb{C}_{(\text{wec}, \cdot)}$. (Here w, e, and c stand for weakening, exchange, and contraction respectively.) We model parametric cubical type theory in the category $\mathcal{C} := \text{Set}^{(\mathfrak{CB} \times \mathfrak{CP})^{\text{op}}}$ of presheaves on the product of the two. We can immediately obtain an interpretation of the standard and cubical type formers (including the universe) by applying a result of Angiuli et al. [2, Theorem 1]. For the interval, we take $(\cdot, x : \mathbb{I}) \in \mathfrak{CB} \times \mathfrak{CP}$; for the generating cofibrations $\text{Cof} \subseteq \Omega_{\text{dec}}$, we take finite unions of equations of the form $r = s$ and $\mathbf{r} = \boldsymbol{\varepsilon}$. It is straightforward to check that these choices satisfy the axioms required to apply their theorem.

On the parametric side, we follow the BCH model. Given a context Γ interpreted as some $\llbracket \Gamma \rrbracket : \mathcal{C}$, its extension $\Gamma, \mathbf{x} : \mathbf{2}$ is interpreted as the separated product $\llbracket \Gamma \rrbracket \otimes \mathbf{y}(\mathbf{x} : \mathbf{2}, \cdot)$, whose elements at $(\Phi, \Psi) \in \mathbb{D}_{\mathbb{B}} \times \mathbb{D}_{\mathbb{P}}$ are pairs (γ, \mathbf{r}) with $\mathbf{r} \in \Phi \cup \{\mathbf{0}, \mathbf{1}\}$ and $\gamma \in \llbracket \Gamma \rrbracket(\Phi \setminus \{\mathbf{r}\}, \Psi)$. Bridge-types are interpreted à la BCH path types; the inclusion of equations $\mathbf{r} = \varepsilon$ in Cof ensures that they support coercion and composition. Gel-types can be interpreted similarly to BCH G-types (though coercion and composition are much simpler in our case).

One advantage of univalence is that we can obtain a relativistic universe without replacing sets with *I*-sets, Bernardy et al. do [6]. In their theory, the isomorphism implemented by `gel` and `ungel` is replaced by an equality $\text{Bridge}_{\mathbf{x}, \text{Gel}_{\mathbf{x}}(A, B, R)}(M, N) = R\langle M, N \rangle$. To ensure that the interpretations of these types have *exactly* the same elements, sets are everywhere replaced with *I*-sets. In our notation, these would be Φ -sets: for $\Phi \in \mathbb{D}_{\mathbb{B}}$, a Φ -element is a family indexed by subcontexts $\Phi' \subseteq \Phi$, and a Φ -set is a set of Φ -elements. Interpreting types as families of *I*-sets makes it possible to give an interpretation of Gel- and Bridge-types that validates the above equation. In our work, on the contrary, the equality is replaced by an isomorphism, obviating the need for *I*-sets. By exploiting univalence, we can still obtain a path and so establish the target isomorphism between $\text{Bridge}_{\mathcal{U}}(A, B)$ and $A \times B \rightarrow \mathcal{U}$.

7 Related and future work

Our parametric cubical type theory is, for the most part, simply the union of Angiuli et al.'s cartesian cubical type theory [3] and Bernardy et al.'s internally parametric type theory [6]. We work with binary rather than unary parametricity, but as Bernardy et al. remark, this requires only cosmetic changes. There is little interaction between the two halves of the theory; we need only to check that coercion and composition can be defined for the new types. For bridge types, this requires a minor change to the definition of composition, the addition of $\mathbf{r} = \varepsilon$ tube equations. As discussed in Section 6, our Gel-types satisfy fewer equations than the corresponding types in [6]; accordingly, our proof of relativity is novel.

A second approach to internal parametricity has been developed by Nuyts, Vezzosi, and Devriese [23]. Like ours, their system is based on *paths* and *bridges*. Where ours are almost entirely separate, however, theirs are connected by a modality, which mediates between variable uses in *continuous* and *parametric* positions. Intuitively, their goal is to internalize the independence of element-level calculation from terms at the type level, whereas ours is merely to internalize the relational interpretation. The divergence of aims leads to very different considerations; in particular, their bridge dimensions are structural. In later work [22], Nuyts and Devriese generalize from paths and bridges to an infinite tower of relations.

Parametric cubical type theory also strongly resembles Riehl and Shulman's *directed type theory* [27]. Where ours has semantics in presheaves on $\mathbb{D}_{\mathbb{B}} \times \mathbb{D}_{\mathbb{P}}$, theirs is aimed at presheaves on $\Delta \times \Delta$, two copies of the simplex category. In both cases, one half of the base category is used for equality, while the other endows types with a relational structure. For directed type theory, the goal is to identify those types for which the relational structure is actually a category structure, meaning that concatenable bridges have path-unique composites. Interestingly, their model does not appear to admit a relativistic universe [26, §2].

Stepping outside the realm of internalization, there has been general interest in higher-dimensional generalizations of parametricity, for example in the work of Atkey et al. [4], Benton et al. [5], Ghani et al. [19], and Sojakova and Johann [28]. Atkey et al.'s use of reflexive graphs to construct a parametric model with a discrete universe is to Bernardy-Moulin-style internal parametricity as the Hofmann-Streicher groupoid model [20] is to cubical type theory.

One unsolved problem looms large: to what extent do results in parametric type theory translate into ordinary type theory? Naturally, not all theorems translate – $(f:\mathcal{U} \rightarrow \text{bool}) \rightarrow \text{Path}_{\text{bool}}(f\top, f\perp)$ is provable in parametric type theory but refuted in classical cubical sets – but one reasonable conjecture is that proofs $\Gamma \gg M \in A$ translate when Γ and A use no function types. It appears fairly straightforward to obtain results of this kind in semantics, but syntactic results are much murkier; similar conservativity questions for homotopy and cubical type theory over ordinary type theory remain open.

References

- 1 Andreas Abel, Jesper Cockx, Dominique Devriese, Amin Timany, and Philip Wadler. Leibniz Equality is Isomorphic to Martin-Löf Identity, Parametrically. Submitted to the Journal of Functional Programming, February 2018. URL: <https://jesper.sikanda.be/files/leibniz-equality.pdf>.
- 2 Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. Syntax and Models of Cartesian Cubical Type Theory. Unpublished draft, February 2019. URL: <https://github.com/dlicata335/cart-cube>.
- 3 Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. Cartesian Cubical Computational Type Theory: Constructive Reasoning with Paths and Equalities. In *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, pages 6:1–6:17, 2018. doi:10.4230/LIPIcs.CSL.2018.6.
- 4 Robert Atkey, Neil Ghani, and Patricia Johann. A relationally parametric model of dependent type theory. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 503–516, 2014. doi:10.1145/2535838.2535852.
- 5 Nick Benton, Martin Hofmann, and Vivek Nigam. Abstract effects and proof-relevant logical relations. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 619–632, 2014. doi:10.1145/2535838.2535869.
- 6 Jean-Philippe Bernardy, Thierry Coquand, and Guilhem Moulin. A Presheaf Model of Parametric Type Theory. *Electr. Notes Theor. Comput. Sci.*, 319:67–82, 2015. doi:10.1016/j.entcs.2015.12.006.
- 7 Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Parametricity and dependent types. In *ICFP 2010, Baltimore, Maryland, USA, September 27-29, 2010*, pages 345–356, 2010.
- 8 Jean-Philippe Bernardy and Guilhem Moulin. A Computational Interpretation of Parametricity. In *LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 135–144, 2012. doi:10.1109/LICS.2012.25.
- 9 Jean-Philippe Bernardy and Guilhem Moulin. Type-theory in color. In *ICFP 2013, Boston, MA, USA - September 25 - 27, 2013*, pages 61–72, 2013. doi:10.1145/2500365.2500577.
- 10 Marc Bezem, Thierry Coquand, and Simon Huber. A Model of Type Theory in Cubical Sets. In *19th International Conference on Types for Proofs and Programs, TYPES 2013, April 22-26, 2013, Toulouse, France*, pages 107–128, 2013. doi:10.4230/LIPIcs.TYPES.2013.107.
- 11 Marc Bezem, Thierry Coquand, and Simon Huber. The Univalence Axiom in Cubical Sets. *J. Autom. Reasoning*, 63(2):159–171, 2019. doi:10.1007/s10817-018-9472-6.
- 12 Guillaume Brunerie. Computer-generated proofs for the monoidal structure of the smash product. *Homotopy Type Theory Electronic Seminar Talks*, November 2018. URL: <https://www.uwo.ca/math/faculty/kapulkin/seminars/hottest.html>.
- 13 Ulrik Buchholtz and Edward Morehouse. Varieties of Cubical Sets. In *Relational and Algebraic Methods in Computer Science - 16th International Conference, RAMiCS 2017, Lyon, France, May 15-18, 2017, Proceedings*, pages 77–92, 2017. doi:10.1007/978-3-319-57418-9_5.

- 14 Evan Cavallo and Robert Harper. Higher inductive types in cubical computational type theory. *PACMPL*, 3(POPL):1:1–1:27, 2019. doi:10.1145/3290314.
- 15 Evan Cavallo and Robert Harper. Parametric Cubical Type Theory, 2019. arXiv:1901.00489.
- 16 James Cheney. A dependent nominal type theory. *Logical Methods in Computer Science*, 8(1), 2012. doi:10.2168/LMCS-8(1:8)2012.
- 17 Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In *21st International Conference on Types for Proofs and Programs, TYPES 2015, May 18-21, 2015, Tallinn, Estonia*, pages 5:1–5:34, 2015. doi:10.4230/LIPIcs.TYPES.2015.5.
- 18 Thierry Coquand, Simon Huber, and Anders Mörtberg. On Higher Inductive Types in Cubical Type Theory. In *LICS 2018, Oxford, UK, July 9-12, 2018*, 2018. doi:10.1145/3209108.3209197.
- 19 Neil Ghani, Patricia Johann, Fredrik Nordvall Forsberg, Federico Orsanigo, and Tim Revell. Bifibrational Functorial Semantics of Parametric Polymorphism. *Electr. Notes Theor. Comput. Sci.*, 319:165–181, 2015. doi:10.1016/j.entcs.2015.12.011.
- 20 Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*, volume 36 of *Oxford Logic Guides*, pages 83–111. Oxford Univ. Press, New York, 1998.
- 21 Guilhem Moulin. *Internalizing Parametricity*. PhD thesis, Chalmers University of Technology, Gothenburg, Sweden, 2016. URL: <https://research.chalmers.se/en/publication/235758>.
- 22 Andreas Nuyts and Dominique Devriese. Degrees of Relatedness: A Unified Framework for Parametricity, Irrelevance, Ad Hoc Polymorphism, Intersections, Unions and Algebra in Dependent Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 779–788, 2018. doi:10.1145/3209108.3209119.
- 23 Andreas Nuyts, Andrea Vezzosi, and Dominique Devriese. Parametric quantifiers for dependent type theory. *PACMPL*, 1(ICFP):32:1–32:29, 2017. doi:10.1145/3110276.
- 24 Ian Orton and Andrew M. Pitts. Axioms for Modelling Cubical Type Theory in a Topos. *Logical Methods in Computer Science*, 14(4), 2018. doi:10.23638/LMCS-14(4:23)2018.
- 25 John C. Reynolds. Types, Abstraction and Parametric Polymorphism. In *IFIP Congress*, pages 513–523, 1983.
- 26 Emily Riehl. On the directed univalence axiom. Talk slides, AMS Special Session on Homotopy Type Theory, Joint Mathematics Meetings, January 2018. URL: <http://www.math.jhu.edu/~eriehl/JMM2018-directed-univalence.pdf>.
- 27 Emily Riehl and Michael Shulman. A type theory for synthetic ∞ -categories. *Higher Structures*, 1(1):116–193, 2017. URL: https://journals.mq.edu.au/index.php/higher_structures/article/view/36.
- 28 Kristina Sojakova and Patricia Johann. A General Framework for Relational Parametricity. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 869–878, 2018. doi:10.1145/3209108.3209141.
- 29 The RedPRL Development Team. redtt, 2018. URL: <https://github.com/RedPRL/redtt>.
- 30 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. <https://homotopytypetheory.org/book>, Institute for Advanced Study, 2013.
- 31 Floris van Doorn. *On the Formalization of Higher Inductive Types and Synthetic Homotopy Theory*. PhD thesis, Carnegie Mellon University, 2018.
- 32 Vladimir Voevodsky. The equivalence axiom and univalent models of type theory, 2014. Talk at CMU on February 4, 2010. arXiv:1402.5556.
- 33 Philip Wadler. Theorems for Free! In *FPCA 1989, London, UK, September 11-13, 1989*, pages 347–359, 1989. doi:10.1145/99370.99404.

Unifying Cubical Models of Univalent Type Theory

Evan Cavallo 

School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
ecavallo@cs.cmu.edu

Anders Mörtberg

School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA
Department of Mathematics, Stockholm University, Sweden
anders.mortberg@math.su.se

Andrew W Swan

Institute for Logic, Language and Computation, University of Amsterdam, The Netherlands
a.w.swan@uva.nl

Abstract

We present a new constructive model of univalent type theory based on cubical sets. Unlike prior work on cubical models, ours depends neither on diagonal cofibrations nor connections. This is made possible by weakening the notion of fibration from the cartesian cubical set model, so that it is not necessary to assume that the diagonal on the interval is a cofibration. We have formally verified in **Agda** that these fibrations are closed under the type formers of cubical type theory and that the model satisfies the univalence axiom. By applying the construction in the presence of diagonal cofibrations or connections and reversals, we recover the existing cartesian and De Morgan cubical set models as special cases. Generalizing earlier work of Sattler for cubical sets with connections, we also obtain a Quillen model structure.

2012 ACM Subject Classification Theory of computation → Constructive mathematics; Theory of computation → Type theory

Keywords and phrases Cubical Set Models, Cubical Type Theory, Homotopy Type Theory, Univalent Foundations

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.14

Funding *Evan Cavallo*: This author was supported by the Air Force Office of Scientific Research through MURI grant FA9550-15-1-0053. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the AFOSR.

Acknowledgements We are grateful to Carlo Angiuli, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, Daniel R. Licata, Andrew Pitts and Jon Sterling for helpful comments and remarks on earlier versions of this work. The first two authors are also thankful to Mathieu Anel and Steve Awodey for their illuminating lectures on homotopical algebra in the CMU HoTT seminar.

1 Introduction

Cubical set models provide a constructive justification for Voevodsky’s univalence axiom and higher inductive types, as introduced in Homotopy Type Theory and Univalent Foundations (HoTT/UF) [38]. In this paper we develop a general axiomatization encompassing many existing cubical set models, allowing us to better understand the relationship between them and prove results about the entire class of models simultaneously.

The first model of HoTT/UF was developed by Voevodsky using Kan simplicial sets [26] and relies crucially on classical logic [9]. A major source of open problems in HoTT/UF has been the quest for constructive models; besides recent progress on a constructive variation of the Kan simplicial set model [23], the most fruitful approaches have been based on cubical



© Evan Cavallo, Anders Mörtberg, and Andrew W Swan;
licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 14; pp. 14:1–14:17



Leibniz International Proceedings in Informatics

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

sets. This was pioneered by the Bezem, Coquand and Huber (BCH) model [7, 8], which uses presheaves on the *symmetric monoidal cube category*. These cubical sets have degeneracy and face maps, but it is not possible to take the diagonal face of a square. An important feature of cubical sets, relative to simplicial sets, is that the product of representable cubical sets is again representable. This makes it possible to represent n -dimensional terms as ordinary terms in a context of n variables, each ranging over the interval object \mathbb{I} . The lack of diagonals in the BCH model corresponds to a lack of contraction for these contexts; the BCH model is *substructural*. This complicates giving a type-theoretic presentation; more fundamentally, it is unclear how to formulate and construct higher inductive types.

A natural approach, then, is to instead allow diagonals and study *cartesian cubical sets*, which model structural interval contexts. The base category here has a compact description as the free finite product category on an interval object [4, 29]. Cartesian cubical sets are hence better-suited as a basis for cubical type theory, and they are known to support higher inductive types. However, constructing univalent universes was an open problem for many years. The difficulties in modeling univalent universes motivated Cohen, Coquand, Huber and Mörtberg (CCHM) [15] to consider a cube category with even more structure, namely connections (\wedge and \vee) and an involutive reversal operation (\neg) satisfying the axioms of a De Morgan algebra. Using these additional operations, they gave the first cubical set model of univalent type theory with higher inductive types, as well as the first cubical type theory. It was later observed by Orton and Pitts (OP) [28] that the CCHM constructions do not require the full structure of a De Morgan algebra; a so-called “connection algebra” suffices. As a special case, there is a cubical category where the connection algebra is the free bounded distributive lattice. We call the resulting presheaf category *Dedekind cubical sets*, following Awodey, as the number of elements of $\text{Hom}(\mathbb{I}^n, \mathbb{I})$ are the Dedekind numbers [5]. Angiuli, Favonia, and Harper (AFH) [3] showed that that a model of HoTT/UF could also be developed in cartesian cubical sets *without* connections or reversals; their computational model was then adapted to an Orton-Pitts style construction by Angiuli et al. (ABCFL) [2].

In short, a wide variety of cube categories give rise to models of univalent type theory. Moreover, the underlying cube category is not the only parameter: one must also formulate *Kan composition*, i.e., choose a class of *fibrations*. Kan composition, a cubical analogue of the lifting condition in Kan simplicial sets, ensures that *Path* types induce a notion of equality. A representative special case of composition is *coercion*. Given a type A that depends on a dimension variable $i : \mathbb{I}$, coercion establishes a relationship between the elements of $A(r/i)$ and $A(s/i)$ for various $r, s : \mathbb{I}$. The nature of this relationship varies from model to model. In CCHM, the simplest case, coercion provides a map $\text{coe}_{i,A}^{0 \rightarrow 1} : A(0/i) \rightarrow A(1/i)$. In AFH, on the other hand, there is an operation $\text{coe}_{i,A}^{r \rightarrow s} : A(r/i) \rightarrow A(s/i)$ for *every* $r, s : \mathbb{I}$, together with an equation $\text{coe}_{i,A}^{r \rightarrow r} a = a : A(r/i)$. Other model constructions use intermediate points between these two extremes. For example, OP include $0 \rightarrow 1$ and $1 \rightarrow 0$. A more expressive cube category can compensate for a more limited form of coercion; in CCHM, coercions $\varepsilon \rightarrow s$ and $r \rightarrow \varepsilon$ for $\varepsilon : \{0, 1\}$ are derivable from the primitive $0 \rightarrow 1$ coercion.

In its general form, Kan composition coerces a cube while preserving some part of its boundary, a generalization necessary in order to derive coercion for *Path* types. The choice of allowable boundary shapes is a third parameter; from the model categorical perspective, it corresponds to a choice of *generating cofibrations*. In CCHM cubical sets, a boundary is specified by a collection of (conjunctions of) faces of the form $(r = 0)$ or $(r = 1)$. For cartesian cubes, AFH took the crucial step of also including $(r = s)$ boundary constraints, corresponding to diagonal faces of cubes. Model categorically, this corresponds to including the diagonal on the interval as a generating cofibration, i.e. to assume *diagonal cofibrations*.

■ **Table 1** Varieties of cubical models of HoTT/UF.

	Diagonals	Additional structure	Kan operations	Diagonal cofibrations
BCH			$0 \rightarrow r, 1 \rightarrow r$	
CCHM	✓	\wedge, \vee, \neg (De Morgan)	$0 \rightarrow 1$	
Dedekind	✓	\wedge, \vee (distributive lattice)	$0 \rightarrow 1, 1 \rightarrow 0$	
OP	✓	\wedge, \vee (connection algebra)	$0 \rightarrow 1, 1 \rightarrow 0$	
AFH/ABCFHL	✓		$r \rightarrow s$	✓

We collect the existing cubical set models in Table 1. As a general rule, these constructions can still be conducted in a setting with additional structure. For example, both the CCHM and ABCFHL model constructions can both be carried out in cubical sets with connections, reversals, and diagonal cofibrations. (The exception is BCH, which apparently relies crucially on the *absence* of diagonal maps.) The constructions produce the same notions of fibration where they are mutually applicable, as is observed for the CCHM and ABCFHL models in [2, Sec. 3.4]. What is lacking, however, is a *single* construction that applies in all cases.

Contributions

Our main contribution is a unification of the structural cubical models (i.e., all but BCH) as instances of a single construction. This is achieved by axiomatizing a class of models in the internal language style of Orton and Pitts [28], based on a “weak” variation of cartesian Kan composition. This notion of fibration specializes to the AFH definition in the presence of diagonal cofibrations (Section 2.3.1) and to the CCHM definition in the presence of connections and reversals (Section 2.3.2). The “weak” fibrations are closed under basic type formers (Section 2.4), Glue types (Section 2.5), and fibrant univalent universes (Section 2.6), thus give rise to a model of HoTT/UF. Furthermore, we obtain algebraic weak factorization systems of *cofibrations and trivial fibrations* (Section 3.2) and of *trivial cofibrations and fibrations* (Section 3.3). Finally, we verify that a theorem of Sattler [32, Thm. 2.8] applies, allowing us to obtain a model structure (Section 3.4) from the factorization systems.

2 A general axiomatization

Following Orton and Pitts [28], we construct models of cubical type theory from locally cartesian closed categories \mathcal{C} : we describe a collection of axioms in the internal language of such categories, then use the language as a tool to show that any category satisfying the axioms induces a class of fibrations closed under various type formers. Rather than relying on an impredicative universe of propositions, as Orton and Pitts do, we follow Licata, Orton, Pitts and Spitters (LOPS) [27] and work in a predicative theory. We use `Agda` [1] extended with postulates for function extensionality and uniqueness of identity proofs to simulate the internal type theory of a locally cartesian closed category.¹

We adopt `Agda`’s (ultimately `Nuprl`’s) syntax here, writing $(x : A) \rightarrow B$ for dependent and $A \rightarrow B$ for non-dependent functions. We assume a non-cumulative hierarchy of universes $\mathcal{U}_0 : \mathcal{U}_1 : \dots$; here, we leave levels implicit and write \mathcal{U} for simplicity, but they are explicit in the formalization. Among `Agda`’s inductive types, we need identity types (written $u = v$

¹ The formalization and additional material can be found at <https://github.com/mortberg/gen-cart>. For a summary of where all of the results in the paper can be found, see <https://github.com/mortberg/gen-cart/blob/master/agda/unifying-summary.agda>.

14:4 Unifying Cubical Models of Univalent Type Theory

and with a single constructor `refl`), an empty type $\perp : \mathcal{U}$, and sum types $A \uplus B$ (with constructors `inl` and `inr`). We write $\Sigma(x : A), B$ for dependent and $A \times B$ for non-dependent product types. Following HoTT/UF, we define the type of (homotopy) propositions as $\mathbf{hProp} \triangleq \Sigma(A : \mathcal{U}), (x y : A) \rightarrow x = y$. We assume a propositional truncation operation $\|-\| : \mathcal{U} \rightarrow \mathbf{hProp}$ universally approximating any type as an \mathbf{hProp} . We then define disjunction $P \vee Q$ of propositions P and Q as the propositional truncation $\|P \uplus Q\|$. The negation of a type $\neg A$ is defined as $A \rightarrow \perp$; this is always a proposition.

This type theory can be interpreted in any presheaf topos [25], in particular the various cubical and simplicial set categories, assuming enough Grothendieck universes. The standard example throughout the paper is the category of cartesian cubical sets.

2.1 The interval and Path types

The axiomatic requirements on \mathcal{C} begin with an interval type $\mathbb{I} : \mathcal{U}$ with endpoints $0 : \mathbb{I}$ and $1 : \mathbb{I}$. We require \mathbb{I} to be connected (**ax**₁) and $0, 1$ to be distinct (**ax**₂).

$$\begin{aligned} \mathbf{ax}_1 : (P : \mathbb{I} \rightarrow \mathcal{U}) \rightarrow ((i : \mathbb{I}) \rightarrow P \text{ } i \uplus \neg(P \text{ } i)) \rightarrow ((i : \mathbb{I}) \rightarrow P \text{ } i) \uplus ((i : \mathbb{I}) \rightarrow \neg(P \text{ } i)) \\ \mathbf{ax}_2 : \neg(0 = 1) \end{aligned}$$

Given $A : \mathbb{I} \rightarrow \mathcal{U}$, we define the type of *paths* in A as $\mathbf{Path}(A) \triangleq (i : \mathbb{I}) \rightarrow A \text{ } i$. Given $a : A \ 0$ and $b : A \ 1$, we write $a \sim b \triangleq \Sigma(p : \mathbf{Path}(A)), (p \ 0 = a) \times (p \ 1 = b)$. Given $p : a \sim b$ and $r : \mathbb{I}$, we write $p \ @ \ r$ for the application of `fst` p to r , which satisfies $p \ @ \ 0 = a$ and $p \ @ \ 1 = b$.

2.2 Cofibrant propositions

Next, we assume a universe à la Tarski of generating cofibrant propositions $\Phi : \mathcal{U}$ supporting the following operations. We write $[_]$: $\Phi \rightarrow \mathbf{hProp}$ for the decoding function and stipulate that it interprets the code constructors appropriately.

$$\begin{aligned} (_ \approx 0) : \mathbb{I} \rightarrow \Phi & & \mathbf{ax}_3 : (i : \mathbb{I}) \rightarrow [(i \approx 0)] = (i = 0) \\ (_ \approx 1) : \mathbb{I} \rightarrow \Phi & & \mathbf{ax}_4 : (i : \mathbb{I}) \rightarrow [(i \approx 1)] = (i = 1) \\ \vee : \Phi \rightarrow \Phi \rightarrow \Phi & & \mathbf{ax}_5 : (\varphi \ \psi : \Phi) \rightarrow [\varphi \vee \psi] = [\varphi] \vee [\psi] \end{aligned}$$

Note that we have two bottom elements, $(0 \approx 1)$ and $(1 \approx 0)$. The decoding of these imply each other, but we need not assume they are equal. The same holds for the two top elements $(0 \approx 0)$ and $(1 \approx 1)$. Note that for all $A : \mathcal{U}$, we have $\mathbf{elim}_\perp : [(0 \approx 1)] \rightarrow A$ by **ax**₂.

► **Remark 1.** If \mathcal{C} is a topos, we can take Φ to be the subobject classifier Ω . To obtain a constructive presheaf model, we can instead take Φ to be the subobject of Ω of sieves with decidable image at each stage. However, the axiomatization of Φ does not presume the existence of a subobject classifier; nor does it require that inter-derivable cofibrations are equal. This is similar to the approach taken in [2, 27], where $\Phi \triangleq \Sigma(A : \mathcal{U}), \mathbf{cof} \ A$ is specified by a predicate $\mathbf{cof} : \mathcal{U} \rightarrow \mathcal{U}$ on types. However, our variation requires that Φ is a *small* type, which is needed to construct identity types while preserving universe level.

A *partial element* of A is a term $f : [\varphi] \rightarrow A$. Given such a partial element f and an element $x : A$, we define the *extension* relation $f \nearrow x \triangleq (u : [\varphi]) \rightarrow f \ u = x$, so that $f \nearrow x$ is the type of proofs that the partial element f extends to the total element x . Following [15], we write $A[\varphi \mapsto f] \triangleq \Sigma(x : A), f \nearrow x$ for the type of all elements of A extending f . Given a partial path $f : [\varphi] \rightarrow \mathbf{Path}(A)$ and $r : \mathbb{I}$, we write $f \cdot r \triangleq \lambda u. f \ u \ r : [\varphi] \rightarrow A \ r$.

This completes the basic set of axioms, which will suffice to interpret the Σ -, Π -, \mathbf{Path} types and basic datatypes. We defer the introduction of two final axioms to Section 2.5, where we will need them to interpret (strict) Glue types.

2.3 Fibration structures

Using the interval and the universe of cofibrant propositions, we can now define our notion of fibration structure, a weaker variation on the fibration structures used in [2, 3].

► **Definition 2** (Weak composition). *Given $r : \mathbb{I}$, $A : \mathbb{I} \rightarrow \mathcal{U}$, $\varphi : \Phi$, $f : [\varphi] \rightarrow \text{Path}(A)$ and $x_0 : (A \ r)[\varphi \mapsto f \cdot i]$, a weak composition structure is given by two operations*

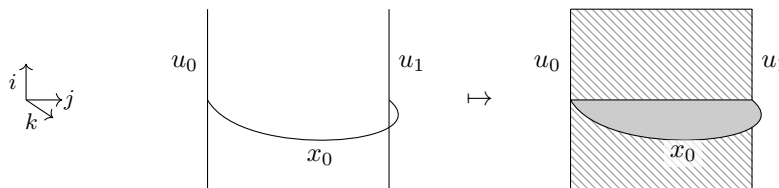
$$\text{wcom} : (s : \mathbb{I}) \rightarrow (A \ s)[\varphi \mapsto f \cdot s] \quad \underline{\text{wcom}} : \text{fst}(\text{wcom } r) \sim \text{fst } x_0$$

satisfying $(i : \mathbb{I}) \rightarrow f \cdot r \nearrow \underline{\text{wcom}} @ i$. We write $\text{WComp } r \ A \ \varphi \ f \ x_0$ for the type of such weak composition structures, i.e.,

$$\text{WComp } r \ A \ \varphi \ f \ x_0 \triangleq \Sigma(\text{wcom} : \dots), \Sigma(\underline{\text{wcom}} : \dots), (i : \mathbb{I}) \rightarrow f \cdot r \nearrow \underline{\text{wcom}} @ i$$

In contrast with [2, 3], we do not require that the equality $\text{wcom } r \ A \ \varphi \ f \ x_0 \ r = x_0$ holds strictly. Instead, the $\underline{\text{wcom}}$ operation enforces the equation up to a path constant on φ . We say that $\text{wcom } r \ A \ \varphi \ f \ x_0 \ s$ composes $r \rightarrow s$ in A , and refer to f as the *tube* and x_0 as the *cap* of the composition. We refer to $\underline{\text{wcom}}$ as the “cap path”, as it relates $\text{wcom } r \ A \ \varphi \ f \ x_0 \ r$ to the cap x_0 .

► **Example 3.** We can illustrate the above choice of terminology with the following example. The composition problem is given by the tube u_0 and u_1 at $(j \approx 0)$ and $(j \approx 1)$ together with a cap x_0 at $(i \approx r)$. The composition from r to i is the interior of the square on the right, while the cap path is the gray path connecting the composition at r to x_0 .



► **Definition 4** (Weak fibrations and fibration structures). *A weak fibration (A, α) over $\Gamma : \mathcal{U}$ is a family $A : \Gamma \rightarrow \mathcal{U}$ equipped with a fibration structure $\alpha : \text{isFib } A$, where*

$$\text{isFib } A \triangleq (r : \mathbb{I})(p : \mathbb{I} \rightarrow \Gamma)(\varphi : \Phi)(f : [\varphi] \rightarrow (i : \mathbb{I}) \rightarrow A(p \ i))(x_0 : A(p \ r)[\varphi \mapsto f \cdot r]) \rightarrow \text{WComp } r \ (A \circ p) \ \varphi \ f \ x_0$$

We write $\text{Fib } \Gamma \triangleq \Sigma(A : \Gamma \rightarrow \mathcal{U}), \text{isFib } A$ for the type of weak fibrations over Γ . As in [28, Def. 5.8], we obtain a *category with families* (CwF) [21] where the families over $\Gamma : \mathcal{U}$ are $(A, \alpha) : \text{Fib } \Gamma$ and elements of such a family are dependent functions in $(x : \Gamma) \rightarrow A \ x$. Given $P : \text{Fib } \Gamma$ and $\sigma : \Delta \rightarrow \Gamma$, we write $P[\sigma] : \text{Fib } \Delta$ for the reindexing of P along σ .

► **Remark 5.** When discussing the model structure in Section 3.4, we will use the term *fibration* for the usual external notion of a map that has the right lifting property against trivial cofibrations. Whenever this overloading of terminology might be confusing we use the terms *weak fibration* and *fibration structure* when referring to the internal notions.

Given $\alpha : \text{isFib } A$, $s : \mathbb{I}$ and r, p, φ, f and x_0 as in Definition 4, we introduce the following more readable notation for the composites provided by α .

$$\text{wcom}_\alpha^{r \rightarrow s} p[\varphi \mapsto f] x_0 \triangleq \text{fst}(\text{fst}(\alpha \ r \ p \ \varphi \ f \ x_0) \ s) : A \ (p \ s)$$

$$\underline{\text{wcom}}_\alpha^r p[\varphi \mapsto f] x_0 \triangleq \text{fst}(\text{snd}(\alpha \ r \ p \ \varphi \ f \ x_0)) : (\text{wcom}_\alpha^{r \rightarrow r} p[\varphi \mapsto f] x_0) \sim \text{fst } x_0$$

14:6 Unifying Cubical Models of Univalent Type Theory

Given $\varphi, \psi : \Phi$, we follow [15] and write $[\varphi \mapsto f, \psi \mapsto g] : [\varphi \vee \psi] \rightarrow A$ for the union of partial elements $f : [\varphi] \rightarrow A$ and $g : [\psi] \rightarrow A$ that agree where they are both defined, i.e. such that $\forall (u : [\varphi]) (v : [\psi]). f u = g v$. This generalizes directly to $[\varphi_1 \mapsto f_1, \dots, \varphi_n \mapsto f_n]$.

We say that a proposition $A : \mathbf{hProp}$ is *cofibrant* if it is logically equivalent to the decoding of a generating cofibrant proposition, i.e. $\mathbf{isCofProp} A \triangleq \Sigma(\varphi : \Phi), A \leftrightarrow [\varphi]$. When $r, s : \mathbb{I}$ are such that $(r = s)$ is cofibrant, we will be able to “improve” weak composition $r \rightarrow s$ to obtain a strict composition that is exactly equal to its cap when $r = s$.

► **Definition 6** (Strict composition). *Given $r : \mathbb{I}$, $A : \mathbb{I} \rightarrow \mathcal{U}$, $\varphi : \Phi$, $f : [\varphi] \rightarrow \text{Path}(A)$ and $x_0 : (A r)[\varphi \mapsto f \cdot i]$, a strict composition structure is given by an operation*

$$\mathbf{scom} : (s : \mathbb{I}) \rightarrow \mathbf{isCofProp}(r = s) \rightarrow (A s)[\varphi \mapsto f \cdot s]$$

satisfying $\text{fst}(\mathbf{scom} r c) = \text{fst} x_0$ for all $c : \mathbf{isCofProp}(r = r)$.

We will leave the argument $\mathbf{isCofProp}(r = s)$ implicit. Writing $\mathbf{SComp} r A \varphi f x_0$ for the type of strict composition operations on A , we define strict fibrations as follows.

► **Definition 7** (Strict fibrations). *A strict fibration (A, α) over $\Gamma : \mathcal{U}$ is a family $A : \Gamma \rightarrow \mathcal{U}$ equipped with a strict fibration structure $\alpha : \mathbf{isSFib} A$, where*

$$\begin{aligned} \mathbf{isSFib} A \triangleq & (r : \mathbb{I})(p : \mathbb{I} \rightarrow \Gamma)(\varphi : \Phi)(f : [\varphi] \rightarrow (i : \mathbb{I}) \rightarrow A(p i))(x_0 : A(p r)[\varphi \mapsto f \cdot r]) \\ & \rightarrow \mathbf{SComp} r (A \circ p) \varphi f x_0 \end{aligned}$$

► **Lemma 8** (Strictification). *Given $\Gamma : \mathcal{U}$ and $A : \Gamma \rightarrow \mathcal{U}$, there is a map $\mathbf{isFib} A \rightarrow \mathbf{isSFib} A$.*

Proof. Given $\alpha : \mathbf{isFib} A$ and r, p, φ, f and x_0 as in Definition 7, let

$$w \triangleq \mathbf{wcom}_{\alpha}^{r \rightarrow s} p [\varphi \mapsto f] x_0 \qquad \underline{w} \triangleq \mathbf{wcom}_{\alpha}^r p [\varphi \mapsto f] x_0$$

Given $s : \mathbb{I}$, we define the following term that corrects the $(r = s)$ face of w using \underline{w} .

$$\mathbf{scom} s \triangleq \mathbf{wcom}_{\alpha}^{0 \rightarrow 1} (\lambda _ . p s) [\varphi \mapsto \lambda u _ . f u s, (r = s) \mapsto \lambda _ . i. \underline{w} @ i] w \quad \blacktriangleleft$$

In particular, as $(r = \varepsilon)$ and $(\varepsilon = r)$ are always cofibrant for $\varepsilon : \{0, 1\}$, we have strict composition operations $\varepsilon \rightarrow r$ and $r \rightarrow \varepsilon$ in any fibration. Defining $\bar{0} \triangleq 1$ and $\bar{1} \triangleq 0$, we note that the weak compositions $\varepsilon \rightarrow \bar{\varepsilon}$ are already strict, as the cap condition is vacuous.

2.3.1 AFH fibrations

We now compare our definition of fibration to that of existing *cartesian* cubical type theories and models. A key feature of these is the use of diagonal cofibrations, which correspond to an operation $(_ \approx _) : \mathbb{I} \rightarrow \mathbb{I} \rightarrow \Phi$ decoding as follows.

$$\mathbf{ax}_{\Delta} : (r s : \mathbb{I}) \rightarrow [(r \approx s)] = (r = s)$$

The form of fibration used in these models was originally proposed by Coquand [16], but it was initially unclear how to model univalent universes. AFH observed that the problems could be dealt with by introducing diagonal cofibrations, and used them to give a complete computational semantics of univalent type theory (we hence refer to these as “AFH fibrations”). These ideas were then adapted in ABCFHL to give an Orton-Pitts style model construction.

► **Definition 9** (AFH composition). *Given $r : \mathbb{I}$, $A : \mathbb{I} \rightarrow \mathcal{U}$, $\varphi : \Phi$, $f : [\varphi] \rightarrow \text{Path}(A)$ and $x_0 : (A r)[\varphi \mapsto f \cdot i]$, an AFH composition structure is given by $\text{com} : (s : \mathbb{I}) \rightarrow (A s)[\varphi \mapsto f \cdot s]$ satisfying $\text{fst}(\text{com } r) = \text{fst } x_0$. We write $\text{AFHComp } r A \varphi f x_0$ for the type of such AFH composition structures, and write*

$$\begin{aligned} \text{isAFHFib } A \triangleq & (r : \mathbb{I})(p : \mathbb{I} \rightarrow \Gamma)(\varphi : \Phi)(f : [\varphi] \rightarrow (i : \mathbb{I}) \rightarrow A(p i)) \\ & (x_0 : A(p r)[\varphi \mapsto f \cdot r]) \rightarrow \text{AFHComp } r (A \circ p) \varphi f x_0 \end{aligned}$$

When isAFHFib is taken as the definition of fibration, it seems that diagonal cofibrations are crucial to construct fibrant univalent universes of fibrant types. Specifically, they are needed to ensure that composition in Glue/V types and the universe satisfies the strict cap condition. In the presence of diagonal cofibrations, our definition of fibration coincides with isAFHFib .

► **Theorem 10.** *Given $\Gamma : \mathcal{U}$ and $A : \Gamma \rightarrow \mathcal{U}$, we have $\text{isAFHFib } A$ iff we have $\text{isFib } A$.²*

Proof. Any AFH composition structure induces a weak composition structure, as any equality can be turned into a path. For the converse direction, apply Lemma 8 with \mathbf{ax}_Δ . ◀

► **Remark 11.** Awodey [6] has formulated a categorical notion of *unbiased fibrations* and shown that this coincides with AFH fibrations; it thus also coincides with weak composition in the presence of diagonal cofibrations.

2.3.2 CCHM fibrations

Next, we compare with the CCHM definition of fibration. Following Orton and Pitts [28], we assume operations $\sqcap, \sqcup : \mathbb{I} \rightarrow \mathbb{I} \rightarrow \mathbb{I}$ satisfying the axioms of a *connection algebra*.

$$\begin{aligned} \mathbf{ax}_\sqcap : & (r : \mathbb{I}) \rightarrow (0 \sqcap r = 0 = r \sqcap 0) \wedge (1 \sqcap r = r = r \sqcap 1) \\ \mathbf{ax}_\sqcup : & (r : \mathbb{I}) \rightarrow (0 \sqcup r = r = r \sqcup 0) \wedge (1 \sqcup r = 1 = r \sqcup 1) \end{aligned}$$

► **Remark 12.** A connection algebra is weaker than the De Morgan algebra used in CCHM: there is no reversal $\neg : \mathbb{I} \rightarrow \mathbb{I}$ and the connections need not form a distributive lattice. Thus, Orton and Pitts [28] obtain a construction that applies to both CCHM and Dedekind cubical sets, compensating for the lack of reversals by parametrizing the composition operation by $\varepsilon : \{0, 1\}$. Following Orton and Pitts, we continue to call this “CCHM composition” despite the superficial difference from the operation defined in [15].

► **Definition 13** (CCHM composition). *Given $\varepsilon : \{0, 1\}$, $A : \mathbb{I} \rightarrow \mathcal{U}$, $\varphi : \Phi$, $f : [\varphi] \rightarrow \text{Path}(A)$ and $x_0 : (A \varepsilon)[\varphi \mapsto f \cdot i]$, a CCHM composition structure is a term $\text{com} : (A \bar{\varepsilon})[\varphi \mapsto f \cdot \bar{\varepsilon}]$. We write $\text{CCHMComp } \varepsilon A \varphi f x_0$ for the type of such CCHM composition structures, and*

$$\begin{aligned} \text{isCCHMFib } A \triangleq & (\varepsilon : \{0, 1\})(p : \mathbb{I} \rightarrow \Gamma)(\varphi : \Phi)(f : [\varphi] \rightarrow (i : \mathbb{I}) \rightarrow A(p i)) \\ & (x_0 : A(p \varepsilon)[\varphi \mapsto f \cdot r]) \rightarrow \text{CCHMComp } \varepsilon (A \circ p) \varphi f x_0 \end{aligned}$$

A key result in CCHM is that connections and composition $0 \rightarrow 1$ suffice to derive composition $0 \rightarrow r$ (i.e. Kan filling). The following result shows that we can in fact derive all of the cartesian composition operations, except for the strict equality for $r \rightarrow r$. This clarifies the relationship between CCHM and AFH composition. As CCHM only requires compositions $\varepsilon \rightarrow \bar{\varepsilon}$, diagonal cofibrations are not needed for Glue types and the universe.

² This is already observed for weak coercion in [2, Sec. 2.7].

14:8 Unifying Cubical Models of Univalent Type Theory

► **Theorem 14.** *Given $\Gamma : \mathcal{U}$ and $A : \Gamma \rightarrow \mathcal{U}$, we have $\text{isCCHMFib } A$ iff we have $\text{isFib } A$.*

Proof. We can go from $\text{isFib } A$ to $\text{isCCHMFib } A$ by simply instantiating r with ε and s with $\bar{\varepsilon}$. For the other direction, let r, p, φ, f and x_0 be as in Definition 4. First, we define the following term, which composes from $A (p r)$ to $A (p (j \wedge r))$ for any $j : \mathbb{I}$.

$$q j \triangleq \text{com}_\alpha^{1 \rightarrow 0} (\lambda i. p ((j \vee i) \wedge r)) \left[\begin{array}{l} \varphi \quad \mapsto \lambda u i. f u ((j \vee i) \wedge r) \\ (j = 1) \mapsto \lambda _ _. x_0 \end{array} \right] x_0$$

We can then define weak composition to $s : \mathbb{I}$.

$$\text{wcom } s \triangleq \text{com}_\alpha^{0 \rightarrow 1} (\lambda i. p (i \wedge s)) [\varphi \mapsto \lambda u i. f u (i \wedge s), (0 \approx 1) \mapsto \text{elim}_\perp] (q 0)$$

The cap path is defined as follows.

$$\underline{\text{wcom}} \triangleq \lambda (j : \mathbb{I}). \text{com}_\alpha^{0 \rightarrow 1} (\lambda i. p ((j \vee i) \wedge r)) \left[\begin{array}{l} \varphi \quad \mapsto \lambda u i. f u ((j \vee i) \wedge r) \\ (j = 1) \mapsto \lambda _ _. x_0 \end{array} \right] (q j) \quad \blacktriangleleft$$

2.4 Fibration structures for basic type formers

The collection of fibrations is closed under all of the basic type formers of cubical type theory: Σ -, Π -, Path types and any basic datatypes that \mathcal{C} supports. The arguments are very similar to those of [2, 3], but additional adjustments are necessary to compensate for the new weakness. We include the proof for Σ -types in order to illustrate this in detail.

► **Theorem 15** (Fibrant Σ -types). *Given $\Gamma : \mathcal{U}$, $A : \Gamma \rightarrow \mathcal{U}$, $B : (\Sigma(x : \Gamma), A x) \rightarrow \mathcal{U}$, we have*

$$\text{isFib}_\Sigma : \text{isFib } A \rightarrow \text{isFib } B \rightarrow \text{isFib } (\Sigma A B)$$

where $(\Sigma A B) x \triangleq \Sigma(a : A x), B(x, a)$.

Proof. Let $\alpha : \text{isFib } A$ and $\beta : \text{isFib } B$ and r, p, φ, f and x_0 be as in Definition 4. We first define the composite and cap path for the first components of the open box.

$$\begin{aligned} w_A i &\triangleq \text{wcom}_\alpha^{r \rightarrow i} p [\varphi \mapsto \lambda u j. \text{fst}(f u j)] (\text{fst } x_0) \\ \underline{w}_A &\triangleq \underline{\text{wcom}}_\alpha^r p [\varphi \mapsto \lambda u j. \text{fst}(f u j)] (\text{fst } x_0) \end{aligned}$$

To define the composite of the second components, we first adjust the type of the cap. For this, we use a strict composition $1 \rightarrow k$ in B , which is derivable from β per Lemma 8.

$$b k \triangleq \text{scom}_\beta^{1 \rightarrow k} (\lambda j. (p r, \underline{w}_A @ j)) [\varphi \mapsto \lambda u _. \text{snd}(f u r)] (\text{snd } x_0)$$

When k is 0, this is the corrected cap of our composition in B .

$$\begin{aligned} w_B &\triangleq \text{wcom}_\beta^{r \rightarrow s} (\lambda i. (p i, w_A i)) [\varphi \mapsto \lambda u i. \text{snd}(f u i)] (b 0) \\ \underline{w}_B &\triangleq \underline{\text{wcom}}_\beta^r (\lambda i. (p i, w_A i)) [\varphi \mapsto \lambda u i. \text{snd}(f u i)] (b 0) \end{aligned}$$

Composition in the pair type is then defined to be the pair $\text{wcom } s \triangleq (w_A s, w_B)$. For the cap path, we combine the cap path \underline{w}_B for the composition in B with the path b that relates $b 0$ to $\text{snd } x_0$ over \underline{w}_A .

$$c t \triangleq \text{wcom}_\beta^{1 \rightarrow 0} (\lambda j. (p r, \underline{w}_A @ j)) \left[\begin{array}{l} \varphi \quad \mapsto \lambda u _. \text{snd}(f u r) \\ (t = 0) \mapsto \lambda _ j. \underline{w}_B @ j \\ (t = 1) \mapsto \lambda _ _. \text{snd } x_0 \end{array} \right] (b t)$$

We then let $\underline{\text{wcom}} \triangleq \lambda (t : \mathbb{I}). (\underline{w}_A @ t, c t)$. ◀

The case for Π -types is similar to that of Σ -types: the proof roughly follows that of strict composition, but additional composites have to be inserted to mediate between composites and their caps. The proofs for `Path` types and natural numbers are essentially identical to those of [2, 3]. We omit the details here, but the interested reader may consult [13, Sec. 3] or our `Agda` formalization. It is also straightforward to verify that these definitions are stable under reindexing, so that we obtain a `CwF` that supports Σ -, Π - and `Path` types. This `CwF` also supports natural numbers if \mathcal{C} has a natural numbers object.

2.5 Glueing

Glue types were introduced in [15, Sec. 6] to unify the proofs that the universe of fibrant types is fibrant and univalent. This construction also occurs implicitly in the proof that the universe is univalent in the Kan simplicial set model [26, Thm. 3.4.1]. The construction of these types in the internal language was described in detail by Orton and Pitts [28, Sec. 6]. In this section we only briefly sketch their construction; apart from the proof of Theorem 17, there are no major differences.

► **Definition 16** (Glueing). *Given $\varphi : \Phi$, $A : [\varphi] \rightarrow \mathcal{U}$, $B : \mathcal{U}$ and $f : (x : [\varphi]) \rightarrow A \ x \rightarrow B$, we define $\text{Glue } \varphi \ A \ B \ f : \mathcal{U}$ as follows.*

$$\text{Glue } \varphi \ A \ B \ f \triangleq \Sigma(a : (x : [\varphi]) \rightarrow A \ x), \Sigma(b : B), (x : [\varphi]) \rightarrow f \ x \ (a \ x) = b$$

Elements of this type are thus pairs (a, b) where a is a partial element of A and b is an element of B such that f applied to a extends to b . When φ is \top , the `Glue` type is isomorphic to A . The `Glue` operator lifts to a fiberwise operation on families of types, which we also call `Glue`. To prove that it takes fibrations to fibrations, however, we must also require that f is an equivalence. There are various ways to express this; we follow Voevodsky and say that f is an equivalence when its fibers are contractible [38, 39]. We write $A \simeq B$ for the type of equivalences between A and B .

► **Theorem 17** (Fibrant Glue types). *Given $\Gamma : \mathcal{U}$, $\varphi : \Gamma \rightarrow \Phi$, $A : (x : \Gamma) \rightarrow [\varphi \ x] \rightarrow \mathcal{U}$, $B : \Gamma \rightarrow \mathcal{U}$ and $f : (x : \Gamma) \ (v : [\varphi \ x]) \rightarrow A \ x \ v \rightarrow B \ x$. If f has the structure of an equivalence then there is a function $\text{isFib}_{\text{Glue}} : \text{isFib } A \rightarrow \text{isFib } B \rightarrow \text{isFib } (\text{Glue } \varphi \ A \ B \ f)$.*

The proof of this theorem is a variation of the one of [2]; as with Σ -types, some additional compositions are needed to compensate for the weakness. We refer the interested reader to the detailed type theoretic presentation in [13, Sec. 4.2] and to the `Agda` formalization.

Note that the fibrancy of these types does not require any additional axioms. However, they are weaker than the `Glue` types of [15]: they are not strictly equal to A when φ is \top , only isomorphic. In order to prove univalence and fibrancy of the universe, we first need to strictify. Writing $A \cong B$ for the type of isomorphisms between A and B , we require the following *strictness axiom* (**ax₉** in [28]).

$$\mathbf{ax}_6 : (\varphi : \Phi) (A : [\varphi] \rightarrow \mathcal{U}) (B : \mathcal{U}) (s : (u : [\varphi]) \rightarrow A \ u \cong B) \rightarrow \\ \Sigma(B' : \mathcal{U}), \Sigma(s' : B' \cong B), (u : [\varphi]) \rightarrow (A \ u, s \ u) = (B', s')$$

Using this axiom, we can perform the same construction as in [28, Def. 6.1] and obtain a type `SGlue` $\varphi \ A \ B \ f$ that satisfies the desired equation strictly and is isomorphic to `Glue` $\varphi \ A \ B \ f$. We then transport the weak fibration structure from `Glue` to `SGlue` along this isomorphism. However, the weak composition operation that we obtain this way will not

14:10 Unifying Cubical Models of Univalent Type Theory

necessarily reduce to the composition operation of A when φ is \top . In order to correct this, we assume an operation $\forall : (\mathbb{I} \rightarrow \Phi) \rightarrow \Phi$ satisfying the following.

$$\mathbf{ax}_7 : (\varphi : \mathbb{I} \rightarrow \Phi) \rightarrow [\forall \varphi] = (i : \mathbb{I}) \rightarrow [\varphi i]$$

Using this axiom, we can perform the same “alignment” as in [28, Thm. 6.13] and obtain a weak fibration structure for \mathbf{SGLue} that reduces that of A when φ is \top .

2.5.1 Univalence

Voevodsky’s univalence axiom states that the canonical map $\mathbf{idtoequiv} : (A \sim B) \rightarrow (A \simeq B)$ is an equivalence. This formulation of univalence assumes a universe of (fibrant) types. As we have not yet constructed a universe, we instead define a variation of univalence that uses a primitive notion of lines between types. For $\Gamma : \mathcal{U}$ and $A, B : \mathbf{Fib} \Gamma$, we define

$$A \sim_{\mathcal{U}} B \triangleq \Sigma(P : \mathbf{Fib}(\Gamma \times \mathbb{I}), P[(\mathbf{id}, 0)] = A \times P[(\mathbf{id}, 1)] = B)$$

► **Theorem 18** (Univalence for $\sim_{\mathcal{U}}$). *We have $(A \sim_{\mathcal{U}} B) \simeq (\mathbf{fst} A \simeq \mathbf{fst} B)$.*

Proof. This is equivalent³ to the existence of a term $\mathbf{ua} : A \simeq B \rightarrow A \sim_{\mathcal{U}} B$ such that $\mathbf{idtoequiv} \circ \mathbf{ua} = \mathbf{id}$. The \mathbf{ua} term follows directly from \mathbf{SGLue} in the standard way [28, Thm. 7.2]. The inverse condition can be proven by unfolding the algorithm for weak composition in \mathbf{SGLue} , in analogy with [28, Thm. 7.3]. ◀

This model hence satisfies this variation of the univalence axiom. Following [27], we may also construct a universe and prove the standard formulation of the univalence axiom.

2.6 Fibrant univalent universes

The universe construction of LOPS [27] can be performed in a modal extension of type theory called *crisp type theory*. Andrea Vezzosi has developed an extension of \mathbf{Agda} with the crisp modality called $\mathbf{Agda}\text{-}\flat$. However, this was only recently incorporated into the standard version of \mathbf{Agda} , so we have not formally verified the content of this section.

A key component in the LOPS universe construction is a special feature of the interval in the various cubical set categories: it is *tiny*, i.e. exponentiation by it has a right adjoint. This is *not* true for Δ^1 , so the following theorem does not apply to Kan simplicial sets.

► **Theorem 19** (Universe construction). *If \mathbb{I} is tiny, then we can construct a universe \mathbf{U} with a fibration \mathbf{El} that is classifying in the sense of [27, Thm. 5.2].*

Proof. We need to check that the assumptions of [27, Thm. 5.2] are satisfied. First of all, the arguments of \mathbf{isFib} and \mathbf{WComp} can be rearranged to match [27, Def. 2.2]. We then need to check that axioms (1)–(4) in [27] hold. The first two are function extensionality and uniqueness of identity proofs, which we are assuming. The other two are disjointness of endpoints and that \perp is a cofibrant proposition, both of which follow from \mathbf{ax}_2 . ◀

We next need to show that this universe has a weak fibration structure, is closed under all of the type formers of cubical type theory, and satisfies the univalence axiom. This has been formalized in $\mathbf{Agda}\text{-}\flat$ for AFH fibrations in [2], and we do not expect any difficulty doing the

³ This was originally pointed out by Daniel R. Licata in <https://groups.google.com/forum/#!msg/homotopytypetheory/j2KBIvDw53s/YTDK4D0NFQAJ>.

same here, the only difference being the strictness of the cap equation. For a type theoretic proof that the universe is fibrant and univalent using the fibration structures in this paper, see [13, Sec. 4.3 and 4.4].

3 Model structures on cubical sets

We will now prove that our definition of fibration structures forms part of a Quillen model structure. This helps to clarify the relation between our definition and already established and well known definitions in homotopical algebra. We assume the reader is familiar with standard concepts in homotopical algebra such as model structures, algebraic weak factorization systems (awfs's), and the Leibniz adjunction. See e.g. [31] for these definitions.

Further details, including proofs of these results, are available in [14]. We have also defined the two factorization systems in *Agda* by postulating the existence of *W-types with reductions* [36], a simple class of (extensional) higher inductive types.

We will use some extra notational conventions for this section. We write $\delta_i : 1 \rightarrow \mathbb{I}$ for $i : \{0, 1\}$ for the endpoint inclusions. We use the subscript B when working with objects in a slice category \mathcal{C}/B . In particular, we have an interval object \mathbb{I}_B defined as the projection $\mathbb{I} \times B \rightarrow B$, with obvious endpoint maps $\delta_{Bi} : 1_B \rightarrow \mathbb{I}_B$.

3.1 Cofibrantly generated awfs's

To construct a model structure, we first need to define two weak factorization systems, one for *cofibrations and trivial fibrations* and one for *trivial cofibrations and fibrations*. In both cases, we will use the following definitions and theorems from [36] and [34].

► **Definition 20** ([36, Def. 6.1]). *Let m be a map in a slice category \mathcal{C}/I and let f be a map in another slice category \mathcal{C}/J . A family of lifting problems of m against f consists of an object K , together with maps $\sigma : K \rightarrow I$ and $\tau : K \rightarrow J$ and a lifting problem of $\sigma^*(m)$ against $\tau^*(f)$ in \mathcal{C}/K .*

We say m has the fibered left lifting property against f and f has the fibered right lifting property against m if every family of lifting problems has a diagonal filler.

A family of lifting problems K, σ, τ, p, q is universal if for any other family of lifting problems $K', \sigma', \tau', p', q'$, there is a unique map $t : K' \rightarrow K$ such that $\sigma' = t \circ \sigma$, $\tau' = t \circ \tau$, $p' = t^(p)$ and $q' = t^*(q)$.*

► **Proposition 21** ([34, Prop. 3.2.4], [36, Def. 6.2]). *Universal lifting problems exist.*

► **Proposition 22** ([34, Prop. 3.2.5]). *f has the fibered right lifting property against m iff the universal lifting problem has a filler.*

► **Definition 23.** *A fibered algebraic weak factorization system or fibered awfs consists of an algebraic weak factorization system (L_J, R_J) on each slice category \mathcal{C}/J preserved by reindexing (up to isomorphism).*

A fibered awfs is cofibrantly generated if there exists a map m in some slice category \mathcal{C}/I such that for each J and each map f in \mathcal{C}/J , R_J algebra structures on f correspond precisely to diagonal fillers of the universal lifting problem of m against f .

The following theorem will allow us to construct the two weak factorization systems of the model structure.

► **Theorem 24.** *Let m be a map in some slice category \mathcal{C}/I . The fibered awfs cofibrantly generated by m exists if either of the two conditions below are satisfied.*

14:12 Unifying Cubical Models of Univalent Type Theory

1. \mathcal{C} is an internal category of presheaves in a locally cartesian closed category with finite colimits, disjoint sums and W -types, and m is a locally decidable monomorphism.
2. \mathcal{C} is a IIW-pretopos (e.g. \mathcal{C} is a topos with natural number object), and it satisfies the axiom weakly initial set of covers (WISC).

Proof. If (1) holds, apply [36, Thm. 6.14], and if (2) holds, apply [36, Cor. 6.12]. ◀

3.2 Cofibration and trivial fibration awfs

We can view the cofibrant propositions $[-] : \Phi \rightarrow \mathbf{hProp}$ as a monomorphism $\top : \Phi_{\text{true}} \rightarrow \Phi$, where $\Phi_{\text{true}} \triangleq \Sigma(\varphi : \Phi), [\varphi] = \top$.

► **Definition 25** (Generating cofibrations). *Let $m : A \rightarrow B$ be a map in a slice category \mathcal{C}/I . We say m is a generating cofibration if either of the equivalent conditions below holds.*

1. $\sum_I m$ is a pullback of \top .
2. m is a pullback of $I^*(\top) : I^*(\Phi_{\text{true}}) \rightarrow I^*(\Phi)$ in \mathcal{C}/I .

► **Proposition 26.** *Generating cofibrations are closed under pullbacks and binary unions. Every isomorphism is a generating cofibration.*

► **Proposition 27.** *Let $f : X \rightarrow Y$ be a map in a slice \mathcal{C}/J . The following are equivalent.*

1. f has the fibered right lifting property against \top , viewed as a map $\Phi_{\text{true}} \rightarrow 1_{\Phi}$ in \mathcal{C}/Φ .
2. f has the fibered right lifting property against generating cofibrations of the form $A \rightarrow 1_B$ in slice categories \mathcal{C}/B .
3. f has the fibered right lifting property against every generating cofibration.
4. f has the right lifting property against every generating cofibration in \mathcal{C}/J .

► **Definition 28** (Trivial fibrations and cofibrations). *If a map $f : X \rightarrow Y$ in a slice category \mathcal{C}/J satisfies one, and so all, of the equivalent conditions in Proposition 27 we say that f is a trivial fibration. A map m in a slice category \mathcal{C}/I is a cofibration if it has the fibered left lifting property against every trivial fibration.*

When working in **Agda** we found it helpful to use an alternative definition of trivial fibration following [15, Sec. 5.1]. We say that a type $A : \mathcal{U}$ is *contractible* if the type $\mathbf{SContr} A$ is inhabited, where we define $\mathbf{SContr} A \triangleq (\varphi : \Phi) \rightarrow (t : [\varphi] \rightarrow A) \rightarrow A[\varphi \mapsto t]$. We define a map $f : X \rightarrow Y$ to be a trivial fibration if every fiber is contractible.

If m and \mathcal{C} satisfy the necessary conditions to apply Theorem 24 then there is an awfs (\mathcal{C}, F^t) where the class underlying F^t is precisely the class of trivial fibrations. We refer to maps in the class underlying \mathcal{C} as *cofibrations*.

3.3 Trivial cofibration and fibration awfs

We now give a more abstract characterization of weak fibrations (Definition 4) and define an awfs where the right maps are weak fibrations. Following Gambino and Sattler [24], we use the Leibniz adjunction to describe fibrations, writing $\hat{\times}_B$ and $\mathbf{hom}_B(-, -)$ for the Leibniz product and exponential constructed in a slice category \mathcal{C}/B . We also use the following notion of *weak lifting property*. This definition (although not the name) has been used before in homotopical algebra by Dold [20] and also by Reedy [30]. Note however that the definition of fibration considered by Dold is weaker than the one here, as one may see from Lemma 8.

► **Definition 29** (Weak left lifting property). *Let $m : A \rightarrow B$ and $f : X \rightarrow Y$. We say m has the weak left lifting property against f if for every commutative square, as in the solid lines below, there is a diagonal map, as in the dotted line below, such that the lower triangle commutes strictly, and the upper triangle commutes up to a homotopy $h : j \circ m \sim a$ such that $f \circ h$ is constant. We refer to such diagonal maps as weak fillers.*

$$\begin{array}{ccc} A & \xrightarrow{a} & X \\ m \downarrow & \sim \nearrow & \downarrow f \\ B & \xrightarrow{b} & Y \end{array}$$

► **Theorem 30.** *A map $f : X \rightarrow Y$ is a weak fibration if and only if for every object B , every map $r : 1_B \rightarrow \mathbb{I}_B$ and generating cofibration $m : A \rightarrow 1_B$ in \mathcal{C}/B , r has the weak left lifting property against $\hat{\text{hom}}_B(m, f)$.*

Proof. Working in \mathcal{C}/B , r has the weak left lifting property against $\hat{\text{hom}}_B(m, f)$ iff every lifting problem of $r \hat{\times}_B m$ against f has a weak filler satisfying the additional condition of being strict on A . This holds for all B , r and m and every choice of lifting problem iff it holds for the universal lifting problem of $\Delta \hat{\times}_{\mathbb{I} \times \Phi} \top$ against f , where Δ is the map $1_{\mathbb{I} \times \Phi} \rightarrow \mathbb{I}_{\mathbb{I} \times \Phi}$ in $\mathcal{C}/(\mathbb{I} \times \Phi)$ defined as the diagonal map $\mathbb{I} \times \Phi \rightarrow \mathbb{I} \times \mathbb{I} \times \Phi$. Such fillers of the universal lifting problem correspond precisely to WComp terms. ◀

In order to obtain an awfs, we show that the above is equivalent to an alternative definition using the mapping cylinder factorization, which we recall is defined as below.

► **Definition 31** (Mapping cylinder factorization). *Let $m : A \rightarrow B$. We define the mapping cylinder factorization to be the maps $A \xrightarrow{L(m)} \text{Cyl}(m) \xrightarrow{R(m)} B$, defined as follows. We first define $\text{Cyl}(m)$ as the pushout of δ_{A_0} and m , writing $\iota_0 : \mathbb{I} \times A \rightarrow \text{Cyl}(m)$ and $\iota_1 : B \rightarrow \text{Cyl}(m)$ for the pushout inclusions. We define $L(m)$ to be $\iota_0 \circ \delta_{A_1}$ and define $R(m)$ to be the unique map such that $R(m) \circ \iota_0 = m \circ \pi_1$ and $R(m) \circ \iota_1 = 1_B$.*

► **Theorem 32.** *Let f be a map in \mathcal{C} . Then f is a weak fibration if and only if it has the fibered right lifting property against the map $L_{\mathbb{I} \times \Phi}(\Delta) \hat{\times}_{\mathbb{I} \times \Phi} \top$ in the slice category $\mathcal{C}/(\mathbb{I} \times \Phi)$.*

Using this alternative definition, we can apply Theorem 24 to obtain an awfs (C^t, F) where F is precisely the class of weak fibrations. We refer to maps in C^t as *trivial cofibrations*.

3.4 The model structure

Now that we have defined the awfs's (C, F^t) and (C^t, F) , we use Sattler's [32, Thm. 2.8] in order to obtain a model structure on \mathcal{C} .

► **Lemma 33.** *The awfs's (C, F^t) and (C^t, F) have the following key properties.*

1. *The functor $\hat{\text{hom}}(\delta_i, -)$ maps fibrations to trivial fibrations.*
2. *The functor $\hat{\text{hom}}([\delta_0, \delta_1], -)$ preserves fibrations and trivial fibrations.*
3. *Every cofibration is a monomorphism.*
4. *Cofibrations are stable under pullback.*

► **Theorem 34.** *Suppose that \mathcal{C} satisfies axioms **ax**₁–**ax**₅ and that every fibration is \mathbb{U} -small for some universe of small fibrations where the underlying object \mathbb{U} is fibrant, and that \mathcal{C} and Φ satisfy one of the conditions required to apply Theorem 24.*

Let (C, F^t) be the awfs defined in Section 3.2 and let (C^t, F) be the awfs defined in Section 3.3 (restricted to $\mathcal{C}/1$). Then C and F form the cofibrations and fibrations of a (uniquely determined) model structure on \mathcal{C} .

14:14 Unifying Cubical Models of Univalent Type Theory

Proof. By Sattler’s [32, Thm. 2.8] it suffices to check the following conditions.

1. The span property holds.
2. Trivial fibrations satisfy 2-out-of-3 relative to fibrations.
3. Fibrations and trivial fibrations extend along trivial cofibrations.
4. The wfs (C^t, F) satisfies the Frobenius property.

Conditions (1) and (2) follow from the key properties (1) and (2) in Lemma 33 by essentially the same arguments used by Sattler in [32, Sec. 4].

Trivial fibrations extend along all cofibrations, by the same argument used by Sattler in [32, Lem. 3.9] together with the key properties (3) and (4) in Lemma 33.

As Sattler remarks in [32, Rem. 7.6], to show fibrations extend along trivial cofibrations it suffices to show every fibration belongs to a universe \mathbf{U} where the underlying object is fibrant, which we assumed.

Finally, (C^t, F) is Frobenius by the existence of fibration structures on \mathbb{I} -types and the adjunction between pullback and dependent product. ◀

In particular, if \mathbf{ax}_6 and \mathbf{ax}_7 hold and \mathbb{I} is tiny, we can use the construction of \mathbf{U} from Section 2.6 together with the proof of fibrancy in [13, Sec. 4.3].

The model structure obtained this way is “minimal” in the following sense [14, Sec. 1.6].

► **Theorem 35.** *The class C^t is as small as possible subject to the following two conditions.*

1. *For every object B , the map $\delta_{B0} : B \rightarrow B \times \mathbb{I}$ belongs to C^t .*
2. *C and C^t form the cofibrations and trivial cofibrations of a model structure.*

4 Identity types and higher inductive types

We have formalized three constructions of identity types in **Agda**, each of which requires additional assumptions. The first follows [15, Sec. 9.1]; this requires a dominance on Φ and extensionality for cofibrant propositions. The second approach uses the (C, F^t) factorization system following [33], while the third approach uses the (C^t, F) factorization system following [12, 11]. These rely on W -types with reductions to obtain the factorization systems. We refer the interested reader to the **Agda** formalization for details.

A crucial component for modeling universes closed under higher inductive types is the decomposition of composition into *homogeneous* composition and coercion [12, 18]. A type $A : \Gamma \rightarrow \mathcal{U}$ supports weak homogeneous composition if all of its fibers support weak composition, i.e. for all $(x : \Gamma)$ the type $A x$ has a weak composition structure. Supporting weak coercion corresponds to having weak composition only in the case when φ is \perp (i.e., the tube is empty). We have formalized that a type has weak composition if and only if it has weak homogeneous composition and weak coercion. This makes it possible for us to follow the same approach as in [12, 18] to model higher inductive types. We refer the reader to [13, Sec. 5.1] for the construction of a circle type in this setting.

5 Conclusions

We have proved that any locally cartesian closed category \mathcal{C} with \mathbb{I} and Φ satisfying \mathbf{ax}_1 – \mathbf{ax}_7 and where \mathbb{I} is tiny provides a constructive model of HoTT/UF. Examples of such categories are CCHM and Dedekind cubical sets as proven in [28, Sec. 8], and cartesian cubical sets as proven in [2, Sec. 3.2]. Our conditions hold for cubical assemblies [37] and also apply to new variants of cubical assemblies based on cartesian cubes rather than Dedekind cubes.

Our construction of a model structure also applies to all of the above examples. As observed by Sattler [32, Cor. 8.5], the LOPS construction of a universe does not apply for simplicial sets because the interval is not tiny, but one can still obtain a model structure using the non-constructive theorem that the definition of Kan fibration here is equivalent to the classical definition using horn inclusions.

From the perspective of practical implementation and usability, the type theory corresponding to this model is inferior to the type theories it generalizes: equalities that are strict in the specialized type theories here only hold up to paths, so additional path algebra is necessary to implement composition at the various types. The objective is rather to present a theory with which the mathematical properties of the various type theories and models can be studied simultaneously.

Future work

Now that we have given a unified construction for the various cubical models, the natural next step is to use it to establish relationships between its various instantiations. One option is to prove homotopy canonicity for the type theory using categorical gluing as in [19]. This would show that closed terms of natural number type written in weak cartesian type theory evaluate to the same numeral in any of the existing cubical type theories.

The construction may also be useful for uniformly analyzing the model structures induced by different choices of cube category and generating cofibrations. Sattler has observed [17] that the CCHM and ABCFHL constructions give model structures that are *not* Quillen equivalent to spaces. However, the question is open for Dedekind cubes. One might also investigate the relationships *between* the various cubical model structures.

Finally, the program of unification remains unfinished, as the BCH model is not an instance of our construction. Indeed, our approach seems ill-suited to BCH, as it crucially involves the diagonal ($r = s$) of compositions $r \rightarrow s$. It is unclear to us whether BCH can be naturally accommodated; it may simply be a fundamentally different construction.

5.1 Related work

As the notion of fibration defined in this paper coincides with the one of Orton and Pitts [28] in the presence of a connection algebra, and this is equivalent to the Gambino-Sattler definition [24], we recover the model structure of Sattler [32] when the category also has connections. Another presentation of this model structure on CCHM and Dedekind cubical sets can be found in Boulier’s Ph.D. thesis [10], formalized in the `Coq` proof assistant. Since an equivalent definition of fibration was used by Van den Berg and Frumin in [22], when our model structure exists we can recover theirs by restricting to fibrant objects. However, our proof does not apply to their main example of the effective topos because it is unknown how to construct a universe satisfying \mathbf{ax}_6 in this setting (see [35, Thm. 5.7]).

Furthermore, as we recover AFH fibrations when we assume diagonal cofibrations, we also recover the model structure on cartesian cubical sets sketched by Coquand based on Sattler’s model structure [17]. Awodey [4] uses a variation of composition $0 \rightarrow r$ and $1 \rightarrow r$ to construct an awfs on cartesian cubical sets, but it is unclear whether this is sufficient to obtain a model structure. Awodey has recently [6] introduced a notion of “unbiased fibrations” that are equivalent to AFH fibrations, so the resulting model structure is also a special case of ours when we assume diagonal cofibrations. Our generalization hence clarifies the relationship between some of the various model structures on different cubical set categories.

References

- 1 Agda development team. *Agda 2.6.0.1 documentation*, 2018. URL: <https://agda.readthedocs.io/en/v2.6.0.1/>.
- 2 Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. Syntax and Models of Cartesian Cubical Type Theory. Draft available at <https://github.com/dlicata335/cart-cube/blob/master/cart-cube.pdf>, 2017.
- 3 Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. Cartesian Cubical Computational Type Theory: Constructive Reasoning with Paths and Equalities. In *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, pages 6:1–6:17, 2018. doi:10.4230/LIPIcs.CSL.2018.6.
- 4 Steve Awodey. A cubical model of homotopy type theory. *Annals of Pure and Applied Logic*, 169(12):1270–1294, 2018. Logic Colloquium 2015. doi:10.1016/j.apal.2018.08.002.
- 5 Steve Awodey. Recent Work in Homotopy Type Theory: Modal, Algebraic, Synthetic, and Cubical, March 2018. Talk given at MURI Team Meeting at CMU, available at <https://ncatlab.org/homotopytypetheory/files/awodeyMURI18.pdf>.
- 6 Steve Awodey. A Quillen model structure on the category of cartesian cubical sets. Preprint available at <https://github.com/awodey/math/blob/master/QMS/qms.pdf>, 2019.
- 7 Marc Bezem, Thierry Coquand, and Simon Huber. A Model of Type Theory in Cubical Sets. In *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 107–128, 2014. doi:10.4230/LIPIcs.TYPES.2013.107.
- 8 Marc Bezem, Thierry Coquand, and Simon Huber. The Univalence Axiom in Cubical Sets. *J. Autom. Reasoning*, 63(2):159–171, 2019. doi:10.1007/s10817-018-9472-6.
- 9 Marc Bezem, Thierry Coquand, and Erik Parmann. Non-Constructivity in Kan Simplicial Sets. In Thorsten Altenkirch, editor, *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015)*, volume 38 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 92–106, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.TLCA.2015.92.
- 10 Simon Boulier. *Extending Type Theory with Syntactic Models*. PhD thesis, Université Bretagne Loire, 2018.
- 11 Evan Cavallo. Stable factorization from a fibred algebraic weak factorization system. Preprint arXiv:1910.03121 [math.CT], 2019.
- 12 Evan Cavallo and Robert Harper. Higher Inductive Types in Cubical Computational Type Theory. *Proc. ACM Program. Lang.*, 3(POPL):1:1–1:27, January 2019. doi:10.1145/3290314.
- 13 Evan Cavallo and Anders Mörtberg. A Unifying Cartesian Cubical Type Theory. Preprint available at <https://github.com/mortberg/gen-cart/blob/master/unifying-cartesian.pdf>, 2019.
- 14 Evan Cavallo, Anders Mörtberg, and Andrew Swan. Model Structures on Cubical Sets. Preprint available at <https://github.com/mortberg/gen-cart/blob/master/modelstructure.pdf>, 2019.
- 15 Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In *Types for Proofs and Programs (TYPES 2015)*, volume 69 of *LIPIcs*, pages 5:1–5:34, 2018. doi:10.4230/LIPIcs.TYPES.2015.5.
- 16 Thierry Coquand. Variation on cubical sets. Unpublished note available at www.cse.chalmers.se/~coquand/diag1.pdf, 2014.
- 17 Thierry Coquand. Quillen model structure. Link and discussion available at https://groups.google.com/forum/#!msg/homotopytypetheory/RQkLWZ_83kQ/tAyb3zYTBQAJ, 2018.
- 18 Thierry Coquand, Simon Huber, and Anders Mörtberg. On Higher Inductive Types in Cubical Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '18*, pages 255–264. ACM, 2018. doi:10.1145/3209108.3209197.

- 19 Thierry Coquand, Simon Huber, and Christian Sattler. Homotopy Canonicity for Cubical Type Theory. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany.*, volume 131 of *LIPICs*, pages 11:1–11:23. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPICs.FSCD.2019.11.
- 20 Albrecht Dold and Rene Thom. Quasifaserungen und Unendliche Symmetrische Produkte. *Annals of Mathematics*, 67(2):239–281, 1958. doi:10.2307/1970005.
- 21 Peter Dybjer. Internal Type Theory. In *Lecture Notes in Computer Science*, pages 120–134. Springer Verlag, Berlin, Heidelberg, New York, 1996. doi:10.1007/3-540-61780-9_66.
- 22 Dan Frumin and Benno Van den Berg. A homotopy-theoretic model of function extensionality in the effective topos. *Mathematical Structures in Computer Science*, 29(4):588–614, 2019. doi:10.1017/S0960129518000142.
- 23 Nicola Gambino and Simon Henry. Towards a constructive simplicial model of Univalent Foundations. Preprint arXiv:1905.06281 [math.CT], 2019.
- 24 Nicola Gambino and Christian Sattler. The Frobenius condition, right properness, and uniform fibrations. *Journal of Pure and Applied Algebra*, 221(12):3027–3068, 2017. doi:10.1016/j.jpaa.2017.02.013.
- 25 Martin Hofmann. Syntax and semantics of dependent types. In A.M. Pitts and P. Dybjer, editors, *Semantics and logics of computation*, volume 14 of *Publ. Newton Inst.*, pages 79–130. Cambridge University Press, Cambridge, 1997. Papers from the Summer School held at the University of Cambridge, Cambridge, September 1995. doi:10.1017/CB09780511526619.004.
- 26 Chris Kapulkin and Peter LeFanu Lumsdaine. The Simplicial Model of Univalent Foundations (after Voevodsky). Preprint arXiv:1211.2851v4 [math.LO], November 2012.
- 27 Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. Internal Universes in Models of Homotopy Type Theory. In *FSCD*, volume 108 of *LIPICs*, pages 22:1–22:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.FSCD.2018.22.
- 28 Ian Orton and Andrew M. Pitts. Axioms for Modelling Cubical Type Theory in a Topos. *Logical Methods in Computer Science*, Volume 14, Issue 4, December 2018. doi:10.23638/LMCS-14(4:23)2018.
- 29 Jason Parker. Duality between Cubes and Bipointed Sets. Master’s thesis, Carnegie Mellon University, 2014.
- 30 C. L. Reedy. Homotopy Theory of Model Categories. Unpublished note available at <http://www-math.mit.edu/~psh/reedy.pdf>, 1974.
- 31 Emily Riehl. *Categorical Homotopy Theory*. New Mathematical Monographs. Cambridge University Press, 2014. doi:10.1017/CB09781107261457.
- 32 Christian Sattler. The Equivalence Extension Property and Model Structures. Preprint arXiv:1704.06911v1 [math.CT], 2017.
- 33 Andrew Swan. Identity Types in Algebraic Model Structures and Cubical Sets. Preprint arXiv:1808.00915 [math.CT], August 2018.
- 34 Andrew Swan. Lifting Problems in Grothendieck Fibrations. Preprint arXiv:1802.06718 [math.CT], February 2018.
- 35 Andrew Swan. Separating Path and Identity Types in Presheaf Models of Univalent Type Theory. Preprint arXiv:1808.00920 [math.LO], 2018.
- 36 Andrew Swan. W -types with reductions and the small object argument. Preprint arXiv:1802.07588 [math.CT], February 2018.
- 37 Taichi Uemura. Cubical Assemblies and Independence of the Propositional Resizing Axiom. Preprint arXiv:1803.06649 [cs.LO], 2018.
- 38 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. <http://homotopytypetheory.org/book>, Institute for Advanced Study, 2013.
- 39 Vladimir Voevodsky. An experimental library of formalized Mathematics based on the univalent foundations. *Mathematical Structures in Computer Science*, 25:1278–1294, 2015. doi:10.1017/S0960129514000577.

FO-Definability of Shrub-Depth

Yijia Chen 

Fudan University, Shanghai, China
yijiachen@fudan.edu.cn

Jörg Flum

Albert-Ludwigs-Universität Freiburg, Germany
flum@uni-freiburg.de

Abstract

Shrub-depth is a graph invariant often considered as an extension of tree-depth to dense graphs. We show that the model-checking problem of monadic second-order logic on a class of graphs of bounded shrub-depth can be decided by AC^0 -circuits after a precomputation on the formula. This generalizes a similar result on graphs of bounded tree-depth [3]. At the core of our proof is the definability in first-order logic of tree-models for graphs of bounded shrub-depth.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis; Mathematics of computing → Graph theory

Keywords and phrases shrub-depth, model-checking, monadic second-order logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.15

Funding The collaboration of the authors is funded by the Sino-German Center for Research Promotion (GZ 1518). Yijia Chen is also supported by National Natural Science Foundation of China (Project 61872092).

Acknowledgements We thank Abhisekh Sankaran for discussions concerning problems of the last section. Anonymous reviewers' comments also help to improve the presentation.

1 Introduction

In [15] Ganian et al. introduced the graph invariant shrub-depth with the goal to extend the invariant tree-depth in a similar way as clique-width extends tree-width. Shrub-depth turned out to be a quite robust notion as shown by the following result of [15].

For a class K of graphs the following are equivalent:

- (i) K has bounded shrub-depth.
- (ii) K has an MSO-interpretation (i.e., an interpretation definable in monadic second-order logic MSO) of width one in a class of rooted labelled trees of bounded depth.
- (iii) K has bounded SC-depth (subset-complementation depth).

Let $p\text{-MC}(K, \text{MSO})$ denote the parameterized model-checking problem for MSO on the class K parameterized by the length of the formula. In [3] we showed that $p\text{-MC}(K, \text{MSO})$ is in para-AC^0 for every class K of graphs of bounded tree-depth. The parameterized circuit complexity class para-AC^0 is considered to be the parameterized analog of the circuit complexity class (dlogtime-uniform) AC^0 . In fact, by definition, a parameterized problem is in para-AC^0 if it is in (dlogtime-uniform) AC^0 after a precomputation on the parameter. Recall that the class FPT (fixed-parameter-tractability) consists of the parameterized problems that are solvable in polynomial time after a precomputation on the parameter.

As the main result of this paper we extend our result on the MSO-model-checking for classes of bounded tree-depth to classes of bounded shrub-depth.

► **Theorem 1.** $p\text{-MC}(K, \text{MSO}) \in \text{para-AC}^0$ for every class of bounded shrub-depth.



© Yijia Chen and Jörg Flum;
licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 15; pp. 15:1–15:16

Leibniz International Proceedings in Informatics



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

15:2 FO-Definability of Shrub-Depth

It is well known that $p\text{-MC}(K, \text{MSO}) \in \text{FPT}$ if K has bounded clique-width (a result due to Courcelle et al. [5]). By [15] every class of bounded shrub-depth has bounded clique-width. Hence, $p\text{-MC}(K, \text{MSO}) \in \text{FPT}$ for K of bounded shrub-depth. However, there exist graph classes K of bounded clique-width with $p\text{-MC}(K, \text{MSO}) \notin \text{para-AC}^0$, e.g., the class of all graphs consisting of disjoint paths [3, Theorem 7.3]. Therefore, the algorithmic techniques via clique-width cannot be adapted to para-AC^0 . Instead, we develop some combinatorial machinery on graphs of bounded shrub-depth which can be defined in first-order logic (FO).

We briefly explain some ingredients of this combinatorial machinery. Central to the definition of shrub-depth are tree-models of graphs. The tree-models are rooted trees of constant depth with colored leaves, the leaves being the vertices of the corresponding graph. Their FO-definability has presented some major challenges. To better understand tree-models, we find it more convenient to work with the SC-depth instead of the shrub-depth. Roughly speaking, the SC-depth $\text{SC}(G)$ of a graph G is the minimum number of parallel subset complementations required to construct G from graphs without any edges (i.e, from graphs of isolated vertices). The equivalence between (i) and (iii) mentioned at the beginning tells us that the boundedness of the SC-depth of a class of graphs is equivalent to the boundedness of its shrub-depth. As a first step we prove that the complementation subsets (we call them flipping sets) underlying $\text{SC}(G)$ can be uniquely determined in FO if we are given an *unambiguous representative system* of G (once having the flipping sets we can construct a tree-model as in the proof of the implication (iii) \Rightarrow (i) in [15]). However the size of a representative system cannot be bounded in terms of the depth of the graph. Hence we cannot afford to guess such a system in FO. We show that every graph of bounded SC-depth has a representation as a *tiered graph*. For such graphs we can guess appropriate representative systems iteratively in FO. Once all the flipping subsets have been obtained, we can FO-define a tree-model. More precisely, we show:

► **Theorem 2.** *If K is a class of bounded shrub-depth (or of bounded SC-depth), there is an FO-interpretation that assigns to every ordered graph $(G, <)$ with $G \in K$ a tree-model.*

Barrington et al. [1] showed that the expressive power of FO with built-in arithmetic coincides exactly with $\text{dlogtime-uniform AC}^0$ -computability. Using this fact we get Theorem 1 from Theorem 2 in the same way as we did for tree-depth in [3].

We obtain a further consequence of our proof of the FO-definability of tree-models. In fact, we get that every MSO-sentence is equivalent to an FO-sentence on *ordered* graphs of bounded shrub-depth. More precisely:

► **Proposition 3.** *Let K be a class of graphs of bounded shrub-depth. Then for every MSO-sentence φ there is an FO-sentence ψ such that for any ordered graph $(G, <)$ with $G \in K$,*

$$G \models \varphi \iff (G, <) \models \psi.$$

Observe that the above φ has no access to the order, while ψ does. So it is natural to ask whether we can eliminate all occurrences of $<$ in ψ , in other words, whether $\text{MSO} = \text{FO}$ on K . The result was already claimed by Gajarský and Hliněný [11]:

► **Theorem 4.** *$\text{MSO} = \text{FO}$ on every class of graphs of bounded shrub-depth.*

We prove this result from Proposition 3 using Craig's Interpolation Theorem. Craig's Interpolation Theorem [6] is a basic result in classical model theory. However it fails on finite models [17]. To circumvent this problem, in a straightforward way we generalize the notion of

SC-depth to infinite graphs and observe that our combinatorial characterization of bounded SC-depth carries over to infinite graphs as well. The excursion to the infinite yields new insights for finite graphs, e.g., we show the following effective version of [15, Corollary 5.6]:

► **Theorem 5.** *There is an algorithm that applied to d eventually stops and outputs a finite set F_d of graphs such that a graph has SC-depth $\leq d$ if and only if it excludes the graphs in F_d as induced subgraphs.*

Related work. Theorem 1 can be viewed as a part of the recent efforts to extend algorithmic meta-theorem to dense graphs. Algorithmic meta-theorems unify many algorithmic results on graph classes where the underlying computational problems can be defined in terms of logic. Most existing such meta-theorems concern sparse graph classes, i.e., graph classes where the number of edges is linearly bounded by the number of vertices. As examples we mention Courcelle’s Theorem [4] that the p -MC(MSO, K) can be solved in fixed-parameter linear time provided that K has bounded tree-width and the result due to Grohe et al. [16] stating that p -MC(FO, K) \in FPT if K is a nowhere dense class. The dependence of the parameter in Courcelle’s Theorem is non-elementary as shown by Frick and Grohe [9]. Improvements of the dependence of the parameter are known for various classes of graphs (see e.g., Gajarský and Hliněný [10] and Lampis [19]). A similar better dependence of the parameter holds for graph classes of bounded shrub-depth if every graph is given alongside with a tree-model of corresponding depth [10]. As far as circuit complexity is concerned, Pilipczuk et al. showed [20] that the model-checking problem for FO on graphs of bounded expansion can be decided by circuits of size $f(k) \cdot n^{O(1)}$ and of depth $f(k) + O(\log n)$, where k is the size of the input formula and n the size of the graph.

Compared to sparse graph classes, much less is known for dense graphs. We have already mentioned that p -MC(K , MSO) \in FPT if K has bounded clique-width. Recall that the class of cliques (i.e., complete graphs), which are obviously dense, has clique-width 1. For first-order logic, algorithmic meta-theorems are known e.g., for interval graphs [14], partial orders [12], and graphs FO-interpretable in bounded degree graphs [13].

In [8] Elberfeld et al. proved that MSO = FO on graphs of bounded tree-depth. Graphs of bounded tree-depth has bounded shrub-depth as well. Thus Theorem 4 generalizes this result. As already mentioned, Theorem 4 was first claimed in [11, Theorem 5.14]. One crucial tool is based on the proof of [11, Theorem 5.2]. However, we could not verify this proof. Besides that, our proof uses completely different techniques.

The MSO-sentence ψ in Proposition 3 contains a symbol for the order relation, however its validity in a graph G (of the class K) does not depend on what order of the set of vertices of G we choose. That is, by definition, the sentence φ is order-invariant on K . In a recent paper Eickmeyer et al. [7] obtain FPT-tractability results for the set of order-invariant MSO-sentences essentially for the same classes of graphs as in the unordered case. However the model-checking problem for order-invariant MSO on graphs of bounded tree-depth (thus on graphs of bounded shrub-depth) is not in para-AC⁰. In fact, consider graphs consisting of disjoint triangles and isolated vertices. Then the parity of the number of the triangles can be expressed by an order-invariant MSO-sentence. On the other hand, it is easy to see that this property cannot be in AC⁰ by PARITY \notin AC⁰.

Organization of this paper. In Section 2 we fix some notations. In Section 3 and Section 4 we recall the definitions and some basic properties of shrub-depth and of SC-depth, respectively, and show that the classes $\text{TM}_m(d)$ and $\text{SC}(d)$ are MSO-axiomatizable. In Sections 5–7

we stepwise develop the machinery which finally allows us to prove Theorem 2 (and hence, Theorem 1) in Section 8. Theorem 4 and Theorem 5 are shown in Section 9 and Section 10, respectively.

Due to space limitations we defer many proofs to the full version of this paper. Sometimes we indicate this by writing “Proof: full paper” at the end of the statement of a theorem, proposition, . . .

2 Preliminaries

We denote by \mathbb{N} the set of natural numbers ≥ 0 . For $n \in \mathbb{N}$ let $[n] := \{1, 2, \dots, n\}$.

First-order logic FO and monadic second-order logic MSO. A *vocabulary* τ is a finite set of relation symbols. Each relation symbol has an *arity*. A *structure* \mathcal{A} of vocabulary τ , or τ -*structure*, consists of a nonempty set A , called the *universe* of \mathcal{A} and of an interpretation $R^{\mathcal{A}} \subseteq A^r$ of each r -ary relation symbol $R \in \tau$. In this paper all structures have a finite universe with the exception of Section 9 and Section 10.

Formulas φ of *first-order logic* FO of vocabulary τ are built up from *atomic formulas* $x_1 = x_2$ and $Rx_1 \dots x_r$ (where $R \in \tau$ is of arity r and x_1, x_2, \dots, x_r are variables) using the boolean connectives \neg , \wedge , and \vee and the universal \forall and existential \exists quantifiers. By the notation $\varphi(\bar{x})$ with $\bar{x} = x_1, \dots, x_e$ we indicate that the variables free in φ are among x_1, \dots, x_e . In addition to the individual variables of FO, *formulas* of *monadic second-order logic* MSO may also contain *set variables*. We use lowercase letters (usually x, y, z) to denote individual variables and uppercase letters (usually X, Y, Z) to denote set variables. To obtain MSO the syntax of FO is enhanced by new atomic formulas of the form Xy and quantification is also allowed over set variables.

Graphs and trees. In this paper *graphs* are always simple and undirected. When considering definability problems for graphs we view graphs as $\tau := \{E\}$ -structures where the edge relation is an irreflexive and symmetric binary relation. Otherwise we use the notation G for a graph and view it as a pair $G = (V(G), E(G))$, where $V(G)$ is the set of vertices and $E(G)$ the set of edges. For graphs G and H with disjoint vertex sets we denote by $G \dot{\cup} H$ the graph with vertex set $V(G) \cup V(H)$ and edge set $E(G) \cup E(H)$.

We view *rooted trees with m labels* as $\tau_m := \{P, L_1, \dots, L_m\}$ -structures $\mathcal{T} = (T, P^{\mathcal{T}}, L_1^{\mathcal{T}}, \dots, L_m^{\mathcal{T}})$. Here P is a binary relation symbol and L_1, \dots, L_m are unary. $P^{\mathcal{T}}$ is the *parent-child relation* of the tree. The root of the tree can be defined by the formula $root(x) := \forall y \neg Pyx$. The relations $L_1^{\mathcal{T}}, \dots, L_m^{\mathcal{T}}$ are the *labels*. Recall that the *depth* of \mathcal{T} is the maximum length of a path from the root to a leaf. We denote by $leaves(\mathcal{T})$ the set of leaves of \mathcal{T} . For $m, d \in \mathbb{N}$ we denote by $TREE[m, d]$ the class of rooted trees with m labels and of depth d , where each root-to-leaf path is of length exactly d .

3 Shrub-depth

We recall the notion of the shrub-depth of a graph (introduced in [15]) and show that the classes $TM_m(d)$ (with $m, d \in \mathbb{N}$) of bounded shrub-depth are axiomatizable in MSO.

► **Definition 6.** Let $m, d \in \mathbb{N}$. A tree-model of m labels and depth d of a graph G is a pair (\mathcal{T}, D) with $\mathcal{T} \in TREE[m, d]$ and $D \subseteq \{1, 2, \dots, m\}^2 \times \{1, 2, \dots, h\}$ for some $h \geq d$ ¹ (called

¹ For technical reasons (in particular, for the proof of Proposition 12), we allow h to be greater than d .

the signature of the tree-model) such that

- $V(G) = \text{leaves}(\mathcal{T})$,
- each leaf of T holds exactly one label from $\{P_1, \dots, P_m\}$ and no other node of T holds a label, i.e., $\text{leaves}(\mathcal{T}) = P_1^{\mathcal{T}} \dot{\cup} \dots \dot{\cup} P_m^{\mathcal{T}}$,
- for any $i, j \in [m]$ and $s \in [d]$ if $(i, j, s) \in D$, then $(j, i, s) \in D$,
- $E(G) = \{\{u, v\} \mid u, v \in V(G), u \neq v, u \in P_i^{\mathcal{T}}, v \in P_j^{\mathcal{T}}, \text{ and } (i, j, \text{dist}^{\mathcal{T}}(u \wedge v, u)) \in D\}$.

By $\text{dist}^{\mathcal{T}}(u \wedge v, u)$ we denote the distance from the least common ancestor $u \wedge v$ of u and v to u . Note that $\text{dist}^{\mathcal{T}}(u \wedge v, u) = \text{dist}^{\mathcal{T}}(u \wedge v, v)$, as both u and v are leaves of \mathcal{T} (of the same depth). In the context of tree-models we also speak of the colors P_i and say that vertex v has color P_i if $v \in P_i^{\mathcal{T}}$.

► **Definition 7** ([15]). Let $\text{TM}_m(d)$ denote the class of graphs with a tree-model of m labels and depth d . A class K of graphs has shrub-depth d if there exists m such that $K \subseteq \text{TM}_m(d)$, while for all $m' \in \mathbb{N}$ we have $K \not\subseteq \text{TM}_{m'}(d-1)$.

The class K has bounded shrub-depth if $K \subseteq \text{TM}_m(d)$ for some $m, d \in \mathbb{N}$.

The following lemma shows that the shrub-depth is relevant only to infinite classes of graphs and not to a single graph.

► **Lemma 8.** For every graph G we have $G \in \text{TM}_{|V(G)|}(1)$.

Proof. Assume $V(G) = [m]$. Then a tree $\mathcal{T} \in \text{TREE}[m, 1]$ with $P_i^{\mathcal{T}} = \{i\}$ for $i \in [m]$ together with the signature $D := \{(i, j, 1) \mid i, j \in [m] \text{ and } \{i, j\} \in E(G)\}$ is a tree-model for G . ◀

The shrub-depth hierarchy is strict:

► **Proposition 9.** Let $d \in \mathbb{N}$. The class of graphs underlying rooted trees in $\text{TREE}[m, d]$ for some (= all) m has shrub depth d . Proof: full paper.

The following facts are easy to verify.

► **Lemma 10.**

- (a) $\text{TM}_m(d) \subseteq \text{TM}_{m'}(d')$ for $m \leq m'$ and $d \leq d'$.
- (b) $\text{TM}_m(d)$ is closed under induced subgraphs.
- (c) Every graph which is a clique is in $\text{TM}_1(1)$.

A proof of the next result can be found in [15].

► **Proposition 11.** There is a computable function $\ell : \mathbb{N}^2 \rightarrow \mathbb{N}$ such that for $G \in \text{TM}_m(d)$ every two vertices, which are in the same connected component of G , have a distance $\leq \ell(m, d)$. Hence, for fixed $m, d \in \mathbb{N}$, we can express in FO that two vertices are in the same connected component in graphs of $\text{TM}_m(d)$.

By this lemma we see that the class of paths is not of bounded shrub-depth. As every path is a subgraph of some clique, Lemma 10(c) shows that the classes $\text{TM}_m(d)$ for $m, d \geq 1$ are not closed under subgraphs.

By [15, Corollary 5.6] we know that for $m, d \in \mathbb{N}$ there is a finite set $F_{m,d}$ of graphs such that a graph G is in $\text{TM}_m(d)$ if and only if “ G excludes the graphs in $F_{m,d}$ as induced subgraphs”, i.e, no graph in $F_{m,d}$ is isomorphic to an induced subgraph of G . Hence there is an FO-sentence $\rho(m, d)$ axiomatizing $\text{TM}_m(d)$. However the proof of [15, Corollary 5.6] sheds no light on how to compute $\rho(m, d)$ from (m, d) . We need the corresponding result for MSO in order to get such an effective FO-axiomatization of $\text{TM}_m(d)$ in Section 9.

► **Proposition 12.** We can effectively compute for $m, d \in \mathbb{N}$ an MSO-axiomatization of $\text{TM}_m(d)$. Proof: full paper.

4 SC-depth

Classes of graphs of bounded shrub-depth coincide with classes of graphs of bounded SC-depth. For our goal it is more convenient to work with the SC-depth. Let G be a graph and S a subset of its vertex set $V(G)$. Then G^S denotes the graph *obtained from G by flipping the set S* ; that is, G^S has the vertex set $V(G)$ and edge set

$$\{\{u, v\} \in E(G) \mid u \notin S \text{ or } v \notin S\} \cup \{\{u, v\} \mid u, v \in S, u \neq v, \text{ and } \{u, v\} \notin E(G)\}.$$

Here, we deviate from the original notation \bar{G}^S , which might become cumbersome when there are several flipping sets. For subsets S_1, \dots, S_n of $V(G)$ we write $G^{S_1 \dots S_n}$ for $(\dots((G^{S_1})^{S_2}) \dots)^{S_n}$. The following lemma contains some simple facts about $G \mapsto G^S$.

► **Lemma 13.** *Let $S \subseteq V(G)$, $T \subseteq V(G)$, and H be a further graph.*

- (a) *If $|S| \leq 1$, then $G^S = G$;*
- (b) *if $G^S = G^T$ and $|S| \geq 2$, then $S = T$;*
- (c) *$G^{ST} = G^{TS}$;*
- (d) *$G^{SS} = G$;*
- (e) *$(G^S \dot{\cup} H) = (G \dot{\cup} H)^S$ (recall that $S \subseteq V(G)$).*

We introduce the class $\text{SC}(d)$ of graphs of *complementation depth $\leq d$* (or, *SC-depth $\leq d$*).

► **Definition 14.** *Let $d \in \mathbb{N}$. We define inductively the class $\text{SC}(d)$.*

$\text{SC}(0)$ *is the class of graphs whose vertex set is a singleton.*

Assume that $m \geq 1$ and the graphs $G_1, \dots, G_m \in \text{SC}(d)$ have pairwise disjoint vertex sets. Then for $S \subseteq V(G_1) \cup \dots \cup V(G_m)$ we have

$$(G_1 \dot{\cup} G_2 \dot{\cup} \dots \dot{\cup} G_m)^S \in \text{SC}(d+1).$$

A class of graphs is of bounded SC-depth if it is contained in $\text{SC}(d)$ for some $d \in \mathbb{N}$.

As every clique has SC-depth ≤ 1 we see that the class of cliques has bounded SC-depth. The following lemma shows that every graph is in some $\text{SC}(d)$. We define the *SC-depth* $\text{SC}(G)$ of a graph G as the least $d \in \mathbb{N}$ such that $G \in \text{SC}(d)$.

► **Lemma 15.** *$G \in \text{SC}(|V(G)| - 1)$ for every graph G .*

Proof. The proof is a simple induction on $|V(G)|$. A graph with only one vertex is in $\text{SC}(0)$ by definition. Let $d \geq 1$ and let u be any vertex of a graph G with exactly $d+1$ vertices. Let H be the graph induced by G on $V(G) \setminus \{u\}$ and set $H_1 := H^{\{v \in V(H) \mid \{u, v\} \in E(G)\}}$. By induction hypothesis, $H_1 \in \text{SC}(d-1)$ as H_1 has d elements. Let U denote the graph with $V(U) = \{u\}$. As $G = (H_1 \dot{\cup} U)^{\{u\} \cup \{v \in V(H) \mid \{u, v\} \in E(G)\}}$, we get $G \in \text{SC}(d)$. ◀

If we write $G \dot{\cup} H$ we tacitly assume that the graphs G and H have disjoint vertex sets and if we write G^S we assume that $S \subseteq V(G)$.

The following basic properties of the classes $\text{SC}(d)$ will be proven in the full paper.

► **Lemma 16.** *Let $d \in \mathbb{N}$.*

- (a) $\text{SC}(d) \subseteq \text{SC}(d+1)$.
- (b) $\text{SC}(d)$ *is closed under taking induced subgraphs.*

$$(c) \text{SC}(d+1) = \left\{ (G_1 \dot{\cup} G_2 \dot{\cup} \dots \dot{\cup} G_m)^S \mid \begin{array}{l} m \geq 1, G_1, \dots, G_m \in \text{SC}(d) \text{ are con-} \\ \text{nected and } S \subseteq V(G_1) \cup \dots \cup V(G_m) \end{array} \right\}.$$

Moreover, assume that $H = (G_1 \dot{\cup} G_2 \dot{\cup} \dots \dot{\cup} G_m)^S$ with $G_i \in \text{SC}(d)$. Then for the connected components H_{ij} of G_i we have $H_{ij} \in \text{SC}(d)$ and $H = (\dot{\cup} H_{ij})^S$.

Parts (a) and (b) of the following proposition show that a class of graph has bounded shrub-depth if and only if it has bounded SC-depth. Then its part (c) follows by Proposition 11.

► **Proposition 17** ([15]).

(a) Let $m, d \in \mathbb{N}$. Then $\text{TM}_m(d) \subseteq \text{SC}(d \cdot m \cdot (m + 1))$.

(b) Let $d \in \mathbb{N}$. Then $\text{SC}(d) \subseteq \text{TM}_{2^d}(d)$.

(c) There is a computable function $\ell_{\text{SC}} : \mathbb{N} \rightarrow \mathbb{N}$ such that for $G \in \text{SC}(d)$ every two vertices, which are in the same connected component of G , have a distance $\leq \ell_{\text{SC}}(d)$.

Again using the existence of a characterization of $\text{SC}(d)$ in terms of excluding a finite set of induced subgraphs, one gets the FO-axiomatizability of $\text{SC}(d)$. We will show the effective FO-axiomatizability (see Corollary 43(a)). Here we get (see the full paper for a proof):

► **Proposition 18.** We can effectively compute for $d \in \mathbb{N}$ an MSO-axiomatization ρ_d of $\text{SC}(d)$.

5 Towers and representative systems

For $d \geq 1$ we denote by $\text{TOW}(d)$ the class of towers $\leq d$, i.e., the class of graphs which can be written in the form

$$G = (I(V(G)))^{S_1 \dots S_d}. \quad (1)$$

Here $I(X)$ denotes the graph with vertex X and no edges. By Lemma 13(a) we have $\text{TOW}(d) \subseteq \text{TOW}(d + 1)$ and by Lemma 13(e), every graph is in $\text{TOW}(d)$ for some d . Note that $\text{TOW}(d) \subseteq \text{SC}(d)$ for $d \geq 1$. However, already $\text{SC}(2)$ is not contained in any class $\text{TOW}(d)$. In fact, the graphs $G_n := \left(\{a_1, \dots, a_n, b_1, \dots, b_n\}, \{ \{a_i, b_i\} \mid i \in \mathbb{N} \} \right)$ for $n \geq 1$ are all contained in $\text{SC}(2)$. Note that every two vertices of the graph in (1), which are in the same atom of the boolean algebra generated by S_1, \dots, S_d , “behave in the same way.” Hence, $G_n \notin \text{TOW}(d)$ for $d < \log_2 n$. Readers familiar with [19] will realize that a class of graphs has bounded *neighborhood diversity* if and only if it is contained in $\text{TOW}(k)$ for some $k \in \mathbb{N}$.

In this section, as a first step towards the main results we show that the classes $\text{TOW}(d)$ are FO-axiomatizable, thereby getting familiar with some tools relevant to the general case.

We set $\mathbf{S} := S_1 \dots S_d$. For G as in (1) we associate with every $v \in G$ a “color”

$$\chi_{\mathbf{S}}(v) := (b_1, \dots, b_d) \in \{0, 1\}^d, \quad \text{where } b_i = \begin{cases} 1 & \text{if } v \in S_i \\ 0 & \text{otherwise.} \end{cases}$$

For $b = (b_1, \dots, b_d) \in \{0, 1\}^d$ and $b' = (b'_1, \dots, b'_d) \in \{0, 1\}^d$ we define

$$\langle b, b' \rangle := \sum_{i \in [d]} b_i \cdot b'_i \pmod{2}.$$

► **Lemma 19.** Let $G = (I(X))^{\mathbf{S}}$ and $v, w \in V(G)$ with $v \neq w$. Then

$$\{v, w\} \in E^G \iff \langle \chi_{\mathbf{S}}(v), \chi_{\mathbf{S}}(w) \rangle = 1.$$

Note that the mapping $\chi_{\mathbf{S}} : V(G) \rightarrow \{0, 1\}^d$ is not necessarily surjective. Assume that $\chi_{\mathbf{S}}(V(G)) = \{b_1, \dots, b_m\}$ with pairwise distinct b_i 's in $\{0, 1\}^d$. For $i \in [m]$ choose a vertex $u_i \in V(G)$ such that $\chi_{\mathbf{S}}(u_i) = b_i$. Then, $(u_1, \dots, u_m; b_1, \dots, b_m)$ is a d -representative system for G in the sense of the following definition and $\chi_{\mathbf{S}}$ is a corresponding coloring.

► **Definition 20.** Let G be a graph and $d \geq 1$. A d -representative system for G is a tuple

$$\mathcal{R} := (u_1, \dots, u_m; b_1, \dots, b_m)$$

with $u_1, \dots, u_m \in V(G)$ and with pairwise distinct elements b_1, \dots, b_m of $\{0, 1\}^d$ if there is a “coloring” $\chi: V(G) \rightarrow \{b_1, \dots, b_m\}$ with (R1) and (R2).

(R1) For every $i \in [m]$: $\chi(u_i) = b_i$.

(R2) For all $v, w \in V(G)$ with $v \neq w$: $(\{v, w\} \in E^G \iff \langle \chi(v), \chi(w) \rangle = 1)$.

The vertices u_1, \dots, u_m are then called representatives.

► **Proposition 21.** A graph is in $\text{TOW}(d)$ if and only if it has a d -representative system.

We prove this characterization of $\text{TOW}(d)$ in the full paper. It does not yield an FO-axiomatization of $\text{TOW}(d)$ as we need the coloring χ . In general this coloring is not uniquely determined (again see the full paper). This fact motivates the following definition.

► **Definition 22.**

(i) Let $d \geq 1$ and $B \subseteq \{0, 1\}^d$. The set B is unambiguous if for $b_1, b_2 \in B$, $\langle b_1, b \rangle = \langle b_2, b \rangle$ for all $b \in B$ implies $b_1 = b_2$.

(ii) Let G be a graph with $G = (I(V(G)))^{S_1 \dots S_d}$. Then S_1, \dots, S_d is unambiguous if $\chi_{S_1 \dots S_d}(V(G))$ is unambiguous. A d -representative system $(u_1, \dots, u_m; b_1, \dots, b_m)$ for G is unambiguous if $\{b_1, \dots, b_m\}$ is unambiguous.

Every representative system contains an unambiguous representative system.

► **Lemma 23.** Let $\mathcal{R} := (u_1, \dots, u_m; b_1, \dots, b_m)$ be a d -representative system for a graph G . Then there is an $s \in [m]$ and $1 \leq i_1 < \dots < i_s \leq m$ such that $(u_{i_1}, \dots, u_{i_s}; b_{i_1}, \dots, b_{i_s})$ is an unambiguous d -representative system for G . Proof: full paper.

Why is unambiguity an important property? The next result shows that for unambiguous representative systems there is a unique coloring. Its value for a vertex is already determined by its neighbors in the set of representatives.

► **Proposition 24.** Let G be a graph, $\mathcal{R} := (u_1, \dots, u_m; b_1, \dots, b_m)$ be an unambiguous d -representative system for G , and χ a corresponding coloring. Then (by Definition 20 and unambiguity) for $v \in V(G) \setminus \{u_1, \dots, u_m\}$ the color $\chi(v)$ is the unique b_j with $j \in [m]$ such that for all $i \in [m]$ we have

$$\{v, u_i\} \in E(H) \iff \langle b_j, b_i \rangle = 1.$$

Then $S_1, \dots, S_d \subseteq V(G)$ with $S_i := \{v \in V(G) \mid (\chi(v))_i = 1\}$ (by $(\chi(v))_i$ we denote the i th component of $\chi(v)$) is unambiguous and $G = (I(V(G)))^{S_1 \dots S_d}$.

Proof. The second part follows from the fact that $\chi_{S_1 \dots S_d} = \chi$. ◀

Now we easily get the FO-axiomatizability of $\text{TOW}(d)$ (for a proof see the full paper).

► **Theorem 25.** For $d \geq 1$ the class $\text{TOW}(d)$ is axiomatizable in FO.

6 Tiered graphs

We introduce (d, q) -tiered graphs, a technical tool we use to obtain our main results. Every graph we considered in the previous section is $(0, q)$ -tiered for some $q \in \mathbb{N}$. So in the preceding section we saw that for $(0, q)$ -tiered graphs G we can FO-define flipping sets that applied to $V(G)$ yield G . (d, q) -tiered graphs G contain some distinguished sets of flipping sets $\mathbf{S}_0, \dots, \mathbf{S}_d$. In this section we show that essentially we can FO-define flipping sets $\mathbf{S}'_0, \dots, \mathbf{S}'_d$ with $G^{\mathbf{S}_0, \dots, \mathbf{S}_d} = G^{\mathbf{S}'_0, \dots, \mathbf{S}'_d}$ (see Corollary 32).

► **Definition 26.** For a set X and $S_1, \dots, S_d \subseteq X$ we set

$$\text{color}(X, S_1 \dots S_d) := \{\chi_{S_1 \dots S_d}(v) \mid v \in X\},$$

the set of colors of elements of X w.r.t. S_1, \dots, S_d .

► **Definition 27.** Let $q, d \in \mathbb{N}$.

(a) If $G = (I(V(G)))^{\mathbf{S}}$ with $|\mathbf{S}| \leq q$, then G is a $(0, q)$ -tiered graph.

(b) Assume $d \geq 1$ and let $G_0, \dots, G_d = G$ be graphs. If

(i) $G_0 = (I(X))^{\mathbf{S}_0}$ for some set X and some \mathbf{S}_0 with $|\mathbf{S}_0| \leq q$,

(ii) for every $t \in [d]$ we have $G_t = \left(G_{t-1} \dot{\cup} \bigcup_{\delta \in \Delta_t, e \in F_\delta} H_{te}\right)^{\mathbf{S}_t}$ for some \mathbf{S}_t with $|\mathbf{S}_t| \leq q$, for some finite Δ_t and F_δ for $\delta \in \Delta_t$, and for some graphs H_{te} for $e \in F_\delta$ with $\delta \in \Delta_t$,

(iii) for every $t \in [d]$ and every $\delta \in \Delta_t$ we have $|F_\delta| \geq 3$ and for all $e, e' \in F_\delta$,

$$c(\delta) := \text{color}(V(H_{te}), \mathbf{S}_t \dots \mathbf{S}_d) = \text{color}(V(H_{te'}), \mathbf{S}_t \dots \mathbf{S}_d),$$

then G is a (d, q) -tiered graph.

Note that part (b)(iii) is the only restriction on the graphs H_{te} even though in our applications these graphs will be “simpler” than G . In part (b)(ii) we allow that on the right hand side of the equality at most one of the terms is missing. That is, it can be that either the term G_{t-1} is not present (G_{t-1} is the “empty” graph) or that $\Delta_t = \emptyset$ ($\left(\bigcup_{\delta \in \Delta_t, e \in F_\delta} H_{te}\right)^{\mathbf{S}_t}$ is the “empty” graph). If for $t = 1$ the term G_0 is not present, then X is empty in (b)(i).

In this section and the next one terms may represent the “empty” graph by similar reasons.

For G as in Definition 27(b)(ii) and $t \in \{0, 1, \dots, d\}$ we define the t th tier T_t by

$$T_0 = X \quad \text{and} \quad T_t = \bigcup_{\delta \in \Delta_t, e \in F_\delta} V(H_{te}) \quad \text{for all } t \in [d].$$

► **Lemma 28.** Let $v \in T_t$ with $t \in \{0, \dots, d\}$. Then $\chi_{\mathbf{S}_0 \dots \mathbf{S}_{t-1}}(v) = \bar{0}$.

By Definition 27 we have

$$\begin{aligned} G &= \left(\dots (I(T_0))^{\mathbf{S}_0} \dot{\cup} \bigcup_{\delta \in \Delta_1, e \in F_\delta} H_{1e} \right)^{\mathbf{S}_1} \dot{\cup} \dots \dot{\cup} \bigcup_{\delta \in \Delta_d, e \in F_\delta} H_{de} \right)^{\mathbf{S}_d} \\ &= \left(I(T_0) \dot{\cup} \bigcup_{t \in [d], \delta \in \Delta_t, e \in F_\delta} H_{te} \right)^{\mathbf{S}_0 \mathbf{S}_1 \dots \mathbf{S}_d} \quad (\text{by Lemma 13 (e)}). \end{aligned} \quad (2)$$

Our goal is to show that in G we can FO-define flipping sets “equivalent to” $\mathbf{S}_0, \mathbf{S}_1, \dots, \mathbf{S}_d$. To that end, we introduce an auxiliary graph

$$L := \left(\dots \left((I(T_0))^{\mathbf{S}_0} \dot{\cup} I(T_1) \right)^{\mathbf{S}_1} \dot{\cup} \dots \dot{\cup} I(T_d) \right)^{\mathbf{S}_d} = (I(V(G)))^{\mathbf{S}_0 \mathbf{S}_1 \dots \mathbf{S}_d}. \quad (3)$$

We want to apply to $L = (I(V(G)))^{\mathbf{S}_0 \mathbf{S}_1 \dots \mathbf{S}_d}$ the results developed in the preceding section. First we show that relevant information on $E(L)$ can be FO-defined in G .

Let $t \in [d]$. For $\delta \in \Delta_t$ we fix pairwise distinct $e_1, e_2, e_3 \in F_\delta$. As for $i \in [3]$, $c(\delta) = \text{color}(V(H_{te_i}), \mathbf{S}_t \dots \mathbf{S}_d)$, we choose for $I(V(H_{te_i}))^{\mathbf{S}_t \dots \mathbf{S}_d}$ a representative system $(u_{i1}^\delta, \dots, u_{i|c(\delta)|}^\delta; \chi_{\mathbf{S}_t \dots \mathbf{S}_d}(u_{i1}^\delta), \dots, \chi_{\mathbf{S}_t \dots \mathbf{S}_d}(u_{i|c(\delta)|}^\delta))$ such that for $i, j \in [3]$ and $\ell \in [|c(\delta)|]$,

$$\chi_{\mathbf{S}_t \dots \mathbf{S}_d}(u_{i\ell}^\delta) = \chi_{\mathbf{S}_t \dots \mathbf{S}_d}(u_{j\ell}^\delta). \quad (4)$$

15:10 FO-Definability of Shrub-Depth

Then $VV_t := \{u_{i\ell}^\delta \mid \delta \in \Delta_t, i \in [3], \text{ and } \ell \in [c(\delta)]\}$ is the set of *voting vertices* in T_t . Let $c(0) := \text{color}(T_0, \mathbf{S}_0 \dots \mathbf{S}_d)$ and let $(u_1, \dots, u_{|c(0)|}; \chi_{\mathbf{S}_0 \dots \mathbf{S}_d}(u_0), \dots, \chi_{\mathbf{S}_0 \dots \mathbf{S}_d}(u_{|c(0)|}))$ be a representative system for $(I(T_0))^{\mathbf{S}_0 \dots \mathbf{S}_d}$. We define the *set VV of all voting vertices* by

$$VV := \{u_1, \dots, u_{|c(0)|}\} \cup \bigcup_{t \in [d]} VV_t.$$

One easily shows:

► **Lemma 29.**

(a) $\chi_{\mathbf{S}_0 \dots \mathbf{S}_d}(\{u \mid u \in VV\}) = \chi_{\mathbf{S}_0 \dots \mathbf{S}_d}(V(L))$.

(b) *The size of VV can be bounded in terms of d, q , and $\sum_{t \in [d]} |\Delta_t|$.*

Part (a) of the next lemma shows that we can decide in G whether $\{u, v\} \in E(L)$ between a voting vertex u and any other vertex v (see the full paper for a proof). Essentially, the majority opinion of the voting vertices decides. Part (b) is an immediate consequence of (a).

► **Lemma 30.**

(a) *Let $v \in V(G)$ and $u \in VV_t$ for some $t \geq 1$, say $u = u_{i\ell}^\delta$ where $\delta \in \Delta_t, i \in [3]$, and $\ell \in [c(\delta)]$. If $v \neq u$, then*

$$\{v, u\} \in E(L) \iff \text{there are } 1 \leq i_1 < i_2 \leq 3 \text{ such that } \{v, u_{i_1\ell}^\delta\}, \{v, u_{i_2\ell}^\delta\} \in E(G).$$

For $u \in \{u_1, \dots, u_{|c(0)|}\}$ with $v \neq u$ we have $(\{v, u\} \in E(L) \iff \{v, u\} \in E(G))$.

(b) *In G we can express in FO with parameters for the elements of VV whether $\{v, u\} \in E(L)$ for $v \in V(L)$ and $u \in VV$.*

However the information of part (a) doesn't allow us to compute $\chi_{\mathbf{S}_0 \dots \mathbf{S}_d}(v)$ for all $v \in V(L)$. Again we have the problem of ambiguity. We turn to this problem. By Lemma 29(a) we can choose vertices $u_1, \dots, u_m \in VV$ such that

$$\mathcal{R} := (u_1, \dots, u_m; \chi_{\mathbf{S}_0 \dots \mathbf{S}_d}(u_1), \dots, \chi_{\mathbf{S}_0 \dots \mathbf{S}_d}(u_m))$$

is a representative system for $L = (I(V(G)))^{\mathbf{S}_0 \mathbf{S}_1 \dots \mathbf{S}_d}$. In the preceding section we have seen how to obtain unambiguous $\mathbf{S}'_0, \dots, \mathbf{S}'_d$ with $L = (I(V(G)))^{\mathbf{S}'_0 \mathbf{S}'_1 \dots \mathbf{S}'_d}$. Let us recall how we did this. So assume \mathcal{R} is not unambiguous. Then there are distinct $j_1, j_2 \in [m]$ such that

$$\text{for all } i \in [m]: \quad \chi_{\mathbf{S}_0 \dots \mathbf{S}_d}(u_{j_1}) \oplus \chi_{\mathbf{S}_0 \dots \mathbf{S}_d}(u_i) = \chi_{\mathbf{S}_0 \dots \mathbf{S}_d}(u_{j_2}) \oplus \chi_{\mathbf{S}_0 \dots \mathbf{S}_d}(u_i).$$

Then we gave all vertices of color $\chi_{\mathbf{S}_0 \dots \mathbf{S}_d}(u_{j_2})$ the color $\chi_{\mathbf{S}_0 \dots \mathbf{S}_d}(u_{j_1})$. This could be problematic as we also want to preserve (2), that is, we also aim at:

$$G = \left(I(T_0) \dot{\cup} \bigcup_{t \in [d], \delta \in \Delta_t, e \in F_\delta} H_{te} \right)^{\mathbf{S}'_0 \mathbf{S}'_1 \dots \mathbf{S}'_d}. \quad (5)$$

If e.g. the vertex u_{j_2} is in H_{5e} and u_{j_1} is in $H_{3e'}$ and we give u_{j_2} the color of u_{j_1} , then already \mathbf{S}_3 (more precisely, \mathbf{S}'_3) could introduce edges between u_{j_2} and vertices in $H_{3e'}$ that destroy the validity of (5). In fact, the first equality of (2) implies that \mathbf{S}_3 cannot contain any vertex from H_{5e} . In the proof of our goal (the following proposition) in the full version we take care of this problem. In essence, we will always keep a vertex in its original tier.

► **Proposition 31.** *Let the (d, q) -tiered graph G and L be as above. There exist sequences of subsets $\mathbf{S}'_0, \dots, \mathbf{S}'_d$ of $V(G) = V(L)$ such that:*

(a) $G = \left(\dots \left(I(T_0)^{\mathbf{S}'_0} \dot{\cup} \dot{\bigcup}_{\delta \in \Delta_1, e \in F_\delta} H_{1e} \right)^{\mathbf{S}'_1} \dot{\cup} \dots \dot{\cup} \dot{\bigcup}_{\delta \in \Delta_d, e \in F_\delta} H_{de} \right)^{\mathbf{S}'_d}$ and thus,

$$G = \left(I(T_0) \dot{\cup} \dot{\bigcup}_{t \in [d], \delta \in \Delta, e \in F_\delta} H_{te} \right)^{\mathbf{S}'_0 \mathbf{S}'_1 \dots \mathbf{S}'_d}.$$

In particular, all Δ_t 's, F_δ 's, and H_{te} 's are the same as in (2) and for $t \in [d]$, $\delta \in \Delta_t$, and $e, e' \in F_\delta$ we have $\text{color}(V(H_{te}), \mathbf{S}'_t \dots \mathbf{S}'_d) = \text{color}(V(H_{te'}), \mathbf{S}'_t \dots \mathbf{S}'_d)$. Hence, also the first equality for G witnesses that G is a (d, q) -tiered graph.

(b) $L := \left(\dots \left(I(T_0)^{\mathbf{S}'_0} \dot{\cup} I(T_1) \right)^{\mathbf{S}'_1} \dot{\cup} \dots \dot{\cup} I(T_d) \right)^{\mathbf{S}'_d} = (I(V(G)))^{\mathbf{S}'_0 \mathbf{S}'_1 \dots \mathbf{S}'_d}$. Moreover, $\mathbf{S}'_0 \mathbf{S}'_1 \dots \mathbf{S}'_d$ is unambiguous with respect to L .

By (2) and Lemma 13(c), (d) we get the following immediate consequence of part (a).

► **Corollary 32.** $G^{\mathbf{S}'_0 \mathbf{S}'_1 \dots \mathbf{S}'_d} = I(T_0) \dot{\cup} \dot{\bigcup}_{t \in [d], \delta \in \Delta_t, e \in F_\delta} H_{te} = G^{\mathbf{S}'_0, \dots, \mathbf{S}'_d}$.

The main message of this section is the following: For (d, q) -tiered graphs once we have guessed the correct unambiguous representative system we can FO-define the edge relation of the graph $G^{\mathbf{S}'_0 \dots \mathbf{S}'_d}$.

7 From graphs of bounded SC-depth to tiered graphs

Here we reduce graphs of bounded SC-depth to tiered graphs (see Proposition 36). For this purpose it is useful to consider the generalized SC-depth of graphs obtained from the SC-depth by allowing “at the end” a bounded number of flipping sets.

► **Definition 33.** Let $d, q \in \mathbb{N}$. By $\text{GSC}(d, q)$ we denote the class of graphs of q -generalized SC-depth $\leq d$ (here q refers to the bound for the number of flipping sets in the last step). The classes $\text{GSC}(d, q)$ are defined as follows.

- (i) If the vertex set $V(G)$ of the graph G is a singleton, then $G \in \text{GSC}(0, q)$.
- (ii) Assume $d \geq 1$ and $G = \left(\dot{\bigcup}_{e \in F} G_e \right)^{\mathbf{S}}$ with $|\mathbf{S}| \leq q$ and $G_e \in \text{SC}(d-1)$ for every $e \in F$. Then $G \in \text{GSC}(d, q)$.

► **Lemma 34.**

- (a) $\text{SC}(d) = \text{GSC}(d, 1)$.
- (b) If $G \in \text{GSC}(1, q)$, then G is a $(0, q)$ -tiered graph.

Let $d \geq 2$ and $q \geq 1$ and $G \in \text{GSC}(d, q)$. Hence $G = \left(\dot{\bigcup}_{e \in F} G_e \right)^{\mathbf{S}}$, where $|\mathbf{S}| \leq q$ and $G_e \in \text{SC}(d-1)$ for every $e \in F$. We let F_0 be the set of $e \in F$ such that there is at most one $e' \in F$ with $e' \neq e$ and $\text{color}(V(G_{e'}), \mathbf{S}) = \text{color}(V(G_e), \mathbf{S})$. We partition $F \setminus F_0$ into sets $(F_\delta)_{\delta \in \Delta}$ such that for every $\delta \in \Delta$ there is a color $c(\delta)$ such that for all $e \in F_\delta$ we have $c(\delta) = \text{color}(V(G_e), \mathbf{S})$ and for distinct $\delta, \delta' \in \Delta$ we have $c(\delta) \neq c(\delta')$. By definition, $|\Delta| \leq 2^{2^q}$ and $|F_\delta| \geq 3$ for all $\delta \in \Delta$ (note that Δ may be empty). Observe that

$$|F_0| \leq 2 \cdot 2^{2^{|\mathbf{S}|}} = 2^{2^q+1}. \quad (6)$$

As $d \geq 2$, for every $e \in F_0$ the graph G_e can be written in the form $G_e = \left(\dot{\bigcup}_{f \in F_e} G_{ef} \right)^{S_e}$, where all G_{ef} are in $\text{SC}(d-2)$ and $S_e \subseteq V(G_e)$. Let \mathbf{T} be the sequence of all sets S_e with $e \in F_0$. By (6), $|\mathbf{T}| \leq 2^{2^q+1}$. We define the graph G' by (note that F_0 may be empty)

$$G' := \left(\dot{\bigcup}_{e \in F_0, f \in F_e} G_{ef} \right)^{\mathbf{T}}.$$

The following statements result directly from the definitions.

15:12 FO-Definability of Shrub-Depth

► **Lemma 35.**

(a) $G' \in \text{GSC}(d-1, 2^{2^q+1})$.

(b) $G = \left(G' \dot{\cup}_{\delta \in \Delta, e \in F_\delta} G_e \right)^{\mathbf{S}}$ with $|\Delta| \leq 2^{2^q}$, $G_e \in \text{SC}(d-1)$ for $e \in F_\delta$ and $\delta \in \Delta$.

We define the function h by: $h(0, q) := 0$, $h(1, q) := q$, and $h(d+1, q) := h(d, 2^{2^q+1})$ for $d \geq 1$. Now a simple induction shows (for a proof see the full paper):

► **Proposition 36.** *Let $d \geq 1$ and $G \in \text{GSC}(d, q)$. Then G is a $(d-1, h(d, q))$ -tiered graph, i.e., G can be written in the form*

$$G = \left(\dots \left((I(T_0))^{\mathbf{S}_1} \dot{\cup}_{\delta \in \Delta_1, e \in F_\delta} H_{1e} \right)^{\mathbf{S}_2} \dot{\cup} \dots \dot{\cup}_{\delta \in \Delta_{d-1}, e \in F_\delta} H_{d-1 e} \right)^{\mathbf{S}_d},$$

where $|\mathbf{S}_t| \leq h(d, q)$ for $t \in [d]$ and where $|\Delta_t| \leq h(d, q)$ for $t \in [d-1]$. In addition, for $d \geq 2$ we have $H_{te} \in \text{SC}(t-1)$ for $t \in [d-1]$, $\delta \in \Delta_t$, and $e \in F_\delta$.

8 FO-definition of tree-models for graphs of bounded shrub-depth

Using the results of the preceding sections we first prove that there is a computable function $d \mapsto \varphi_d$ where φ_d is an FO-sentence whose class of models has bounded shrub-depth and contains $\text{SC}(d)$. Then we show how using ideas from [15] we can refine this proof to obtain Theorem 2, i.e., the FO-definability of tree-models for graphs of bounded shrub-depth.

► **Proposition 37.** *Let $d \in \mathbb{N}$ and $\Gamma := h(d, 1)$. There is an FO-sentence φ_d with (a) and (b).*

(a) *If $G \in \text{SC}(d)$, then $G \models \varphi_d$.*

(b) *If $G \models \varphi_d$, then $\text{SC}(G) \leq \frac{d \cdot (d+1) \cdot \Gamma}{2}$.*

For later purposes we assume that φ_d also expresses that E is irreflexive and symmetric.

Proof. We set $\varphi_0 := \forall x \forall y (x = y \wedge \neg Exy)$. Let $d \geq 1$ and G be a graph with $\text{SC}(G) = d$. Hence $G \in \text{GSC}(d, 1)$ by Lemma 34(a). By Proposition 36 the graph G is $(d-1, \Gamma)$ -tiered, thus G can be written in the form

$$G = \left(\dots \left((I(T_0))^{\mathbf{S}_1} \dot{\cup}_{\delta \in \Delta_1, e \in F_\delta} H_{1e} \right)^{\mathbf{S}_2} \dot{\cup} \dots \dot{\cup}_{\delta \in \Delta_{d-1}, e \in F_\delta} H_{d-1 e} \right)^{\mathbf{S}_d},$$

where in particular, $|\mathbf{S}_t| \leq \Gamma$ for $t \in [d]$ and $H_{te} \in \text{SC}(t-1)$ for every $t \in [d-1]$, $\delta \in \Delta_t$ and $e \in F_\delta$. By Proposition 31 we can assume that for

$$L := \left(\dots \left((I(T_0))^{\mathbf{S}_1} \dot{\cup} I(T_1) \right)^{\mathbf{S}_2} \dot{\cup} \dots \dot{\cup} I(T_{d-1}) \right)^{\mathbf{S}_d} = (I_{V(G)})^{\mathbf{S}_1 \dots \mathbf{S}_d}$$

$\mathbf{S}_1 \dots \mathbf{S}_d$ is unambiguous (with respect to L). Here $T_i = \dot{\cup}_{\delta \in \Delta_i, e \in F_\delta} V(H_{ie})$ for $i \in [d-1]$.

By Lemma 29(b) the size of voting vertices can be bounded in terms of d . Thus as φ_d we can take an FO-sentence which (existentially) guesses voting vertices for such an unambiguous $\mathbf{S}_1 \dots \mathbf{S}_d$ and guesses a subset of these vertices which together with their $\chi_{\mathbf{S}_1 \dots \mathbf{S}_d}$ -colors (which are also guessed) yield a representative system for L . Then it defines the $\mathbf{S}_1, \dots, \mathbf{S}_d$ and expresses that every connected component of $G^{\mathbf{S}_1 \dots \mathbf{S}_d}$ satisfies φ_{d-1} . Now the validity of (a) should be clear.

We prove (b) by induction on d . Of course, (b) holds for $d = 0$. So assume that $d \geq 1$ and that the statement (b) is true for $d-1$. Let $G \models \varphi_d$. Then there are $\mathbf{S}_1, \dots, \mathbf{S}_d$ such that every connected component H of $G_1 := G^{\mathbf{S}_1 \dots \mathbf{S}_d}$ satisfies φ_{d-1} . Hence, $\text{SC}(H) \leq \frac{(d-1) \cdot d \cdot \Gamma}{2}$ by induction hypothesis. As $G = \left(\dot{\cup}_{H \text{ connected component of } G_1} H \right)^{\mathbf{S}_1 \dots \mathbf{S}_d}$ by Lemma 16(c), we get

$$\begin{aligned} \text{SC}(G) &\leq d \cdot \Gamma + \max\{\text{SC}(H) \mid H \text{ connected component of } G_1\} \\ &\leq d \cdot \Gamma + \frac{(d-1) \cdot d \cdot \Gamma}{2} \leq \frac{d \cdot (d+1) \cdot \Gamma}{2}. \end{aligned} \quad \blacktriangleleft$$

We turn to a proof of Theorem 2. It suffices to show the following result (cf. Proposition 17).

► **Theorem 38.** *Let $d \in \mathbb{N}$. There is an FO-interpretation that assigns to every ordered graph $(G, <)$ with $G \in \text{SC}(d)$ a tree-model.*

Proof. We know that $\text{SC}(d) \subseteq \text{TM}_{2^d}(d)$ (see Proposition 17(b)). We recall the proof of this result from [15, Theorem 3.6]. For an SC-derivation W witnessing $G \in \text{SC}(d)$ it constructs a tree $\mathcal{T}(W) \in \text{TREE}[2^d, d]$ (see page 4 for the definition of $\text{TREE}[m, d]$), which together with a signature D will be a tree-model of G . We denote the labels by L_b with $b \in \{0, 1\}^d$. Essentially the tree $\mathcal{T}(W)$ is the “tree of the SC-derivation”: The leaves of $\mathcal{T}(W)$ are the vertices of G . Each internal node t of $\mathcal{T}(W)$ is associated with a flipping set S_t . By adding nodes with the empty flipping set we can assume that every path from the root to a leaf has length d . Let $v \in V(G)$ and let $t_0 = r, t_1, \dots, t_d = v$ be the path from the root r of $\mathcal{T}(W)$ to v . Then v gets the color (= label) L_b if for $i \in [d]$ we have $(b_i = 1 \iff v \in S_{t_i})$. The pair $\{u, v\}$ is an edge of G if and only if u and v are simultaneously contained in an odd number of flipping sets S_t , where t ranges over all internal nodes. This can easily be determined from the colors of u and v , and from the depth of their least common ancestor $u \wedge v$. So it yields the definition of the corresponding signature $D(d)$ (note that $D(d)$ doesn't depend on the concrete SC-derivation but only on d).

Now let $<^G$ be an arbitrary order of $V(G)$. As $G \in \text{SC}(d)$, the graph G is a model of the sentence φ_d of Proposition 37. The process described in φ_d shows how one gets an SC-derivation witnessing $\text{SC}(G) \leq g(d) := \frac{d \cdot (d+1) \cdot \Gamma}{2}$. Using $<^G$ we can describe in FO such a derivation $W(<^G)$: According to φ_d first we guess voting vertices with certain properties, now we choose the lexicographically $<^G$ -first voting vertices with these properties. Then by φ_d we guess a subset of these vertices together with their colors as representative system. Now we choose the lexicographically $<^G$ -smallest such subset and the “smallest” colors which do the job. Then by φ_d we get $\mathbf{S}_1, \dots, \mathbf{S}_d$ with $|\mathbf{S}_i| \leq \Gamma$ for $i \in [d]$. W.l.o.g. we may assume that $|\mathbf{S}_i| = \Gamma$. The root of the tree $\mathcal{T}(W(<^G))$ starts with a path of length $d \cdot \Gamma - 1$ ending with a node t_0 . The nodes of the path are associated with the flipping sets in $\mathbf{S}_1, \dots, \mathbf{S}_d$. Then for every $v \in V(G)$, which in $G^{\mathbf{S}_1 \dots \mathbf{S}_d}$ is not in the connected component of an $<^G$ -smaller element, we add an derivation for this component according to φ_{d-1} as one child of t_0 . By this procedure we get a tree in $\mathcal{T}(W(<^G)) \in \text{TREE}[2^{g(d)}, g(d)]$, where $g(d) := \frac{d \cdot (d+1) \cdot \Gamma}{2}$.

In this way we get an FO-interpretation \mathcal{I} defining in $(G, <^G)$ with $G \models \varphi_d$ the tree-model $\mathcal{T}(W(<^G))$ of G . That is, we can present (it is tedious but straightforward) a tuple of FO-formulas $\mathcal{I} = (\varphi_{\text{uni}}(\bar{x}), \varphi_P(\bar{x}, \bar{y}), (\varphi_{L_b}(\bar{x}))_{b \in \{0,1\}^{g(d)}})$, where $\bar{x} = x_0, \dots, x_{g(d)}$ and $\bar{y} = y_0, \dots, y_{g(d)}$ such that $(G, <^G)^{\mathcal{I}} := (\varphi_{\text{uni}}^{(G, <^G)}, \varphi_P^{(G, <^G)}, (\varphi_{L_b}^{(G, <^G)})_{b \in \{0,1\}^{g(d)}}$ is isomorphic to the tree-model $\mathcal{T}(W(<^G))$. For example, $\varphi_{\text{uni}}^{(G, <^G)} := \{(v_0, \dots, v_{g(d)}) \in G^{g(d)} \mid (G, <^G) \models \varphi_{\text{uni}}(\bar{v})\}$ is the universe of $(G, <^G)^{\mathcal{I}}$ and $\varphi_P^{(G, <^G)} := \{(\bar{v}, \bar{w}) \mid \bar{v}, \bar{w} \in \varphi_{\text{uni}}^{(G, <^G)}, (G, <^G) \models \varphi_P(\bar{v}, \bar{w})\}$ is the parent-child relation of $(G, <^G)^{\mathcal{I}}$. ◀

As already mentioned in the Introduction we get Theorem 1 from the preceding result in the same way as we did for tree-depth in [3]. For the sake of completeness let us recall that for a class K of graphs the parameterized model-checking $p\text{-MC}(K, \text{MSO})$ for MSO on K is defined by

Instance: A graph $G \in K$ and an MSO-sentence φ .
Parameter: $k \in \mathbb{N}$.
Problem: Decide if $k = |\varphi|$ and $\mathcal{A} \models \varphi$.

In the next section we will apply a further consequence of Theorem 38, an improvement of Proposition 3, which again can be obtained as the corresponding result for tree-depth in [3]:

15:14 FO-Definability of Shrub-Depth

► **Proposition 39.** *Let $d \in \mathbb{N}$. There is an algorithm that assigns to every $\text{MSO}[\{E\}]$ -sentence φ an $\text{FO}[\{E, <\}]$ -sentence φ^+ such that for every ordered graph $(G, <)$ with $G \models \varphi_d$,*

$$G \models \varphi \iff (G, <^G) \models \varphi^+.$$

9 MSO = FO on classes of bounded shrub-depth

The goal of this section is to show that in models of φ_d every $\text{MSO}[\{E\}]$ -sentence is equivalent to an $\text{FO}[\{E\}]$ -sentence, i.e., that we can omit the order relation used in Proposition 39. To get this result we use Craig's interpolation which is known to be true only if we consider finite and infinite models. However, we have introduced the notion of SC-depth for finite graphs only. So let us extend this concept to infinite graphs. *If not stated otherwise explicitly, in the following "graph" always means a finite or infinite graph.*

Let G be a graph and $S \subseteq V(G)$, then G^S is defined as in the finite case.

► **Definition 40.** *The class $\text{SC}(d)$ (extending the "old" $\text{SC}(d)$) is defined by induction on d :*

$\text{SC}(0)$ is the class of graphs whose vertex set is a singleton.

Assume that I is a set with $|I| \geq 1$ and that for $i \in I$ the graphs G_i are in $\text{SC}(d)$ and have pairwise disjoint vertex sets. Then $(\dot{\bigcup}_{i \in I} G_i)^S \in \text{SC}(d+1)$ for every $S \subseteq \bigcup_{i \in I} V(G_i)$.

The SC-depth of a graph is the least $d \in \mathbb{N}$ such that $G \in \text{SC}(d)$.

Not every graph has an SC-depth, that is, the analogue of Lemma 15 fails. For example, an infinite path has no SC-depth. We leave it to the reader to generalize the notion of shrub-depth and to realize that Lemma 8, the analogue of Lemma 15 for shrub-depth, fails. But all other results in Section 3–Section 7 and Proposition 37 of Section 8 are true for graphs in the way stated (or with obvious changes). One exception: In the definition of tiered graph we have to require that the H_{te} 's have an SC-depth.

Let us look what happens with Theorem 38 of Section 8. On page 13 for the first time we considered orders on graphs. Once we have an order $<^G$ in a finite model G of the sentence φ_d we got a canonical SC-derivation $W(<^G)$ of G , which could be described by an interpretation \mathcal{I} . When defining the derivation $W(<^G)$ we used a few times the property that every nonempty subset of $V(G)$ contains a $<^G$ -least element or the same property for subsets of $V(G)^r$ for some $r \geq 1$ with respect to the lexicographic order $<_{\text{lex}}^G$ of r -tuples induced by $<^G$. All these subsets were definable by an $\text{FO}[\{E\}]$ -formula $\psi(\bar{x}, \bar{y})$, where $|\bar{x}| = r$ and the variables in \bar{y} are parameters. Then we used the fact that the following sentence is true in every *finite* $(G, <^G)$:

$$\text{least-element}(\psi) := \forall \bar{y} \left(\exists \bar{x} \psi(\bar{x}, \bar{y}) \rightarrow \exists \bar{x} (\psi(\bar{x}, \bar{y}) \wedge \forall \bar{x}' (\psi(\bar{x}', \bar{y}) \rightarrow \bar{x} \leq_{\text{lex}} \bar{x}')) \right).$$

Here $|\bar{x}'| = |\bar{x}|$. Let

$$\text{LE}(<) := \{ \text{least-element}(\psi) \mid \psi = \psi(\bar{x}, \bar{y}) \in \text{FO}[\{E\}] \text{ and } |\bar{x}| \geq 1 \}$$

be the set of least-element sentences for all $\text{FO}[\{E\}]$ -formulas. Furthermore set

$$\text{LE}^*(<) := \text{LE}(<) \cup \{ \text{"}E \text{ is irreflexive and symmetric"} \} \cup \{ \text{"} < \text{ is an order"} \}.$$

Then we can reformulate Proposition 39 and extend it to arbitrary graphs:

► **Proposition 41.** *For every $\text{MSO}[\{E\}]$ -sentence φ there exists an $\text{FO}[\{E, <\}]$ -sentence φ^+ such that $\text{LE}^*(<) \cup \{\varphi_d\} \models (\varphi \leftrightarrow \varphi^+)$.*

Now we turn to the main result of this section, which extends Theorem 4.

► **Theorem 42.** *Let $d \geq 1$. In models of φ_d (hence, in particular, in graphs in $\text{SC}(d)$) every $\text{MSO}[\{E\}]$ -sentence φ is equivalent to an $\text{FO}[\{E\}]$ -sentence ψ . Moreover, there is an algorithm that on input φ yields ψ .* Proof: full paper.

► **Corollary 43.**

- (a) *There is a computable function $d \mapsto \psi_d$, where $\psi_d \in \text{FO}[\{E\}]$ axiomatizes $\text{SC}(d)$.*
- (b) *There is a computable function $(m, d) \mapsto \psi_{m,d}$, where $\psi_{m,d} \in \text{FO}[\{E\}]$ axiomatizes $\text{TM}_m(d)$.*

Proof.

- (a) We know from Proposition 18 that there is a computable function $d \mapsto \rho_d$, where $\rho_d \in \text{MSO}[\{E\}]$ axiomatizes $\text{SC}(d)$. By the preceding theorem we effectively get a $\psi^d \in \text{FO}[\{E\}]$ equivalent to ρ_d in models of φ_d . Then $\psi_d := \varphi_d \wedge \psi^d$ axiomatizes $\text{SC}(d)$.
- (b) As by Proposition 17 (a) we have $\text{TM}_m(d) \subseteq \text{SC}(d \cdot m \cdot (m+1))$, we can argue for $\text{TM}_m(d)$ similarly, now using Proposition 12. ◀

10 The excursion to the infinite yields further results

The following result is an effective version of the result [15, Corollary 5.6] mentioned on page 5 and on page 7.

► **Theorem 44.** *There is an algorithm that applied to (m, d) eventually stops and outputs a finite set $F_{m,d}$ of finite graphs such that a graph is in $\text{TM}_m(d)$ if and only if it excludes the graphs in $F_{m,d}$ as induced subgraphs. The analogous result holds for $\text{SC}(d)$.*

Proof. As $\text{TM}_m(d)$ is closed under induced subgraphs, by the Łoś-Tarski Theorem of classical model theory (cf. [2, 18]) we effectively find (from $\psi_{m,d}$ of Corollary 43) a universal FO -sentence $\nu_{m,d}$ axiomatizing $\text{TM}_m(d)$ (recall that a universal FO -sentence is a sentence of the form $\forall x_1 \dots \forall x_n \chi$ with quantifier-free χ). Every universal sentence just expresses that there is a finite set of finite graphs that are excluded as induced subgraphs. ◀

Various applications of the same flavour may be obtained using the following lemma. We will prove this lemma and present various applications in the full version of this paper.

► **Lemma 45.** *Let $d \geq 1$ and $K \subseteq \text{SC}(d)$ be a class closed under induced subgraphs. For every MSO -sentence φ , if the class of finite models of φ in K is closed under induced subgraphs, so is the class of models of φ in K .*

For the class of finite graphs Rossman [21] has proved the analogue of the result of classical model theory that a sentence preserved under homomorphisms is equivalent to an existential-positive FO -sentence (a sentence is *positive* if it does not contain the negation symbol). Along the previous lines one can show that this preservation theorem holds for $\text{TM}_m(d)$: A sentence preserved under homomorphisms between finite graphs in $\text{TM}_m(d)$ is equivalent to an existential-positive FO -sentence.

References

- 1 D. A. Mix Barrington, N. Immerman, and H. Straubing. On Uniformity within NC^1 . *Journal of Computer and System Sciences*, 41(3):274–306, 1990.
- 2 C. C. Chang and H. J. Keisler. *Model Theory*. Studies in Logic and the Foundations of Mathematics. Elsevier, 3rd edition, 1992.

- 3 Y. Chen and J. Flum. Tree-depth, quantifier elimination, and quantifier rank. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 225–234, 2018.
- 4 B. Courcelle. Graph Rewriting: An Algebraic and Logic Approach. In *Handbook of Theoretical Computer Science, Volume B:*, pages 193–242. Elsevier and MIT Press, 1990.
- 5 B. Courcelle, J. Makowsky, and U. Rotics. Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width. *Theory of Computing Systems*, 33(2):125–150, 2000.
- 6 W. Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *Journal of Symbolic Logic*, 22(3), 1957.
- 7 K. Eickmeyer, J. van den Heuvel, K. Kawarabayashi, S. Kreutzer, P. Ossona de Mendez, M. Pilipczuk, D. Quiroz, R. Rabinovich, and S. Siebertz. Model-Checking on Ordered Structures. *CoRR*, abs/1812.08003, 2018.
- 8 M. Elberfeld, M. Grohe, and T. Tantau. Where First-Order and Monadic Second-Order Logic Coincide. *ACM Transactions on Computational Logic*, 17(4):25:1–25:18, 2016.
- 9 M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic*, 130(1-3):3–31, 2004.
- 10 J. Gajarský and P. Hliněný. Faster Deciding MSO Properties of Trees of Fixed Height, and Some Consequences. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, pages 112–123, 2012.
- 11 J. Gajarský and P. Hliněný. Kernelizing MSO Properties of Trees of Fixed Height, and Some Consequences. *Logical Methods in Computer Science*, 11(1), 2015.
- 12 J. Gajarský, P. Hliněný, D. Lokshtanov, J. Obdržálek, S. Ordyniak, M. S. Ramanujan, and S. Saurabh. FO model checking on posets of bounded width. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 963–974, 2015.
- 13 J. Gajarský, P. Hliněný, J. Obdržálek, D. Lokshtanov, and M. S. Ramanujan. A New Perspective on FO Model Checking of Dense Graph Classes. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS’16, New York, NY, USA, July 5-8, 2016*, pages 176–184, 2016.
- 14 R. Ganian, P. Hliněný, D. Král, J. Obdržálek, J. Schwartz, and J. Teska. FO model checking of interval graphs. *Logical Methods in Computer Science*, 11(4), 2015.
- 15 R. Ganian, P. Hliněný, J. Nešetřil, J. Obdržálek, P. Ossona de Mendez, and R. Ramadurai. When Trees Grow Low: Shrubs and Fast MSO₁. In *Mathematical Foundations of Computer Science 2012 - 37th International Symposium, MFCS 2012, Bratislava, Slovakia, August 27-31, 2012. Proceedings*, pages 419–430, 2012.
- 16 M. Grohe, S. Kreutzer, R. Rabinovich, S. Siebertz, and K. Stavropoulos. Coloring and Covering Nowhere Dense Graphs. *SIAM Journal on Discrete Mathematics*, 32(4):2467–2481, 2018.
- 17 Y. Gurevich. Toward logic tailored for computational complexity. In *Computation and Proof Theory, Lecture Notes in Mathematics*, pages 1104:175–216, 1984.
- 18 W. Hodges. *Model theory*, volume 42 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 1993.
- 19 M. Lampis. Algorithmic Meta-theorems for Restrictions of Treewidth. *Algorithmica*, 64(1):19–37, 2012.
- 20 M. Pilipczuk, S. Siebertz, and S. Toruńczyk. Parameterized circuit complexity of model-checking on sparse structures. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 789–798, 2018.
- 21 B. Rossman. Homomorphism preservation theorems. *Journal of the ACM*, 55(3):15:1–15:53, 2008.


Taylor expansion for Call-By-Push-Value

Jules Chouquet 

IRIF UMR 8243, Université de Paris, CNRS, France

<https://www.irif.fr/~chouquet/>

Jules.Chouquet@irif.fr

Christine Tasson 

IRIF UMR 8243, Université Paris Diderot, Sorbonne Paris Cité, CNRS, France

<https://www.irif.fr/~tasson/>

christine.tasson@irif.fr

Abstract

The connection between the Call-By-Push-Value lambda-calculus introduced by Levy and Linear Logic introduced by Girard has been widely explored through a denotational view reflecting the precise ruling of resources in this language. We take a further step in this direction and apply Taylor expansion introduced by Ehrhard and Regnier. We define a resource lambda-calculus in whose terms can be used to approximate terms of Call-By-Push-Value. We show that this approximation is coherent with reduction and with the translations of Call-By-Name and Call-By-Value strategies into Call-By-Push-Value.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Call-By-Push-Value, Quantitative semantics, Taylor expansion, Linear Logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.16

Funding This work was supported by french ANR project Rapido (ANR number: ANR-14-CE25-0007).

Acknowledgements The authors thank the ANR project Rapido, together with Lionel Vaux and Thomas Ehrhard for their useful advises and fertile discussions.

1 Introduction

Linear Logic [15] has been introduced by Girard as a refinement of Intuitionistic Logic that take into account the use, reuse or erasing of formulas. In order to mark formulas that can be reused or erased, Girard introduced the *exponential* $!X$ and considered a *linear implication* $X \multimap Y$. Following the proof/program correspondence paradigm, Linear Logic can be used to type λ -calculus according to a chosen reduction strategy as Call-By-Name or Call-By-Value. Abstraction terms $\lambda x.M$ usually typed by $X \Rightarrow Y$ will be typed as $!X \multimap Y$ when following a Call-By-Name evaluation strategy and by $!(X \multimap Y)$ when following a Call-By-Value strategy. Therefore, both evaluation strategies can be faithfully encoded in Linear Logic.

Levy followed a related goal when he introduced Call-By-Push-Value [21]: having a lambda calculus where both Call-By-Name and Call-By-Value can be taken into account. Since its introduction this calculus has been related to the Linear Logic approach [4, 12, 6, 22, 20]. We adopt this latest presentation which differentiates two kinds of types: positive and general types used for typing two kinds of terms: values and general terms respectively. The marker $!I$ is used to transform a general type I into a value type $!I$ which can be erased, used and duplicated. The idea behind $!$ is to stop the evaluation of the terms typed by $!I$ by placing them into thunks (*i.e.* putting them into boxes).

The purpose of this article is to push further the relations between Call-By-Push-Value and Linear Logic and to underline the resource consumption at play. For this we use syntactical Taylor expansion, that reflects Taylor expansion into semantics. Indeed, several semantics of Linear Logic and λ -calculus are interpreting types as topological vector spaces and terms as smooth functions that enjoy Taylor expansion [5, 7, 8, 18]. Indeed, those functions can



© Jules Chouquet and Christine Tasson;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 16; pp. 16:1–16:16

Leibniz International Proceedings in Informatics



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

be written as power series whose coefficients are computed thanks to a derivative operator. The syntactical Taylor expansion enable the representation of terms as a combination of approximants named *resource terms*.

Taylor expansion has first been introduced by Ehrhard and Regnier while they presented the differential λ -calculus [9], they noticed that it was possible to give a syntactical version of Taylor formula, and that this object was defined on the multilinear fragment of differential λ -calculus. It consists in associating to a λ -term an infinite series of resource terms, that enjoy a linearity property, in the following sense: resource calculus is endowed with an operational semantics similar to λ -calculus, but with no duplication nor erasing of subterms during reduction. As, in analysis analytic maps are approximated by series of monomials, here λ -terms are approximated by series of resource terms. Taylor expansion gives a natural semantics, where the reduction rules of resource calculus aim to identify the terms having the same interpretation in a denotational model. In particular, the normal form of Taylor expansion (or *Taylor normal form*) is a pleasant notion of approximation of normal forms in various λ -calculi, and is strongly linked to the notion of Böhm trees, since Ehrhard and Regnier's seminal works [10]. This link has been extended in several direction, see *e.g.* Vaux [27] for algebraic λ -calculus, Kerinec, Manzonetto and Pagani [17] for Call-By-Value calculus, or Dal Lago and Leventis [19] for probabilistic λ -calculus. Let us also mention two other related approaches to approximation of λ -calculus with polyadic terms instead of resource terms [23, 24]. Taylor expansion has also been studied for the *Bang Calculus*, an untyped analogue of Call-By-Push-Value, by Guerrieri and Ehrhard [13] and then by Guerrieri and Manzonetto [16].

We propose, following that fertile discipline, a syntactical Taylor expansion for Λ_{pv} , which is the Linear Logic-oriented presentation of Call-By-Push-Value we use (and corresponds to Λ_{hp} in Ehrhard's paper [12]).

A first difficulty we have to tackle, is the fact that designing a convenient resource calculus, say Δ_{pv} , that respects Λ_{pv} dynamics is not trivial. In particular, in a redex, the argument is a value but is not necessary of exponential type. Then, the argument of a resource redex shall not be necessarily a multiset, while it is always the case in Call-By-Name and Call-By-Value resource calculi, as it ensures the reductions are linear. The semantical reason of that phenomenon is that in a quantitative model of Λ_{pv} , all values with a positive type are freely duplicable, thanks to the coalgebras morphisms associated to those types' interpretation. The solution we adopt is to give a syntactical account to those morphisms in the reduction rules, so as to Δ_{pv} stays consistent with Call-By-Push-Value operational and denotational semantics, while keeping the resource reduction linear.

We can then consider a Taylor expansion, as a function from Λ_{pv} to sets of terms in Δ_{pv} , that consists of *approximants*. Once this framework is set, we are able to show that the properties of Call-By-Push-Value, relative to the embeddings of various strategies of evaluation, can be transported at the resource level.

The principal result of the paper is the simulation of Λ_{pv} reductions in full Taylor expansion, where resource terms take coefficients in a commutative semiring. The key ingredients for this simulation to run are intrinsic to the properties of Δ_{pv} : the dynamics of reduction must reflect the reduction of Λ_{pv} , and the mechanisms of the calculus must enjoy combinatorial properties, so that the coefficients commute with the simulation. More precisely, it means that for $M, N \in \Lambda_{\text{pv}}$ such that M reduces to N , if Taylor expansion of M is equal to $\sum_{i \in I} a_i m_i$, where a_i are coefficients taken in a semiring, and m_i are resource terms approximating M , then we have a notion of reduction such that $\sum_{i \in I} a_i m_i \Rightarrow \sum_{j \in I} a_j n_j$, and for each resource term n , its coefficient in the latter combination is the same as its coefficient in the Taylor expansion of N .

Contents of the paper

We first present (Section 2) Λ_{pv} as the starting point of our study, describing its operational semantics, provide examples of its expressive power, and give elements of its denotational semantics relative to coalgebras. We introduce and develop in Section 3 the resource calculus Δ_{pv} together with its operational semantics. Then, in Section 4, we define Taylor expansion for Λ_{pv} . First, in a qualitative way, with sets of approximants, where we show that it allows the simulation of Λ_{pv} reductions. We also describe how the embeddings of Call-By-Name and Call-By-Value into Call-By-Push-Value are transported at the resource level. Finally, we introduce quantitative Taylor expansion, with coefficients, and prove the commutation property between Taylor expansion and reduction that demonstrates that Taylor expansion is compatible with Λ_{pv} operational semantics.

Terminology and notations

We write \mathbf{N} for the set of natural numbers, and \mathfrak{S}_k for the group of permutations on $\{1, \dots, k\}$. For a term m , and a variable x , we denote as $\text{deg}_x(m)$ the number of free occurrences of x in m . These occurrences might be written $x_1, \dots, x_{\text{deg}_x(m)}$, while all referring to x .

Finite multisets of elements of a set X are written $\bar{x} = [x_1, \dots, x_k]$ for any $k \in \mathbf{N}$, and are functions from X to \mathbf{N} . We use the additive notation $\bar{x} + \bar{x}'$ for the multiset such that for all $y \in X$, $(\bar{x} + \bar{x}')(y) = \bar{x}(y) + \bar{x}'(y)$. The size of \bar{x} is written $|\bar{x}|$ and is equal to $\sum_{y \in X} \bar{x}(y)$. We denote as $X^!$ the set of all finite multisets of elements of X . We might write $(x, \dots, x)_k$ for tuples or $[x, \dots, x]_k$ for multisets to denote k occurrences of the same element x .

If σ is a linear combination of terms $\sum_{i \in I} a_i \cdot m_i$, we use the notation $\lambda x \sigma = \sum_{i \in I} a_i \cdot \lambda x m_i$, $\text{der}(\sigma) = \sum_{i \in I} a_i \cdot \text{der}(m_i)$, and $\sigma^! = \sum_{k \in \mathbf{N}} \sum_{i_1, \dots, i_k \in I} a_{i_1} \dots a_{i_k} \cdot [m_{i_1}, \dots, m_{i_k}]$. In the same way, if $\tau = \sum_{j \in J} a_j \cdot n_j$, we write $(\sigma, \tau) = \sum_{i \in I} \sum_{j \in J} a_i a_j \cdot (m_i, n_j)$. $\langle \sigma \rangle \tau = \sum_{i \in I} \sum_{j \in J} a_i a_j \cdot \langle m_i \rangle n_j$. This notation corresponds to the linearity of syntactic constructors with respect to potentially infinite sums of terms that will appear in Taylor expansion.

2 Call-By-Push-Value

2.1 Syntax and operational semantics

We consider a presentation of Call-By-Push-Value coming from Ehrhard [12], and convenient for its study through Linear Logic semantics.

► **Definition 1** (Call-By-Push-Value calculus Λ_{pv}).

$$\Lambda_{\text{pv}} : M ::= x \mid \lambda x M \mid \langle M \rangle M \mid \text{case}(M, y \cdot M, z \cdot M) \mid \mathbf{fix}_x(M) \mid (M, M) \mid \pi_1(M) \mid \pi_2(M) \mid M^! \mid \text{der}(M) \mid \iota_1(M) \mid \iota_2(M)$$

We distinguish a subset of Λ_{pv} , the values :

$$V ::= x \mid M^! \mid (V, V) \mid \iota_1(M) \mid \iota_2(M)$$

Positive types: $A, B ::= !I \mid A \otimes B \mid A \oplus B$

General types : $I, J ::= A \mid A \multimap I \mid \top$

The typing rules are given in Figure 1 and reduction rules are given below:

$$\begin{array}{ll} \langle \lambda x M \rangle V \rightarrow_{\text{pv}} M[V/x] & \text{der}(M^!) \rightarrow_{\text{pv}} M \\ \pi_i(V_1, V_2) \rightarrow_{\text{pv}} V_i & \mathbf{fix}_x(M) \rightarrow_{\text{pv}} M[(\mathbf{fix}_x(M))^! / x] \\ \text{case}(\iota_i(V), x_1 \cdot M_1, x_2 \cdot M_2) \rightarrow_{\text{pv}} M_i[V/x_i] & \end{array}$$

$$\begin{array}{c}
\frac{}{\Gamma, x : A \vdash x : A} \quad \frac{\Gamma \vdash M : I}{\Gamma \vdash M^! : !I} \quad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x M : A \multimap B} \quad \frac{\Gamma \vdash M : A \multimap I \quad \Delta \vdash N : A}{\Gamma, \Delta \vdash \langle M \rangle N : I} \\
\\
\frac{\frac{\Gamma \vdash M : A \quad \Delta \vdash N : B}{\Gamma, \Delta \vdash (M, N) : A \otimes B} \quad \frac{\Gamma \vdash M : A_1 \otimes A_2}{\Gamma \vdash \pi_i(M) : A_i} \quad i \in \{1, 2\}}{\Gamma \vdash M : A_i} \quad i \in \{1, 2\} \quad \frac{\Gamma \vdash m : !A}{\Gamma \vdash \mathbf{der}(m) : A} \\
\frac{\Gamma \vdash M_1 : A \oplus B \quad \Delta \vdash M_2 : I \quad \Theta \vdash M_3 : I}{\Gamma, \Delta, \Theta \vdash \mathbf{case}(M_1, y \cdot M_2, z \cdot M_3) : I} \quad \frac{\Gamma, x : !I \vdash M : I}{\Gamma \vdash \mathbf{fix}_x(M) : I}
\end{array}$$

■ **Figure 1** Typing rules for Λ_{pv} .

We define evaluation contexts E , for all terms M, N .

$$E ::= [] \mid \langle M \rangle E \mid \langle E \rangle M \mid \pi_i(E) \mid \iota_i(E) \mid (M, E) \mid (E, M) \mid \mathbf{case}(E, x \cdot M, y \cdot N) \mid \mathbf{der}(E)$$

and we set as an additional reduction rule $E[M] \rightarrow_{\text{pv}} E[N]$ for every M, N such that $M \rightarrow_{\text{pv}} N$.

2.2 An overview of denotational semantics and coalgebras

Let us give an overview of the denotational semantics of Call-By-Push-Value that justifies the introduction of the resource calculus below. This semantics is based on the semantics of Linear Logic that types the Call-By-Push-Value we are studying.

Let us describe briefly what is a model of Linear Logic (see [25] for a detailed presentation). It is given by a category \mathcal{L} together with a **symmetric monoidal** structure $(\otimes, 1, \lambda, \rho, \alpha, \sigma)$ which is **closed**¹ and we write $X \multimap Y$ for the **object of linear morphisms**. It has a **cartesian** structure with cartesian product $\&$ and terminal object \top . The category \mathcal{L} is equipped with a **comonad** $! : \mathcal{L} \rightarrow \mathcal{L}$ together with a counit $\mathbf{der}_X \in \mathcal{L}(!X, X)$ and a comultiplication $\mathbf{dig}_X \in \mathcal{L}(!X, !!X)$. This comonad comes with a symmetric monoidal structure² from $(\mathcal{L}, \&)$ to (\mathcal{L}, \otimes) , that is two natural isomorphisms $m^0 \in \mathcal{L}(1, !\top)$ and $m^2 \in \mathcal{L}(!X \otimes !Y, !(X \& Y))$.

By using isomorphisms m^0 and m^2 ; the functoriality of the comonad $!$ and the cartesian structure, we can build a structure of **comonoid** on any $!X$, which enable erasing and duplication of resources as we will see below.

$$\text{erase}_{!X} \in \mathcal{L}(!X, 1) \quad \text{split}_{!X}^2 \in \mathcal{L}(!X, !X \otimes !X)$$

A **coalgebra**³ (P, h_P) is made of an object P and a morphism $h_P \in \mathcal{L}(P, !P)$ which is compatible with the comonad structure as $\mathbf{der}_P h_P = \mathbf{Id}$ and $\mathbf{dig}_P h_P = !h_P h_P$. Every coalgebra inherits the comonoid structure of $!P$, that is it is equipped with: $\text{erase}_P \in \mathcal{L}(P, 1)$ and $\text{split}_P^2 \in \mathcal{L}(P, P \otimes P)$ defined as:

$$\text{erase}_P : P \xrightarrow{h_P} !P \xrightarrow{w_P} 1 \quad \text{split}_P^2 : P \xrightarrow{h_P} !P \xrightarrow{c_P} !P \otimes !P \xrightarrow{\mathbf{der}_P \otimes \mathbf{der}_P} P \otimes P.$$

¹ Most model we consider are also $*$ -autonomous: there is a \perp such that X is isomorphic to $(X \multimap \perp) \multimap \perp$

² The two isomorphisms m^0 and m^2 correspond to the so-called Seely isomorphisms.

³ We want the semantics we use to interpret Call-By-Push-Value to be compatible with Taylor expansion. That is why, we have chosen to resolve the comonad using the Eilenberg-Moore resolution. The resulting category can be not well-pointed as for example the relational model described below. Another option, which is simpler and should be explored, is to use the Fam resolution [1].

$$\begin{array}{ccc}
\begin{array}{c} \bar{m}_i \quad \bar{m}_Y \quad \bar{m}_Z \\ | \quad \underbrace{\hspace{2cm}} \\ (i, \bar{m}_i) \quad (\bar{m}_Y, \bar{m}_Z) \\ \underbrace{\hspace{2cm}} \\ ((i, \bar{m}_i), (\bar{m}_Y, \bar{m}_Z)) \end{array} & \xrightarrow{h_P} & \begin{array}{c} \bar{x}_i^1 \quad \bar{y}^1 \quad \bar{z}^1 \\ | \quad \underbrace{\hspace{2cm}} \\ (i, \bar{x}_i^1) \quad (\bar{y}^1, \bar{z}^1) \\ \underbrace{\hspace{2cm}} \\ ((i, \bar{x}_i^1), (\bar{y}^1, \bar{z}^1)), \dots ((i, \bar{x}_i^k), (\bar{y}^k, \bar{z}^k)) \end{array}
\end{array}$$

where $\sum_{j=1}^k \bar{x}_i^j = \bar{m}_i$, $\sum_{j=1}^k \bar{y}^j = \bar{m}_Y$, and $\sum_{j=1}^k \bar{z}^j = \bar{m}_Z$.

■ **Figure 2** Action of the coalgebra morphism h_P on a positive type.

Using similar computation, we can define $\text{split}_P^k \in \mathcal{L}(P, \underbrace{P \otimes \dots \otimes P}_k)$.

Notice that the structure of comonad of $!$ induces a coalgebras structure on $!X$. Moreover, every construction of positive type preserves the coalgebra structure. To define the coalgebraic structure of $P \otimes Q$ where P and Q are both coalgebras, let us first define the morphisms $\mu^0 \in \mathcal{L}(1, !1)$ and $\mu^2 \in \mathcal{L}(!X \otimes !Y, !(X \otimes Y))$ as

$$\begin{aligned}
\mu^0 &: 1 \xrightarrow{m^0} !\top \xrightarrow{\text{dig}_\top} !!\top \xrightarrow{!(m^0)^{-1}} !1 \\
\mu^2 &: !X \otimes !Y \xrightarrow{m^2} !(X \& Y) \xrightarrow{\text{dig}_{X \& Y}} !!(X \& Y) \xrightarrow{!(m^2)^{-1}} !(X \otimes Y) \xrightarrow{!(\text{der}_X \otimes \text{der}_Y)} !(X \otimes Y).
\end{aligned}$$

Then, we can define $h_{P \otimes Q} : P \otimes Q \xrightarrow{h_P \otimes h_Q} !P \otimes !Q \xrightarrow{\mu^2} !(P \otimes Q)$. The coalgebraic structure of the coproduct is entirely defined by the morphisms for $i \in \{1, 2\}$: $P_i \xrightarrow{h_{P_i}} !P_i \xrightarrow{! \text{in}_i} !(P_1 \oplus P_2)$ if the category has coproducts.

Thus, we can deduce that every positive type is interpreted as a coalgebra.

Example

The **relational** model is closely related to the Taylor expansion of the λ -calculus. Indeed, every λ -term is interpreted as the set of the interpretation of the resource terms that appear in its Taylor expansion. We can state that Taylor expansion is the syntactical counterpart of the relational model.

Let us describe some of these constructions on the **relational** model of linear logic. The category **Rel** is made of sets and relations. The tensor product is given by the set cartesian product and its unit is the singleton set whose unique element is denoted $*$. The product is given by disjoint union and the terminal object is the emptyset. **Rel** can be equipped with the comonad of finite multisets. The comonadic structure of $!X$ is

$$\text{der}_X = \{([a], a) \mid a \in X\} \quad \text{dig}_X = \{(\bar{m}, [\bar{m}_1, \dots, \bar{m}_k]) \mid \bar{m}_1 + \dots + \bar{m}_k = \bar{m}\}.$$

The comonoidal structure of $!X$ is

$$\text{erase}_{!X} = \{([\], *)\} \quad \text{split}_{!X}^2 = \{(\bar{m}, (\bar{m}_1, \bar{m}_2)) \mid \bar{m}_1 + \bar{m}_2 = \bar{m}\}.$$

A positive type is a finite combination of $\oplus, \otimes, !$. For instance if $P = (!X_1 \oplus !X_2) \otimes (!Y \otimes !Z)$, then P is a coalgebra (see Figure 2):

$$\begin{aligned}
h_P &= \{(((i, \bar{m}_i)), (\bar{m}_Y, \bar{m}_Z)), [((i, \bar{x}_i^1), (\bar{y}^1, \bar{z}^1)), \dots, ((i, \bar{x}_i^k), (\bar{y}^k, \bar{z}^k))]\} \\
&\quad \bar{m}_i = \bar{x}_i^1 + \dots + \bar{x}_i^k, \bar{m}_Y = \bar{y}_1 + \dots + \bar{y}_k, \bar{m}_Z = \bar{z}_1 + \dots + \bar{z}_k\},
\end{aligned}$$

16:6 Taylor expansion for Call-By-Push-Value

and is equipped with the comonoidal structure:

$$\begin{aligned} \text{erase}_P &= \{(((i, []), ([], []), *)\} \\ \text{split}_P^2 &= \{((i, \bar{m}_i), (\bar{m}_Y, \bar{m}_Z)), ((i, (\bar{m}_i^1 + \bar{m}_i^2)), ((\bar{m}_Y^1 + \bar{m}_Y^2), (\bar{m}_Z^1 + \bar{m}_Z^2))) | \\ &\quad \bar{m}_i^1 + \bar{m}_i^2 = \bar{m}_i, \bar{m}_Y^1 + \bar{m}_Y^2 = \bar{m}_Y, \bar{m}_Z^1 + \bar{m}_Z^2 = \bar{m}_Z\}. \end{aligned}$$

Remark that the structural morphisms are the same as those of $!X$ but at the leaves of the tree structure describing the formula P .

3 Resource calculus for Call-By-Push-Value

We introduce a typed resource calculus, able to simulate the operational semantics of Λ_{pv} . The conditional construction is considered through tests of equality, and there is no explicit fixpoint. The main difference with other resource calculi, like Call-By-Name or Call-By-Value, is that redexes of shape $\langle \lambda x m \rangle \bar{n}$ are not enough to entail Λ_{pv} reduction. Indeed, the notion of value is too wide to be entirely captured in multisets of approximants: $\langle \lambda x M \rangle (V_1, V_2)$ is a redex in Λ_{pv} , then we must be able to reduce terms like $\langle \lambda x m \rangle (v_1, v_2)$ in the resource setting, while keeping it sensitive to resource consumption. We proceed so with the introduction of a splitting operator, which allows us to duplicate a value using the structure of its positive type.

► **Definition 2** (Call-By-Push-Value resource calculus Δ_{pv}). *The syntax of types is the same as the syntax of Λ_{pv} .*

$$\begin{aligned} \Delta_{\text{pv}} : m ::= & x \mid 1 \mid 2 \mid \lambda x m \mid \langle m \rangle m \mid (m = m) \cdot m \mid (m, m) \mid \pi_1(m) \mid \pi_2(m) \\ & \mid [m, \dots, m] \mid \mathbf{der}(m) \end{aligned}$$

We distinguish the values of the calculus:

$$v ::= x \mid 1 \mid 2 \mid [m, \dots, m] \mid (v, v)$$

$$\begin{array}{c} \frac{}{\Gamma, x : A \vdash x : A} \quad \frac{\Gamma \vdash m_i : I, i \in \{1, \dots, k\}}{\Gamma \vdash [m_1, \dots, m_k] : !I} \quad \frac{\Gamma, x : A \vdash m : B}{\Gamma \vdash \lambda x m : A \multimap B} \\ \frac{\Gamma \vdash m : A \multimap I \quad \Delta \vdash n : A}{\Gamma, \Delta \vdash \langle m \rangle n : I} \quad \frac{\Gamma \vdash m : !A}{\Gamma \vdash \mathbf{der}(m) : A} \\ \frac{\Gamma \vdash m : A \quad \Delta \vdash n : B}{\Gamma, \Delta \vdash (m, n) : A \otimes B} \quad \frac{\Gamma \vdash m : A_1 \otimes A_2}{\Gamma \vdash \pi_i(m) : A_i} \quad i \in \{1, 2\} \quad \frac{\Gamma \vdash m : A_i}{\Gamma \vdash (i, m) : A_1 \oplus A_2} \quad i \in \{1, 2\} \\ \frac{\Gamma \vdash m_1 : A_1 \oplus A_2 \quad \Delta \vdash m_2 : A_i \quad \Theta \vdash m_3 : I}{\Gamma, \Delta, \Theta \vdash (m_1 = (i, m_2)) \cdot m_3 : I} \end{array}$$

■ **Figure 3** Typing rules for Δ_{pv} .

In order to set the operational semantics of the resource calculus just defined, we introduce a new construction \mathbf{split}^k . Its operational semantics is the duplication of ground values such as integers or variables and the split of the leaves of tree structure induced by pairs and injections, as exemplified in Figure 4. This **splitting operator** is the syntactical counterpart of the semantical morphism associated to each coalgebra P interpreting a positive type: $\text{split}_P^k \in \mathcal{L}(P, \underbrace{P \otimes \dots \otimes P}_k)$ (see Section 2.2).

$$\begin{array}{ccc}
\begin{array}{c} \bar{m} \mid \\ (i, \bar{m}) \oplus \searrow \\ ((i, \bar{m}), \bar{m}') \otimes \nearrow \\ ((i, \bar{m}), (\bar{m}', \bar{m}'')) \end{array} & & \begin{array}{c} \bar{m}_1 \mid \\ (i, \bar{m}_1) \oplus \searrow \\ ((i, \bar{m}_1), (\bar{m}'_1, \bar{m}''_1)) \end{array} & \cdots & \begin{array}{c} \bar{m}_k \mid \\ (i, \bar{m}_k) \oplus \searrow \\ ((i, \bar{m}_k), (\bar{m}'_k, \bar{m}''_k)) \end{array} \\
\begin{array}{c} \bar{m}' \mid \\ (\bar{m}', \bar{m}'') \otimes \nearrow \\ ((i, \bar{m}), (\bar{m}', \bar{m}'')) \end{array} & & \begin{array}{c} \bar{m}'_1 \mid \\ (\bar{m}'_1, \bar{m}''_1) \otimes \nearrow \\ ((i, \bar{m}_1), (\bar{m}'_1, \bar{m}''_1)) \end{array} & & \begin{array}{c} \bar{m}''_k \mid \\ (\bar{m}'_k, \bar{m}''_k) \otimes \nearrow \\ ((i, \bar{m}_k), (\bar{m}'_k, \bar{m}''_k)) \end{array}
\end{array}$$

splits into: $((i, \bar{m}_1), (\bar{m}'_1, \bar{m}''_1)), \dots, ((i, \bar{m}_k), (\bar{m}'_k, \bar{m}''_k))$

where $\sum_{i=1}^k \bar{m}_i = \bar{m}$, $\sum_{i=1}^k \bar{m}'_i = \bar{m}'$, and $\sum_{i=1}^k \bar{m}''_i = \bar{m}''$.

■ **Figure 4** Splitting a value, the tree of its positive type labelled by resource components.

► **Definition 3 (Split).** $\mathbf{split}^k(m)$ is defined as a set of k -tuples of values of same shape than m . It is defined when m is a value itself.

- $\mathbf{split}^k(\bar{m}) = \{(\bar{m}_1, \dots, \bar{m}_k) \mid \sum_{i=1}^k \bar{m}_i = \bar{m}\}$
- $\mathbf{split}^k(x) = \{(x, \dots, x)_k\}$
- $\mathbf{split}^k(i) = \{(i, \dots, i)_k\}$ for $i \in \{1, 2\}$.
- $\mathbf{split}^k((m, n)) = \{((m_1, n_1), \dots, (m_k, n_k)) \mid (m_1, \dots, m_k) \in \mathbf{split}^k(m), (n_1, \dots, n_k) \in \mathbf{split}^k(n)\}$.

We define now the reduction rules associated to Δ_{pv} , by adding the distinguished term 0 to the calculus.

- $\langle \lambda x m \rangle n \rightarrow_{\text{rpv}} m[n_1/x_1, \dots, n_k/x_k]$ for $\text{deg}_x(m) = k$ and all $(n_1, \dots, n_k) \in \mathbf{split}^k(n)$.
- $(v = (i, v')) \cdot n \rightarrow_{\text{rpv}} n$ if $v = (i, v')$. $(v = (i, v')) \cdot n \rightarrow_{\text{rpv}} 0$ otherwise.
- $\mathbf{der}([m_1, \dots, m_k]) \rightarrow_{\text{rpv}} m_1$ if $k = 1$, and $\mathbf{der}([m_1, \dots, m_k]) \rightarrow_{\text{rpv}} 0$ otherwise.
- $\pi_i((m_1, m_2)) \rightarrow_{\text{rpv}} m_i$

We define evaluation contexts e , for all terms t, u of Δ_{pv} :

$$e ::= [] \mid \langle e \rangle m \mid \langle m \rangle e \mid \lambda x e \mid (e, m) \mid (m, e) \mid (e = m) \cdot n \mid (m = e) \cdot n \mid \mathbf{der}(e)$$

and set the additional rule $e[m] \rightarrow_{\text{rpv}} e[n]$ if $m \rightarrow_{\text{rpv}} n$ by one of the above rules, with $e[0] = 0$ for all context e .

We cannot define a reduction for tests of equality that produces non values-terms, because we would lost confluence: for example, if we allow to reduce $m(\pi_1(m_1, m_2) = m_1) \cdot n$, then m reduces to 0, and it reduces as well to $(m_1 = m_1) \cdot n$, which reduces to n .

► **Proposition 4 (Subject Reduction).** For any terms m, n and general type I , if $m : I$ and $m \rightarrow_{\text{rpv}} n$, then $n : I$.

Proof. By induction on m .

- If $m = (\pi_i(m_1, m_2))$ and if $n = m_i$, then there exist A_1, A_2 such that $m_i : A_i$, and we have $m : A_i$ and $n : A_i$.
- If $m = \mathbf{der}([n])$, then there is a type J such that $n : J$, and we have $[n] : !J$ and $m : J$.
- If $m = (v_1 = (i, v_2)) \cdot n$, then if $n : J$ for some type J , then $m : J$.
- If $m = \langle \lambda x m' \rangle v$ and $n = m'[v_1/x_1, \dots, v_k/x_k]$ for $k = \text{deg}_x(m')$ and $(v_1, \dots, v_k) \in \mathbf{split}^k(v)$, then $x : A, v : A, m' : J, \lambda x m' : A \multimap J$, for some types A, J . Then $m : J$, in order to conclude $n : J$, it remains to ensure that for all $i \in \{1, \dots, k\}$, $v_i : A$ which is done easily by an induction on v , and that it implies $m'[v_1/x_1, \dots, v_k/x_k] : A$. That last point follows from a standard argument.
- If $m = e[m']$ and $n = e[n']$ for $n \rightarrow_{\text{rpv}} n'$, we conclude by induction hypothesis. ◀

We define for all $k \in \mathbf{N}$, all variable x and $m \in \Delta_{\text{pv}}$, a set of terms $\mathbf{fix}_x^k(m)$ as follows, with $\mathbf{fix}_x^0(m) = \{m[\]/x_1, \dots, \]/x_{\deg_x(m)}\}$:

$$\mathbf{fix}_x^{k+1}(m) = \{m[\bar{m}_1/x_1, \dots, \bar{m}_{\deg_x(m)}/x_{\deg_x(m)}] \mid \forall i \leq \deg_x(m) : \bar{m}_i \in (\mathbf{fix}_x^k(m))^!\}.$$

4 Taylor expansion

Taylor expansion consists in taking infinitely many approximants of a given object. As analytic maps can be understood as infinite series of polynomials that approximate it, Λ_{pv} terms can be considered through all resource terms that are also multilinear (in the computational sense) approximants. We first introduce a qualitative version, with sets, through which we show a first simulation property (Proposition 9), and we prove that the embeddings of Call-By-Name and Call-By-Value behave well at the resource level (Property 2). Then, we introduce coefficients so as to consider full quantitative Taylor expansion. Lemma 10 ensures that it does not lead to divergence issues through a finiteness property of antireduction. Finally, we prove the full simulation of Λ_{pv} reduction in Taylor expansion, showing that coefficients commute with reduction, in Theorem 17.

4.1 Definition and Simulation

► **Definition 5** (Support of Taylor expansion). *We define the sets of resource terms corresponding to the support of Taylor expansion of Λ_{pv} :*

$$\begin{array}{ll} \mathcal{T}_{\text{pv}}(x) = \{x\} & \mathcal{T}_{\text{pv}}\langle M \rangle N = \{\langle m \rangle n \mid m \in \mathcal{T}_{\text{pv}}(M), n \in \mathcal{T}_{\text{pv}}(N)\} \\ \mathcal{T}_{\text{pv}}(\iota_i(M)) = \{(i, m) \mid m \in \mathcal{T}_{\text{pv}}(M)\} & \mathcal{T}_{\text{pv}}(\mathbf{der}(M)) = \{\mathbf{der}(m) \mid m \in \mathcal{T}_{\text{pv}}(M)\} \\ \mathcal{T}_{\text{pv}}(M^!) = \mathcal{T}_{\text{pv}}(M)^! & \mathcal{T}_{\text{pv}}(\langle M, N \rangle) = \{\langle m, n \rangle \mid m \in \mathcal{T}_{\text{pv}}(M), n \in \mathcal{T}_{\text{pv}}(N)\} \\ \mathcal{T}_{\text{pv}}(\pi_i(M)) = \{\pi_i(m) \mid m \in \mathcal{T}_{\text{pv}}(M)\} & \mathcal{T}_{\text{pv}}(\mathbf{fix}_x^k(M)) = \{\mathbf{fix}_x^k(m) \mid m \in \mathcal{T}_{\text{pv}}(M), k \in \mathbf{N}\} \\ \mathcal{T}_{\text{pv}}(\lambda x M) = \{\lambda x m \mid m \in \mathcal{T}_{\text{pv}}(M)\} & \mathcal{T}_{\text{pv}}(\mathbf{case}(M, z_1 \cdot N_1, z_2 \cdot N_2)) = \{(m = (i, m') \cdot n_i[m'/z_i] \mid i \in \{1, 2\}, m \in \mathcal{T}_{\text{pv}}(M), n_i \in \mathcal{T}_{\text{pv}}(N_i), m' \in \Delta_{\text{pv}})\} \end{array}$$

► **Property 1.** *Let $M \in \Lambda_{\text{pv}}$, $m \in \mathcal{T}_{\text{pv}}(M)$, and $k \in \mathbf{N}$. $\mathbf{split}^k(m)$ is defined if and only if M is a value.*

Proof. One can check that the syntax of resource terms v that are in $\mathcal{T}_{\text{pv}}(V)$ for a value V matches exactly the resource values of Definition 2. It is easy to verify that $\mathbf{split}^k(v)$ is always defined, and that if $m \in \mathcal{T}_{\text{pv}}(M)$ is not such a resource value, then $\mathbf{split}^k(m)$ is not defined. ◀

The following corollary shows that Δ_{pv} is consistent with Λ_{pv} in the following sense: an approximant of a redex in Λ_{pv} is always a redex in Δ_{pv} , and a redex in Δ_{pv} which is an approximant of a term in Λ_{pv} , is the approximation of a redex. This is mostly trivial, but for redexes of shape $\langle \lambda x m \rangle n$ (respectively $\langle \lambda x M \rangle N$), where it is a consequence of Property 1, as stated in the following corollary:

► **Corollary 6.** *Let $\langle \lambda x m \rangle n \in \mathcal{T}_{\text{pv}}(\langle \lambda x M \rangle N)$. There is a term m' such that $\langle \lambda x m \rangle n \rightarrow_{\text{rpv}} m'$ by reducing the most external redex if and only if N is a value. Recall moreover that $\langle \lambda x M \rangle N \rightarrow_{\text{pv}} M[N/x]$ if and only if N is a value.*

► **Lemma 7.** *If M is a value, $k \in \mathbf{N}$, $m \in \mathcal{T}_{\text{pv}}(M)$ and $(m_1, \dots, m_k) \in \mathbf{split}^k(m)$ then for all $i \in \{1, \dots, k\}$, $m_i \in \mathcal{T}_{\text{pv}}(M)$.*

Proof. By induction on M , using Property 1 :

- If $M = x$, then $m = x$ and $\mathbf{split}^k(m) = (x, \dots, x)_k$. We conclude since $\mathcal{T}_{\text{pv}}(x) = \{x\}$.
- If $M = N^!$, then $m = [n_1, \dots, n_l]$, and for all $i \in \{1, \dots, l\}$, $n_i \in \mathcal{T}_{\text{pv}}(N)$. We have $(m_1, \dots, m_k) = (\bar{n}_1, \dots, \bar{n}_k)$ with $\sum_{i=1}^k \bar{n}_i = [n_1, \dots, n_l]$. Then, each \bar{n}_i is a multiset of elements in $\mathcal{T}_{\text{pv}}(N)$, and $\bar{n}_i \in \mathcal{T}_{\text{pv}}(N^!) = \mathcal{T}_{\text{pv}}(M)$.
- If $M = (N, N')$, then $m = (n, n')$ for $n \in \mathcal{T}_{\text{pv}}(N)$ and $n' \in \mathcal{T}_{\text{pv}}(N')$. $(m_1, \dots, m_k) = ((n_1, n'_1), \dots, (n_k, n'_k))$ with $(n_1, \dots, n_k) \in \mathbf{split}^k(N)$ and $(n'_1, \dots, n'_k) \in \mathbf{split}^k(N')$. By induction hypothesis, for all $i \in \{1, \dots, k\}$, $n_i \in \mathcal{T}_{\text{pv}}(N)$ and $n'_i \in \mathcal{T}_{\text{pv}}(N')$. Then for all i , $(n_i, n'_i) \in \mathcal{T}_{\text{pv}}(N, N') = \mathcal{T}_{\text{pv}}(M)$.
- If $M = \iota_j(N)$, then $m = (j, n)$ for $n \in \mathcal{T}_{\text{pv}}(N)$ and $\mathbf{split}^k(m) = ((j, n_1), \dots, (j, n_k))$ with $(n_1, \dots, n_k) \in \mathbf{split}^k(n)$. By induction hypothesis, for all $i \in \{1, \dots, k\}$, $n_i \in \mathcal{T}_{\text{pv}}(N)$. Then for all i , $(j, n_i) \in \mathcal{T}_{\text{pv}}(\iota_j(N)) = \mathcal{T}_{\text{pv}}(M)$. ◀

The following substitution lemma is crucial to ensure that Taylor expansion is compatible with reduction. It will be used for proving simulation, in Proposition 9.

► **Lemma 8** (Substitution). *Let $m \in \mathcal{T}_{\text{pv}}(M)$, $k = \text{deg}_x(m)$, and $n_1, \dots, n_k \in \mathcal{T}_{\text{pv}}(N)$, for $M, N \in \Delta_{\text{pv}}$. We have $m[n_1/x_1, \dots, n_k/x_k] \in \mathcal{T}_{\text{pv}}(M[N/x])$.*

Proof. The proof is by induction on M . We only consider representative cases, the other following by similar applications of induction hypothesis.

- If $M = x$, then $m = x, k = 1, m[n_1/x_1] = n_1$, and $M[N/x] = N$. Then $m[n_1/x_1] \in \mathcal{T}_{\text{pv}}(M[N/x])$.
- If $M = \lambda y M'$, then $\text{deg}_x(M) = \text{deg}_x(M'), m = \lambda y m'$ for $m' \in \mathcal{T}_{\text{pv}}(M')$. By induction hypothesis, $m'[n_1/x_1, \dots, n_k/x_k] \in \mathcal{T}_{\text{pv}}(M'[N/x])$. Since $m[n_1/x_1, \dots, n_k/x_k] = \lambda y m'[n_1/x_1, \dots, n_k/x_k]$, we conclude.
- If $M = \langle M_1 \rangle M_2$, then $m = \langle m_1 \rangle m_2$ for $m_i \in \mathcal{T}_{\text{pv}}(M_i)$, and $\text{deg}_x(m) = l_1 + l_2$ for $l_1 = \text{deg}_x(m_1)$ and $l_2 = \text{deg}_x(m_2)$. By induction hypothesis, $m_1[n_1/x_1, \dots, n_{l_1}/x_{l_1}] \in \mathcal{T}_{\text{pv}}(M_1[N/x])$ and $m_2[n_{l_1+1}/x, \dots, n_{l_1+l_2}/x] \in \mathcal{T}_{\text{pv}}(M_2[N/x])$. Since $m[n_1/x_1, \dots, n_k/x_k] = \langle m_1[n_1/x_1, \dots, n_{l_1}/x_{l_1}] \rangle m_2[n_{l_1+1}/x, \dots, n_{l_1+l_2}/x]$, and $M[N/x] = \langle M_1[N/x] \rangle M_2[N/x]$, we conclude.
- If $M = M^!$, then $m = [m'_1, \dots, m'_l]$ with $m'_i \in \mathcal{T}_{\text{pv}}(M')$ for all i , and $\text{deg}_x(m) = \sum_{i=1}^l k_i$ where $k_i = \text{deg}_x(m'_i)$. By induction hypothesis, $m'_i[n_{k_{i-1}+1}/x_{k_{i-1}+1}, \dots, n_{k_{i-1}+k_i}/x_{k_{i-1}+k_i}] \in \mathcal{T}_{\text{pv}}(M'[N/x])$ for all $i \in \{1, \dots, l\}$ (setting $k_0 = 0$). Then, $M[N/x] = (M'[N/x])^!$, and we can conclude as before.
- In $M = \mathbf{case}(M', z_1 \cdot N_1, z_2 \cdot N_2)$, then $m = (m' = (i, m'')) \cdot n_i[m''/z_i]$ for $i \in \{1, 2\}, m' \in \mathcal{T}_{\text{pv}}(M'), n_i \in \mathcal{T}_{\text{pv}}(N_i), m'' \in \Delta_{\text{pv}}$. We conclude by induction hypothesis as above. ◀

Notice that only the case where N is a value will be used, since the other cases do not appear in the operational semantics.

We can finally prove the first simulation property:

► **Proposition 9** (Simulation). *If $M \rightarrow_{\text{pv}} M'$, then for any $m \in \mathcal{T}_{\text{pv}}(M)$, either $m \rightarrow_{\text{rpv}} 0$ or there is $m' \in \mathcal{T}_{\text{pv}}(M')$ such that $m \rightarrow_{\text{rpv}}^{\bar{\bar{}}} m'$, where $\rightarrow_{\text{rpv}}^{\bar{\bar{}}}$ is the reflexive closure of \rightarrow_{rpv} .*

Proof. By induction on M :

- If $M = \pi_i((M_1, M_2))$ and $M' = M_i$, then $m = \pi_i((m_1, m_2))$ for $m_i \in \mathcal{T}_{\text{pv}}(M_i)$. We conclude since $M \rightarrow_{\text{pv}} M_i$ and $m \rightarrow_{\text{rpv}} m_i$.
- If $M = \mathbf{der}(N^!)$ and $M' = N$, then $m = \mathbf{der}([n_1, \dots, n_k])$, with $n_i \in \mathcal{T}_{\text{pv}}(N)$ for all $i \in \{1, \dots, k\}$. We conclude since $M \rightarrow_{\text{pv}} N$ and $m \rightarrow_{\text{rpv}} n_1$ if $k = 1$ and $m \rightarrow_{\text{rpv}} 0$ otherwise.

16:10 Taylor expansion for Call-By-Push-Value

- If $M = \mathbf{fix}_x(N)$ and $M' = N[(\mathbf{fix}_x(N))^\dagger/x]$, then it is easy to verify that $\mathcal{T}_{\text{pv}}(M) = \mathcal{T}_{\text{pv}}(M')$, using Lemma 8 and unfolding the definition of Taylor expansion of fixpoint. We need a reflexive reduction for this case.
- If $M = (\lambda y.N)V$ and $M' = N[V/y]$, then $m = \langle \lambda y n \rangle v$ for $n \in \mathcal{T}_{\text{pv}}(N)$ and $v \in \mathcal{T}_{\text{pv}}(V)$. By Property 1, $\mathbf{split}^k(v)$ is defined for any $k \in \mathbf{N}$, then $m \rightarrow_{\text{rpv}} n[v_1/y_{f(1)}, \dots, v_k/y_{f(k)}]$ for $\text{deg}_y(n) = k$ and $(v_1, \dots, v_k) \in \mathbf{split}^k(v)$. By Lemma 7, for all $i \in \{1, \dots, k\}$, $v_i \in \mathcal{T}_{\text{pv}}(V)$, and by the substitution Lemma 8, $n[v_1/y_1, \dots, v_k/y_k] \in \mathcal{T}_{\text{pv}}(N[V/y])$.
- If $M = \mathbf{case}(\iota_i(V), x_1 \cdot M_1, x_2 \cdot M_2)$ and $M' = M_i[V/x_i]$, then, $m = ((i, v) = (j, n)) \cdot m_i[v/x_i]$ for $i, j \in \{1, 2\}$, $v \in \mathcal{T}_{\text{pv}}(V)$, $n \in \Delta_{\text{pv}}$, $m_i \in \mathcal{T}_{\text{pv}}(M_i)$. Either $m \rightarrow_{\text{rpv}} 0$, either $(i, v) = (j, n)$ and in this case $m \rightarrow_{\text{rpv}} m_i[n/x_i] = m_i[v/x_i]$. By the substitution Lemma 8 we conclude, since we have $M \rightarrow_{\text{pv}} M_i[V/x_i]$ and $m_i[v/x_i] \in \mathcal{T}_{\text{pv}}(M_i[V/x_i])$.
- If $M = E[N]$ and $M' = E[N']$, then we can easily show that there is a resource context e such that $m = e[n]$ and $n \in \mathcal{T}_{\text{pv}}(N)$. By induction hypothesis, either $n \rightarrow_{\text{rpv}} 0$, and then $e[n] = 0$, or there exists n' such that $n \rightarrow_{\text{rpv}} n'$ and $n' \in \mathcal{T}_{\text{pv}}(N')$. We can easily adapt the substitution Lemma to conclude $e[n'] \in \mathcal{T}_{\text{pv}}(E[N'])$. ◀

4.2 Embeddings of CBV and CBN

Call-By-Push-Value is known to subsume both Call-By-Name and Call-By-Value strategies. In particular, the two strategies can be embedded into Λ_{pv} . If we consider simply typed λ -calculus⁴ Λ , we set two functions $(\)^v, (\)^n : \Lambda \rightarrow \Lambda_{\text{pv}}$, defined in Table 5. We do not consider here calculi with products, or other constructors, in order to focus in a simple setting on the relation between exponentials and strategies of reduction (see Ehrhard and Tasson's work [14] for more developments). Our embeddings ensure *e.g.* the following property: $((\lambda x M)N)^v \rightarrow_{\text{pv}} (M[N/x])^v$ if and only if N is a variable or an abstraction, and $((\lambda x M)N)^n \rightarrow_{\text{pv}} (M[N/x])^n$ for any M, N .

From the Taylor expansion point of view, let \mathcal{T}^n and \mathcal{T}^v be, respectively, usual Call-By-Name expansion, and Call-By-Value expansion (first defined by Ehrhard [11]). We can check the correctness of our construction of Δ_{pv} and \mathcal{T}_{pv} with respect to those embeddings, using \mathcal{T}^n and \mathcal{T}^v defined in Table 2. The first one is defined on Δ^n , which is the original Ehrhard and Regnier's resource calculus [9], and the second one on Δ^v , a Call-By-Value resource calculus, introduced by Ehrhard [11]. Both are described in Table 1.

■ **Table 1** Call-By-Name and Call-By-Value resource calculi.

Δ^n	Δ^v
$m, n ::= x \mid \lambda x m \mid \langle m \rangle \bar{n}$	$m, n ::= [x_1, \dots, x_k] \mid [\lambda x m_1, \dots, \lambda x m_k] \mid \langle m \rangle n$
$\langle \lambda x m \rangle [n_1, \dots, n_k] \rightarrow m[n_1/x_{f(1)}, \dots, n_k/x_{f(k)}]$ if $k = \text{deg}_x(m)$ and $f \in \mathfrak{S}_k$	$\langle [\lambda x m] \rangle [n_1, \dots, n_k] \rightarrow m[n_1/x_{f(1)}, \dots, n_k/x_{f(k)}]$ if $k = \text{deg}_x(m)$ and $f \in \mathfrak{S}_k$

► **Property 2.** For any pure λ -term $M \in \Lambda$, $E(\mathcal{T}_{\text{pv}}((M)^v)) = \mathcal{T}^v(M)$ and $E(\mathcal{T}_{\text{pv}}((M)^n)) = \mathcal{T}^n(M)$, where E is the function that erases all the dereflections (that do not exist in Δ^n nor in Δ^v) in a set of terms.

⁴ We do not make types explicit, since the translation works in the same way with pure λ -calculus (*e.g.* when translated in Linear Logic proof nets). But since the target calculus is typed, this restriction is necessary

■ **Table 2** $\mathcal{T}^v : \Lambda \rightarrow P(\Delta^v)$ and $\mathcal{T}^n : \Lambda \rightarrow P(\Delta^n)$.

Call-By-Name Taylor expansion	Call-By-Value Taylor expansion
$\mathcal{T}^n(x) = \{x\}$	$\mathcal{T}^v(x) = \{x\}^\dagger$
$\mathcal{T}^n(MN) = \{\langle m \rangle \bar{n} \mid m \in \mathcal{T}^n(M), \bar{n} \in \mathcal{T}^n(N)^\dagger\}$	$\mathcal{T}^v(MN) = \{\langle m \rangle n \mid m \in \mathcal{T}^v(M), n \in \mathcal{T}^v(N)\}$
$\mathcal{T}^n(\lambda x M) = \{\lambda x m \mid m \in \mathcal{T}^n(M)\}$	$\mathcal{T}^v(\lambda x M) = \{\lambda x m_1, \dots, \lambda x m_k \mid m_i \in \mathcal{T}^v(M)\}$

■ **Figure 5** Both translations are functions from Λ to Λ_{pv} .

Call-By-Name translation	Call-By-Value translation
$(x)^n = \mathbf{der}(x)$	$(x)^v = \mathbf{der}(x)^\dagger$
$(MN)^n = \langle M^n \rangle (N^n)^\dagger$	$(MN)^v = \langle \mathbf{der}(M) \rangle N$
$(\lambda x M)^n = \lambda x M^n$	$(\lambda x M)^v = (\lambda x M^v)^\dagger$

Proof. The proof consists in a simple examination of the definitions. Let us start with Call-By-Value constructions: The variable case is immediate since $\mathcal{T}_{\text{pv}}(x^v) = \{\mathbf{der}(x)\}^\dagger$, and $\mathcal{T}^v(x) = \{x\}^\dagger$. $\mathcal{T}_{\text{pv}}((\lambda x M)^v) = \{\lambda x m_1, \dots, \lambda x m_k \mid k \in \mathbf{N}, m_i \in \mathcal{T}_{\text{pv}}(M^v)\}$, we conclude since by induction hypothesis, $E(\mathcal{T}_{\text{pv}}(M^v)) = \mathcal{T}^v(M)$ and $\mathcal{T}^v(\lambda x M) = \{\lambda x m'_1, \dots, \lambda x m'_l \mid l \in \mathbf{N}, m'_i \in \mathcal{T}^v(M)\}$. The application case is managed with a similar argument with induction hypothesis, and with the fact that $E(\langle \mathbf{der}(M) \rangle N) = \langle E(M) \rangle E(N)$.

For Call-By-Name, we only consider the application case (the other being straightforward): $\mathcal{T}_{\text{pv}}((MN)^n) = \{\langle m \rangle \bar{n} \mid m \in \mathcal{T}_{\text{pv}}(M^n), \bar{n} \in \mathcal{T}_{\text{pv}}(N^n)^\dagger\}$. By induction hypothesis, $E(\mathcal{T}_{\text{pv}}(M^n)) = \mathcal{T}^n(M)$ and $E(\mathcal{T}_{\text{pv}}(N^n)) = \mathcal{T}^n(N)$, and we can conclude. ◀

Together with the simulation property of \mathcal{T}_{pv} (Property 9), Property 2 proves that Call-By-Push-Value subsumes both Call-By-Name and Call-By-Value strategies, and that remains valid at a resource level.

4.3 Finiteness

The following lemma ensures that one can consider a quantitative version of Taylor expansion \mathcal{T}_{pv} , and extend the resource reduction to an infinite and weighted setting. The conditions of validity of this result have been widely studied in non uniform settings, Linear-Logic proof nets, or various strategies of reduction [2, 3, 26, 27]. This is necessary for proving Lemma 15 that state that coefficients remain finite under reduction.

► **Lemma 10** (Finiteness of antireduction). *Let $n \in \Delta_{\text{pv}}$ and M in Λ_{pv} . $\{m \in \mathcal{T}_{\text{pv}}(M) \mid m \rightarrow_{\text{rpv}}^\# n\}$ is finite.*

(sketch). We do not detail the proof, since we can adapt the first author's work [2] for PCF. The idea is to extend Ehrhard and Regnier's original proof [10], defining a coherence relation on resource terms in a way $\mathcal{T}_{\text{pv}}(M)$ is always a maximal clique for this relation. In particular, $\bigcup_{k \in \mathbf{N}} \mathbf{fix}_x^k(m)$ must be a clique.

Then, it remains to show that the reduction preserves coherence, and that if m, m' are coherent, and both reduce to n , then $m = m'$. We conclude that there cannot be several distinct resource terms in $\mathcal{T}_{\text{pv}}(M)$ reducing to a common term. ◀

4.4 Taylor expansion with coefficients

In the remainder of this section, we will consider infinite linear combinations of resource terms. Those terms will take coefficients in an arbitrary commutative semiring \mathbf{S} with fractions: a semiring in which every natural number $k \neq 0 \in \mathbf{N}$ admits a multiplicative inverse, written

16:12 Taylor expansion for Call-By-Push-Value

$\frac{1}{k}$. For a combination $\varphi = \sum_{i \in I} a_i \cdot m_i \in \mathbf{S}^{\Delta_{\text{pv}}}$, and for a resource term $m \in \Delta_{\text{pv}}$, we denote by $(\varphi)_m$ the coefficient of m in φ , that correspond to $\prod_{m_i=m} a_i$.

All the constructors of Δ_{pv} are linear, in the sense that we can write e.g. $\lambda x (\sum_{i \in I} a_i \cdot m_i) = \sum_{i \in I} a_i \cdot \lambda x m_i$, (see Introduction for those notations). This allows us to give the definition of full Taylor expansion with coefficients as follows:

► **Definition 11** (Full Taylor expansion). *Let \mathbf{S} be any commutative semiring with fractions. We define quantitative Taylor expansion, which is a function $(\cdot)^* : \Delta_{\text{pv}} \rightarrow \mathbf{S}^{\Delta_{\text{pv}}}$, and consists in linear combinations of elements in \mathcal{T}_{pv} .*

- $x^* = x$.
- $(\lambda x M)^* = \lambda x M^*$
- $(\langle M \rangle N)^* = \langle M^* \rangle N^*$
- $((M, N))^* = (M^*, N^*)$
- $(\iota_i(M))^* = (i, M^*)$
- $(\pi_i(M))^* = \pi_i((^*M))$
- $\text{case}(M, x_1 \cdot N_1, x_2 \cdot N_2)^* = \sum_{i \in \{1,2\}} \sum_{r \in \Delta_{\text{pv}}} ((M^*) = (i, r)) \cdot (N_i[M/x_i])^*$
- $(M^!)^* = \sum_{k \in \mathbf{N}} \frac{1}{k!} [M^*, \dots, M^*]_k$
- $(\text{der}(M))^* = \text{der}(M^*)$

Taylor expansion of fixpoints is defined inductively. We set a combination $\mathbf{fix}_x(M)^{*k}$ for all $k \in \mathbf{N}$, which corresponds to k unfoldings of M in x , as a quantitative version of the sets $\mathbf{fix}_x^k(m)$ of Definition 5.

- $(\mathbf{fix}_x(M))^{*0} = (M[\square/x])^*$

$$(\mathbf{fix}_x(M))^{*k+1} = \sum_{m \in \mathcal{T}_{\text{pv}}(M)} \sum_{\vec{m} \in (\mathbf{fix}_x^k(M))^!} (M^*)_m \prod_{i=1}^{\text{deg}_x(m)} ((\mathbf{fix}_x(M))^{*k})_{\vec{m}_i}^! \cdot m[\vec{m}_1/x_1, \dots, \vec{m}_{\text{deg}_x(m)}/x_{\text{deg}_x(m)}]$$

and we set $(\mathbf{fix}_x(M))^* = \sum_{k \in \mathbf{N}} (\mathbf{fix}_x(M))^{*k}$.

We also need to give a quantitative version of the splitting operator, in order to make one step-reduction commute with quantitative Taylor expansion defined above.

► **Definition 12** (Quantitative split). *We define for all $k \in \mathbf{N}$ and all resource value v the weighted finite sum $\mathbf{split}_+^k(v)$ as follows : if $v \in \{1, 2\}$ or $v = x$, then $\mathbf{split}_+^k(v) = (v, \dots, v)_k$.*

If $v = \vec{m}$, then $\mathbf{split}_+^k(v) = \sum_{\vec{m}_1 + \dots + \vec{m}_k = \vec{m}} \frac{|\vec{m}|!}{|\vec{m}_1|! \dots |\vec{m}_k|!} \cdot (\vec{m}_1, \dots, \vec{m}_k)$. If $v = (v_1, v_2)$, then

$\mathbf{split}_+^k(v)$ is defined as following, setting $\vec{v}_i = (v_{i,1}, \dots, v_{i,k})$:

$$\sum_{\substack{(v_{1,1}, \dots, v_{1,k}), (v_{2,1}, \dots, v_{2,k}) \\ \in |\mathbf{split}_+^k(v_1)| \in |\mathbf{split}_+^k(v_2)|}} \left(\mathbf{split}_+^k(v_1) \right)_{\vec{v}_1} \left(\mathbf{split}_+^k(v_2) \right)_{\vec{v}_2} \cdot ((v_{1,1}, v_{2,1}), \dots, (v_{1,k}, v_{2,k}))$$

We now introduce a reduction rule that takes into account the coefficients of definition 12.

► **Definition 13** (Quantitative resource reduction $\rightarrow_{\text{rpv}+}$). *Let $m \in \Delta_{\text{pv}}$ and $k = \text{deg}_x(m)$.*

$$\langle \lambda x m \rangle v \rightarrow_{\text{rpv}+} \sum_{(v_1, \dots, v_k) \in \Delta_{\text{pv}}^k} \left(\mathbf{split}_+^k(v) \right)_{(v_1, \dots, v_k)} m[v_1/x_1, \dots, v_k/x_k]$$

If $m \rightarrow_{\text{rpv}} n$ by reducing a redex of another shape than $\langle \lambda x m \rangle n$, then we also set $m \rightarrow_{\text{rpv}+} n$.

Notice that if $m \rightarrow_{\text{rpv}+} \sum_{i=1}^k a_i \cdot n_i$, then for all $i \in \{1, \dots, k\}$ such that $a_i \neq 0$, we have $m \rightarrow_{\text{rpv}} n_i$.

► **Definition 14** (Reduction between combinations). We define a reduction $\Rightarrow_{\subseteq} \mathbf{S}^{\Delta_{\text{pv}}} \times \mathbf{S}^{\Delta_{\text{pv}}}$. Given a family of resource terms $(m_i)_{i \in I}$ and a family of finite sums of resources terms $(\nu_i)_{i \in I}$ such that for all $i \in I$, and for all $n \in |\nu_i|$ the set $\{j \in I \mid m_j \rightarrow_{\text{rpv}^+}^{\bar{\bar{}}} n\}$ is finite.

In that case, we set $\sum_{i \in I} a_i \cdot m_i \Rightarrow \sum_{i \in I} a_i \cdot n_i$ as soon as $m_i \rightarrow_{\text{rpv}}^{\bar{\bar{}}} n_i$ for all $i \in I$.

► **Lemma 15.** Let $M \in \Lambda_{\text{pv}}$ with $M^* = \sum_{i \in I} a_i \cdot m_i$ and $\varphi = \sum_{i \in I} a_i \cdot \nu_i$ such that $m_i \rightarrow_{\text{rpv}^+}^{\bar{\bar{}}} \nu_i$ for all $i \in I$. Then, for all $i \in I$ and for all $n \in |\nu_i|$, n has a finite coefficient in φ .

In other words, the reduction \Rightarrow is always defined on Taylor expansion.

Proof. This is an immediate consequence of Lemma 10 and Definition 13. ◀

► **Lemma 16.** Let $m \in \Delta_{\text{pv}}$, with $\deg_x(m) = k$, and V a value of Λ_{pv} .

$$\begin{aligned} & \sum_{v \in \mathcal{T}_{\text{pv}}(V)} \sum_{\substack{(v_1, \dots, v_k) \\ \in \mathbf{split}^k(v)}} (V^*)_v \left(\mathbf{split}_+^k(v) \right)_{(v_1, \dots, v_k)} \cdot m[v_1/x_1, \dots, v_k/x_k] \\ &= \sum_{\substack{(v_1, \dots, v_k) \\ \in \mathcal{T}_{\text{pv}}(V)^k}} \prod_{i=1}^k (V^*)_{v_i} \cdot m[v_1/x_1, \dots, v_k/x_k] \end{aligned}$$

Proof. The proof is by induction on V .

- If V is a variable, then all the coefficients $(V^*)_{v_i}$ are equal to 1, and the result is trivial.
- If $V = N^!$, then we want to establish the following, for any $k \in \mathbf{N}$:

$$\begin{aligned} & \sum_{\bar{n} \in \mathcal{T}_{\text{pv}}(N)^!} \sum_{\bar{n} \in \mathbf{split}^k(\bar{n})} \left(\mathbf{split}_+^k(\bar{n}) \right)_{(\bar{n}_1, \dots, \bar{n}_k)} \prod_{i=1}^{|\bar{n}|} (N^*)_{n_i} \frac{1}{|\bar{n}|!} \cdot m[\bar{n}_1/x_1, \dots, \bar{n}_k/x_k] \\ &= \sum_{\substack{(\bar{n}_1, \dots, \bar{n}_k) \\ \in \mathcal{T}_{\text{pv}}(N^!)^k}} \frac{1}{|\bar{n}_1|! \dots |\bar{n}_k|!} \prod_{i=1}^k \prod_{j=1}^{|\bar{n}_i|} (N^*)_{n_{i,j}} \cdot m[\bar{n}_1/x_1, \dots, \bar{n}_k/x_k] \end{aligned}$$

Where for all $i \leq k$, $\bar{n}_i = [n_{i,1}, \dots, n_{i,|\bar{n}_i|}]$.

This equation is verified by looking at the definition of \mathbf{split}_+^k . $\left(\mathbf{split}_+^k(\bar{n}) \right)_{(\bar{n}_1, \dots, \bar{n}_k)}$ is equal to $\frac{|\bar{n}|!}{|\bar{n}_1|! \dots |\bar{n}_k|!}$, which is enough to simplify the above equation and conclude this case.

- If $V = (V_1, V_2)$. Then we want to establish:

$$\begin{aligned} & \sum_{\substack{(v_1, v_2) \\ \in \mathcal{T}_{\text{pv}}((V_1, V_2))}} \sum_{\substack{(u_1, \dots, u_k) \\ \in \mathbf{split}^k((v_1, v_2))}} (V_1, V_2)^*_{(v_1, v_2)} \left(\mathbf{split}_+^k((v_1, v_2)) \right)_{(u_1, \dots, u_k)} \cdot m[u_1/x_1, \dots, u_k/x_k] \\ &= \sum_{\substack{(u_1, \dots, u_k) \\ \in \mathcal{T}_{\text{pv}}((V_1, V_2))^k}} \prod_{i=1}^k (V_1^*)_{v_{1,i}} \prod_{j=1}^k (V_2^*)_{v_{2,j}} \cdot m[u_1/x_1, \dots, u_k/x_k] \end{aligned}$$

Where $(u_1, \dots, u_k) = ((v_{1,1}, v_{2,1}), \dots, (v_{1,k}, v_{2,k}))$, for $(v_{i,1}, \dots, v_{i,k}) \in \mathbf{split}^k(v_i)$.

16:14 Taylor expansion for Call-By-Push-Value

By induction hypothesis, we have for $i \in \{1, 2\}$:

$$\begin{aligned} & \sum_{v_i \in \mathcal{T}_{\text{pv}}(V_i)} \sum_{\substack{(v_{i,1}, \dots, v_{i,k}) \\ \in \mathbf{split}^k(v_i)}} (V_i^*)_{v_i} \left(\mathbf{split}_+^k(v_i) \right)_{(v_{i,1}, \dots, v_{i,k})} \cdot m[v_{i,1}/x_1, \dots, v_{i,k}/x_k] \\ &= \sum_{\substack{(v_{i,1}, \dots, v_{i,k}) \\ \in \mathcal{T}_{\text{pv}}(V_i)^k}} \prod_{j=1}^k (V_i^*)_{v_{i,j}} \cdot m[v_{i,1}/x_1, \dots, v_{i,k}/x_k] \end{aligned}$$

Which allows us to conclude this case since $((V_1, V_2)^*)_{(v_{1,i}, v_{2,j})} = (V_1^*)_{v_{1,i}} \times (V_2^*)_{v_{2,j}}$ and $\left(\mathbf{split}_+^k((v_1, v_2)) \right)_{(u_1, \dots, u_k)} = \prod_{i=1}^2 \left(\mathbf{split}_+^k(v_i) \right)_{(v_{i,1}, \dots, v_{i,k})}$

■ The case $V = \iota_i(V')$ is proved in the same way by induction hypothesis. ◀

► **Property 3.** $(M[N/x])^* =$

$$\sum_{m \in \mathcal{T}_{\text{pv}}(M)} \sum_{(n_1, \dots, n_k) \in \mathcal{T}_{\text{pv}}(N)^k} (M^*)_m \prod_{i=1}^k (N^*)_{n_i} \cdot m[n_1/x_1, \dots, n_k/x_k]$$

where $k = \text{deg}_x(m)$.

Proof. Easy induction on M . ◀

We can finally state the main result of this section and of the paper: Theorem 17 establishes the simulation of Λ_{pv} operational semantics in Taylor expansion with coefficients.

► **Theorem 17.** *Let $M, M' \in \Lambda_{\text{pv}}$, if $M \rightarrow_{\text{pv}} M'$, then $M^* \Rightarrow M'^*$.*

Proof. We use Proposition 9, and verify that it extends to full Taylor expansion, keeping all coefficients in the right place.

■ If $M = \langle \lambda x N \rangle V$ and $M' = N[V/x]$, then $M^* =$

$$\begin{aligned} & \sum_{n \in \mathcal{T}_{\text{pv}}(N)} \sum_{v \in \mathcal{T}_{\text{pv}}(V)} (N^*)_n (V^*)_v \cdot \langle \lambda x n \rangle v \\ & \Rightarrow \sum_{n \in \mathcal{T}_{\text{pv}}(N)} \sum_{\substack{v \in \mathcal{T}_{\text{pv}}(V) \\ (v_1, \dots, v_k) \\ \in \mathbf{split}^k(v)}} \sum_{(v_1, \dots, v_k)} (N^*)_n (V^*)_v \left(\mathbf{split}_+^k(v) \right)_{(v_1, \dots, v_k)} \cdot n[v_1/x_1, \dots, v_k/x_k] \\ &= \sum_{n \in \mathcal{T}_{\text{pv}}(N)} \sum_{(v_1, \dots, v_k) \in \mathcal{T}_{\text{pv}}(V)^k} (N^*)_n \prod_{i=1}^k (V^*)_{v_i} \cdot n[v_1/x_1, \dots, v_k/x_k] \end{aligned}$$

The last equality is obtained by Lemma 16, and is equal to $N[V/x]^*$ by Property 3.

■ If $M = \mathbf{case}(\iota_i(V), x_1 \cdot M_1, x_2 \cdot M_2)$ and $M' = M_i[V/x_i]$, then $M^* =$

$$\begin{aligned} & \sum_{j \in \{1, 2\}} \sum_{r \in \Delta_{\text{pv}}} ((i, V)^* = (j, r)) \cdot N_j^*[V^*/x_{j,1}, \dots, V^*/x_{j,k}] \\ & \Rightarrow N_i^*[V^*/x_{i,1}, \dots, V^*/x_{i,k}] \end{aligned}$$

Which is equal to $(N[V/x])^*$ by Property 3.

■ If $M = \mathbf{der}(N^!)$ and $M' = N$, then we verify immediately $(\mathbf{der}(N^!))^* = \mathbf{der}((N^!)^*) = \mathbf{der}((N^*)^!) = N^*$, since $\mathbf{der}([n_1, \dots, n_k]) \rightarrow_{\text{rpv}} 0$ if $k \neq 1$.

■ If $M = \mathbf{fix}_x(N)$, then, $M^* = (M[(\mathbf{fix}_x M)^! / x])^*$. Property 3 and an examination of the definition of Taylor expansion of fixpoint is sufficient to verify this point.

■ The projections rules are obtained by a straightforward application of the definitions. ◀

5 Conclusions

We have introduced a new resource calculus reflecting Call-By-Push-Value resource handling and based on Linear Logic semantics. We have then defined Taylor expansion for Call-By-Push-Value as an approximation theory of Call-By-Push-Value encounging for resources. Then, we have shown that it behaves well with respect to the original operational semantics: Taylor expansion with coefficients commutes with reduction in Λ_{pv} . For future work, three directions shall be explored:

- The calculus can be extended in order to define inductive and coinductive datatypes. Integers, for instance, could be defined by adding to our syntax $()$: $\underline{0} = \iota_1()$, $\underline{k+1} = \iota_2(\underline{k})$, and all integers defined in this way have the type $\iota = (1 \oplus \iota)$. The successor **suc** can then be defined as the second injection. Then, if x has no free occurrence in N_1 , the term **case** $(M, x \cdot N_1, y \cdot N_2)$ is an adequate encoding of an “if zero” conditional **If** $(M, N_1, y \cdot N_2)$ (where the value to which M evaluates is passed to the following computation). The coinductive datatype of streams can also be defined: let A be a positive type, $S_A = !(A \otimes S_A)$ is the type of lazy streams of type A (the tail of the stream being always encapsulated in an exponential, the evaluation is postponed). We can construct a term of type $S_A \multimap \iota \multimap A$ which computes the k -th element of a stream:

$$\mathbf{fix}_f (\lambda x \lambda y (\mathbf{If}(y, \pi_1(\mathbf{der}(x)), z \cdot \langle \mathbf{der}(f) \rangle \pi_2 \langle \mathbf{der}(x) \rangle z)))$$

and a term of type $!(\iota \multimap A) \multimap S_A$:

$$\mathbf{fix}_f (\lambda g (\mathbf{der}(g) \underline{0}, \langle \mathbf{der}(f) \rangle (\lambda x \langle \mathbf{der}(g) \rangle \mathbf{suc}(x))^!))$$

which builds a stream by applying inductively a function to an integer. There are other classical constructions, such as lists, that can be constructed with these ingredients. For a more detailed presentation, see Ehrhard and Tasson’s work [14]. We have good hope that this kind of extensions can be incorporated in our resource driven-constructions.

- Extend our constructions in a probabilistic setting, to fit with existing quantitative models like probabilistic coherence spaces. Indeed Lemma 10, which is crucial to define reduction on quantitative Taylor expansion, strongly relies on the uniformity of the calculus, *i.e* we use the fact that all resource terms appearing in the Taylor expansion of a Call-By-Push-Value term have the same *shape* (there is a correspondance between their syntactic trees). The extension seems highly non trivial. But, Dal Lago and Leventis’ recent work [19] might be a starting point.

References

- 1 Samson Abramsky and Guy McCusker. Call-by-Value Games. In *CSL*, volume 1414 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1997.
- 2 J. Chouquet. Taylor expansion, finiteness and strategies. In *MFPS 2019*, 2019.
- 3 Jules Chouquet and Lionel Vaux Auclair. An Application of Parallel Cut Elimination in Unit-Free Multiplicative Linear Logic to the Taylor Expansion of Proof Nets. In *CSL*, volume 119 of *LIPICs*, pages 15:1–15:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
- 4 Pierre-Louis Curien, Marcelo P. Fiore, and Guillaume Munch-Maccagnoni. A theory of effects and resources: adjunction models and polarised calculi. In *POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, 2016.
- 5 V. Danos and T. Ehrhard. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Inf. Comput.*, 209(6):966–991, 2011. doi:10.1016/j.ic.2011.02.001.
- 6 Jeff Egger, Rasmus Ejlers Møgelberg, and Alex K. Simpson. The enriched effect calculus: syntax and semantics. *J. Log. Comput.*, 24:615–654, 2014.

- 7 T. Ehrhard. On Köthe Sequence Spaces and Linear Logic. *Mathematical Structures in Computer Science*, 12(5):579–623, 2002. doi:10.1017/S0960129502003729.
- 8 T. Ehrhard. Finiteness spaces. *Mathematical Structures in Computer Science*, 2005. doi:10.1017/S0960129504004645.
- 9 T. Ehrhard and L. Regnier. The differential lambda-calculus. *Theor. Comput. Sci.*, 2003.
- 10 T. Ehrhard and L. Regnier. Uniformity and the Taylor expansion of ordinary lambda-terms. *Theor. Comput. Sci.*, 403(2-3):347–372, 2008.
- 11 Thomas Ehrhard. Collapsing non-idempotent intersection types. In *CSL*, volume 16 of *LIPICs*, pages 259–273. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
- 12 Thomas Ehrhard. Call-By-Push-Value from a Linear Logic Point of View. In *ESOP*, volume 9632 of *Lecture Notes in Computer Science*, pages 202–228. Springer, 2016.
- 13 Thomas Ehrhard and Giulio Guerrieri. The Bang Calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. In *PPDP*, pages 174–187. ACM, 2016.
- 14 Thomas Ehrhard and Christine Tasson. Probabilistic call by push value. *Logical Methods in Computer Science*, 15(1), 2019.
- 15 J.-Y. Girard. Linear Logic. *Theor. Comput. Sci.*, 50:1–102, 1987.
- 16 Giulio Guerrieri and Giulio Manzonetto. The Bang Calculus and the Two Girard’s Translations. *CoRR*, abs/1904.06845, 2019. arXiv:1904.06845, doi:10.4204/EPTCS.292.2.
- 17 E. Kerinec, G. Manzonetto, and M. Pagani. Revisiting Call-by-value Böhm trees in light of their Taylor expansion. *CoRR*, abs/1809.02659, 2018. arXiv:1809.02659.
- 18 M. Kerjean and C. Tasson. Mackey-complete spaces and power series - a topological model of differential linear logic. *Mathematical Structures in Computer Science*, 28(4):472–507, 2018. doi:10.1017/S0960129516000281.
- 19 Ugo Dal Lago and Thomas Leventis. On the Taylor Expansion of Probabilistic λ -Terms. *CoRR*, abs/1904.09650, 2019. arXiv:1904.09650.
- 20 Olivier Laurent and Laurent Regnier. About Translations of Classical Logic into Polarized Linear Logic. In *18th IEEE Symposium on Logic in Computer Science (LICS 2003), 22-25 June 2003, Ottawa, Canada, Proceedings*, pages 11–20. IEEE Computer Society, 2003. doi:10.1109/LICS.2003.1210040.
- 21 P. B. Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *Higher-Order and Symbolic Computation*, 19(4):377–414, 2006. doi:10.1007/s10990-006-0480-6.
- 22 Michael Marz, Alexander Rohr, and Thomas Streicher. Full Abstraction and Universality via Realisability. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 174–182. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782612.
- 23 Damiano Mazza. An Infinitary Affine Lambda-Calculus Isomorphic to the Full Lambda-Calculus. In *LICS*, pages 471–480. IEEE Computer Society, 2012.
- 24 Damiano Mazza, Luc Pellissier, and Pierre Vial. Polyadic approximations, fibrations and intersection types. *PACMPL*, 2(POPL):6:1–6:28, 2018.
- 25 P.-A. Mellies. Categorical semantics of linear logic. In *In: Interactive Models of Computation and Program Behaviour, Panoramas et Synthèses 27, Société Mathématique de France 1–196*, 2009.
- 26 M. Pagani, C. Tasson, and L. Vaux. Strong Normalizability as a Finiteness Structure via the Taylor Expansion of lambda-terms. In *FOSSACS 2016*, pages 408–423, 2016.
- 27 L. Vaux. Taylor expansion, β -reduction and normalization. In *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, Stockholm, Sweden*, pages 39:1–39:16, 2017.

Tangent Categories from the Coalgebras of Differential Categories

Robin Cockett

University of Calgary, Department of Computer Science, Canada
<https://pages.cpsc.ucalgary.ca/~robin/>
robin@ucalgary.ca

Jean-Simon Pacaud Lemay

University of Oxford, Department of Computer Science, UK
<https://www.cs.ox.ac.uk/people/jean-simon.lemay/>
jean-simon.lemay@kellogg.ox.ac.uk

Rory B. B. Lucyshyn-Wright

Brandon University, Department of Mathematics and Computer Science, Canada
<http://lucyshyn-wright.ca/>
lucyshyn-wright@brandonu.ca

Abstract

Following the pattern from linear logic, the coKleisli category of a differential category is a Cartesian differential category. What then is the coEilenberg-Moore category of a differential category? The answer is a tangent category! A key example arises from the opposite of the category of Abelian groups with the free exponential modality. The coEilenberg-Moore category, in this case, is the opposite of the category of commutative rings. That the latter is a tangent category captures a fundamental aspect of both algebraic geometry and Synthetic Differential Geometry. The general result applies when there are no negatives and thus encompasses examples arising from combinatorics and computer science.

2012 ACM Subject Classification Theory of computation → Linear logic; Theory of computation → Categorical semantics

Keywords and phrases Differential categories, Tangent categories, Coalgebra Modalities

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.17

Related Version An extended version of the paper with the proof of Proposition 21 is available at <https://arxiv.org/abs/1910.05617>.

Funding *Robin Cockett*: Partially supported by NSERC (Canada).

Jean-Simon Pacaud Lemay: Thanks to Kellogg College, the Clarendon Fund, and the Oxford Google-DeepMind Graduate Scholarship for financial support.

Rory B. B. Lucyshyn-Wright: Supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

Acknowledgements The authors would like to thank Steve Lack for pointing us to an adjoint lifting theorem of Butler found in Barr and Wells' book [2], as well as the anonymous referee for pointing us to Johnstone's adjoint lifting theorem [27].

1 Introduction

It is well-established, following the pattern from linear logic [21], that the coKleisli category of a (tensor) differential category is a Cartesian differential category [6]. What then is the coEilenberg-Moore category of a (tensor) differential category? The answer, which is the subject of this paper, is that, under mild limit assumptions, it is a tangent category [12]. A tangent category is a category equipped with an endofunctor and several natural



© Robin Cockett, Jean-Simon Pacaud Lemay, and Rory B. B. Lucyshyn-Wright;
licensed under Creative Commons License CC-BY

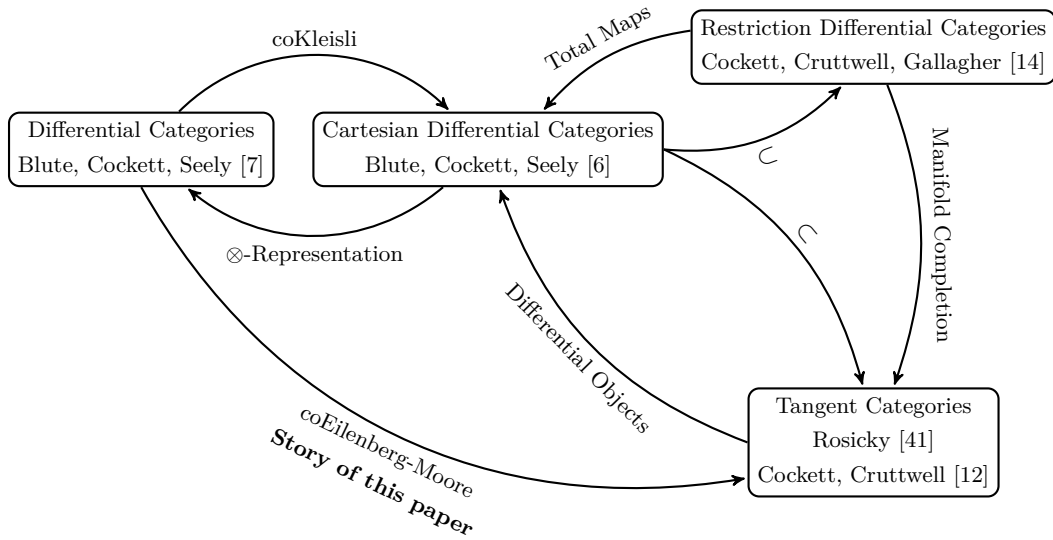
28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 17; pp. 17:1–17:17



Leibniz International Proceedings in Informatics

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



■ **Figure 1** The world of differential categories and how it’s all connected.

transformations that formalize the basic properties of the tangent bundle functor on the category of smooth manifolds. The study of the tangent category structure of coEilenberg-Moore categories of such differential categories was initiated by the third author in talks at the Category Theory 2014 conference [35] and the Foundational Methods in Computer Science conference in 2014 [34].

The use of differentiation in programming, particularly for applications in machine learning, has renewed computer scientists’ interest in the abstract semantics of differentiation. Tensor differential categories provide perhaps the simplest abstract description of differentiation. Tangent categories, on the other hand, are at quite the other end of the spectrum, providing an abstract semantics for differential geometry. That the two are directly linked by the coEilenberg-Moore construction (which is purely algebraic) witnesses that there is a surprisingly direct relationship between differential programming and differential geometry which might usefully be exploited.

In Figure 1 the relationships between the various differential categories are illustrated. The investigation of differential structure of this kind was initiated by Erhrard [17] and formulated as a categorical axiomatization in [7]. Classical smooth functions arose indirectly as the coKleisli maps of these differential categories: thus, the next step was to directly axiomatize this classical intuition. This was accomplished in [6], where Cartesian differential categories were introduced. By considering the representability of a tensor product in Cartesian differential categories it was then possible to extract a (tensor) differential category from a Cartesian differential category [4]. Classical analysis considers maps that are not defined everywhere and, thus, the theory of Cartesian differential categories with partiality was developed [14]. It was a natural step from there to consider differentiable manifolds, and this created a desire to develop a basic axiomatization for differential geometry: this led to the development of abstract differential geometry based on tangent categories, which had been introduced by Rosický in [41] and were later generalized and studied further by Cockett and Cruttwell in [12]. An important alternative approach to tangent categories was introduced by Leung [32] and was further developed by Garner in [20] to provide a view of tangent categories as categories enriched in Weil spaces. Cartesian differential categories are always

examples of tangent categories [12, Proposition 4.7]. Conversely the differential bundles of a tangent category over a fixed base, under mild limit assumptions, form a Cartesian differential category, showing that a tangent category is *locally* a Cartesian differential category [13, Theorem 5.14]. These observations tightly linked Cartesian differential and tangent categories; in fact this relationship is captured by an adjunction [12, Theorem 4.12]. The current paper provides an important direct link between (tensor) differential categories and tangent categories.

An important example of a (tensor) differential category is the opposite of the category of Abelian groups with the free exponential modality [7] where the differential structure is based on differentiating polynomials. The coEilenberg-Moore category, in this case, is the opposite of the category of commutative rings. The fact that this is a tangent category captures a fundamental aspect of both algebraic geometry [24] and Synthetic Differential Geometry [29]. That the coEilenberg-Moore category of a differential category is a tangent category in much more generality allows further significant examples. Not only can one dispense with the necessity of assuming *negatives*, but also with the necessity of having a *monoidal* coalgebra modality [4] or, equivalently, the Seely isomorphisms, $!(A \times B) \cong !(A) \otimes !(B)$ [43] (which the third author required in [34]). Dispensing with the assumption of negatives allows one to generalize the example of commutative rings to commutative semirings [22] and to consider examples from combinatorics and computer science. Dispensing with the assumption of a monoidal coalgebra modality/the Seely isomorphisms allows consideration of such examples as C^∞ -rings [29, 39] or Rota-Baxter algebras [23]. When a coalgebra modality is monoidal, it will give rise, when sufficient limits are present, to a *representable tangent category*. This means that the tangent functor is of the form $_{}^D$ for an infinitesimal object D and so has left adjoint $_{} \times D$. Two examples of such differential categories include the opposite category of vector spaces with the free commutative algebra modality (one of the original examples of a differential category in [7]), as well as the the category of vector spaces with the cofree cocommutative coalgebra modality (as studied by Clift and Murfet in [11]). It is interesting to note that in both cases, the infinitesimal object is the ring of dual numbers over the field. On the other hand, as the free C^∞ -ring modality is not monoidal, this provides an example of a non-representable tangent category that, nonetheless, has a tangent functor that is a right adjoint.

Conventions: This paper assumes a knowledge of basic category theory and of symmetric monoidal categories. We refer the reader to [36] if further details are needed on these topics. In this paper we shall use diagrammatic order for composition: explicitly, this means that the composite map fg is the map that employs f first and then g . Furthermore, to simplify working in symmetric monoidal categories, we will allow ourselves to work in symmetric *strict* monoidal categories, and therefore we suppress the associator and unitor isomorphisms. Symmetric monoidal categories will be denoted by $(\mathbb{X}, \otimes, K, \tau)$ where \mathbb{X} is the underlying category, \otimes is the tensor product, K is the monoidal unit, and $\tau_{A,B} : A \otimes B \rightarrow B \otimes A$ is the symmetry isomorphism.

2 Tangent Categories

Tangent categories were introduced by Rosický in [41], then later generalized and studied further by Cockett and Cruttwell in [12]. This generalization, which replaced Abelian groups with commutative monoids, opened the door to examples of tangent categories from

17:4 Tangent Categories from the Coalgebras of Differential Categories

combinatorics and computer science where one does not expect to have negatives. The axioms of a tangent category abstract the essential properties of tangent bundles over smooth manifolds [31]. In this section we provide a brief overview of tangent categories, and refer the reader to [12, 20] for a more in-depth introduction.

► **Definition 1.** Let \mathbb{X} be a category. A **tangent structure** [12] \mathbb{T} on \mathbb{X} is a sextuple $\mathbb{T} := (\mathbb{T}, p, \sigma, z, \ell, c)$ consisting of:

- An endofunctor $\mathbb{T} : \mathbb{X} \rightarrow \mathbb{X}$;
- A natural transformation $p_M : \mathbb{T}(M) \rightarrow M$, known as the **projection**, such that for each M and each $n \in \mathbb{N}$, there is an n -th fibre power¹ $\mathbb{T}_n(M)$ of p_M (with projections $\rho_i : \mathbb{T}_n(M) \rightarrow \mathbb{T}(M)$), and this fibre power is preserved by \mathbb{T}^m for each $m \in \mathbb{N}$. For each $n \in \mathbb{N}$, this induces a functor $\mathbb{T}_n : \mathbb{X} \rightarrow \mathbb{X}$ where by convention $\mathbb{T}_0 = 1_{\mathbb{X}}$ and $\mathbb{T}_1 = \mathbb{T}$.
- Natural transformations $\sigma_M : \mathbb{T}_2(M) \rightarrow \mathbb{T}(M)$, known as the **sum operation on tangent vectors**, $z_M : M \rightarrow \mathbb{T}(M)$, known as the **zero vector field**, $\ell_M : \mathbb{T}(M) \rightarrow \mathbb{T}^2(M)$, known as the **vertical lift**, and $c_M : \mathbb{T}^2(M) \rightarrow \mathbb{T}^2(M)$, known as the **canonical flip**; and such that p , σ , z , ℓ , and c satisfy the various equational axioms found in [12, 20] and that for each M , the following diagram is an equalizer diagram [36], known as the **universality of the vertical lift**:

$$\mathbb{T}_2(M) \xrightarrow{\langle \rho_0 z_{\mathbb{T}(M)}, \rho_1 \ell_M \rangle \mathbb{T}(\sigma_M)} \mathbb{T}^2(M) \xrightarrow[\substack{\mathbb{T}(p_M) \\ p_{\mathbb{T}(M)} p_M z_M}]{\mathbb{T}(p_M)} \mathbb{T}(M)$$

where $\langle -, - \rangle$ is the pairing operation induced by the universal property of the pullback.

► **Definition 2.** A **tangent category** [12] is a pair (\mathbb{X}, \mathbb{T}) consisting of a category \mathbb{X} and a tangent structure \mathbb{T} on \mathbb{X} . The fibre powers of p , together with the equalizer appearing in the axiom of universality of the vertical lift, are collectively referred to as the **tangent limits** [20] of a tangent category.

We refer the reader to [12] where the axioms of a tangent category are expressed in commutative diagrams. In [32], Leung defined an alternative axiomatization of a tangent category using Weil algebras; this was exploited by Garner in [20] to provide a description of tangent categories as categories enriched in Weil spaces.

► **Example 3.** Here are some well-known examples of tangent categories. Other examples of tangent categories can be found in [12, 20, 13].

1. The canonical example of a tangent category is the category of finite-dimensional smooth manifolds, where for a manifold M , $\mathbb{T}(M)$ is the standard tangent bundle over M .
2. Every Cartesian differential category [6] is a tangent category [12, Proposition 4.7]. In particular this implies that every categorical model of the differential λ -calculus [10, 37] is a tangent category.
3. Let k be a field, and let CALG_k be the category of commutative k -algebras. Then CALG_k is a tangent category where for a commutative k -algebra A , $\mathbb{T}(A) := A[\epsilon]$ is the ring of dual numbers over A , $A[\epsilon] = \{a + b\epsilon \mid a, b \in A\}$ with $\epsilon^2 = 0$. The projection is defined as $p_A(a + b\epsilon) = a$, and so $\mathbb{T}_2(A) := A[\epsilon, \epsilon'] = \{a + b\epsilon + c\epsilon' \mid a, b, c \in A\}$ with $\epsilon^2 = \epsilon'^2 = \epsilon\epsilon' = 0$. On the other hand, $\mathbb{T}^2(A) := \{a + b\epsilon_1 + c\epsilon_2 + d\epsilon_1\epsilon_2 \mid a, b, c, d \in A\}$ with $\epsilon_1^2 = \epsilon_2^2 = 0$. The remaining tangent structure is defined as follows: $\sigma_A(a + b\epsilon + c\epsilon') = a + (b + c)\epsilon$, $z_A(a) = a$, $\ell_A(a + b\epsilon) = a + b\epsilon_1\epsilon_2$, and $c_A(a + b\epsilon_1 + c\epsilon_2 + d\epsilon_1\epsilon_2) = a + c\epsilon_1 + b\epsilon_2 + d\epsilon_1\epsilon_2$.

¹ I.e., a fibered product [36] of n instances of p_M

We will generalize this example in the context of codifferential categories in Section 5. In particular, this example generalizes to the category of commutative algebras over any commutative unital semiring.

Of particular importance to this paper is when the tangent functor has a left adjoint, which induces a tangent structure on the opposite category of the tangent category.

► **Theorem 4.** [12, Proposition 5.17] *Let (\mathbb{X}, \mathbb{T}) be a tangent category such that for each $n \in \mathbb{N}$, $\mathbb{T}_n : \mathbb{X} \rightarrow \mathbb{X}$ has a left adjoint $\mathbb{L}_n : \mathbb{X} \rightarrow \mathbb{X}$. Then \mathbb{X}^{op} has a tangent structure with tangent functor $\mathbb{L} = \mathbb{L}_1$.*

See Example 7.3 for an application of this theorem. In Section 6 we will use Theorem 4 to obtain a tangent structure on the coEilenberg-Moore category of a differential category.

We now turn our attention to *representable tangent categories*, which briefly are tangent categories whose tangent functor is representable. Representable tangent categories are a very important kind of tangent category as they are closely related to synthetic differential geometry [29]. First recall that in a category \mathbb{X} with binary products \times , an object D is an **exponent object** if the functor $- \times D : \mathbb{X} \rightarrow \mathbb{X}$ has a right adjoint $(-)^D : \mathbb{X} \rightarrow \mathbb{X}$. A functor $F : \mathbb{X} \rightarrow \mathbb{X}$ is **representable** if $F(-) \cong (-)^D$ for some exponent object D , and D is said to represent the functor F .

► **Definition 5.** *A **representable tangent category** [12] is a tangent category (\mathbb{X}, \mathbb{T}) such that \mathbb{X} has finite products and for each $n \in \mathbb{N}$, \mathbb{T}_n is a representable functor, that is, $\mathbb{T}_n(-) \cong (-)^{D_n}$ for some exponent object D_n . In the case of $n = 1$, the object D_1 (which we denote simply as D) representing the tangent functor, $\mathbb{T}(-) \cong (-)^D$, is called the **infinitesimal object** [12] of the representable tangent category (\mathbb{X}, \mathbb{T}) .*

Alternatively one can axiomatize representable tangent categories in terms of the infinitesimal object D , see [12, Definition 5.6]. Note that by definition, a representable functor has a left adjoint and therefore one can apply Theorem 4 to a representable tangent category.

► **Theorem 6.** [12, Corollary 5.18] *Let (\mathbb{X}, \mathbb{T}) be a representable tangent category with infinitesimal object D . Then \mathbb{X}^{op} has a tangent structure with tangent functor $- \times D$.*

► **Example 7.** We finish this section with some examples of representable tangent categories.

1. Every tangent category embeds into a representable tangent category [20].
2. The subcategory of infinitesimally and vertically linear objects of any model of synthetic differential geometry [29] is a representable tangent category with infinitesimal object $D = \{x \in R \mid x^2 = 0\}$, where R is the line object [12, Proposition 5.10].
3. Let k be a field. Recall that in CALG_k , the categorical coproduct is given by the tensor product of k -vector spaces \otimes which is therefore a product in CALG_k^{op} . Then CALG_k^{op} is a representable tangent category with infinitesimal object $k[\epsilon]$, the ring of dual numbers over k . For a commutative k -algebra A , $A^{k[\epsilon]}$ (in CALG_k^{op}) is defined as the symmetric A -algebra over the Kähler module of A (see [12, Proposition 5.16] for full details). By applying Theorem 6 to this example, one obtains precisely the tangent structure on CALG_k from Example 3.3, where in particular we note that $A[\epsilon] \cong A \otimes k[\epsilon]$. In Section 6 we will generalize this example to the context of differential categories. We again note that this example generalizes to the category of commutative algebras over any commutative semiring.

3 Coalgebra Modalities and their coEilenberg-Moore Categories

In this section we review (co)algebra modalities and take a look at their (co)Eilenberg-Moore categories. Coalgebra modalities were introduced in the development of differential categories [7], as a weakening of the notion of *linear exponential comonad* that is required for a categorical model of the multiplicative and exponential fragment of linear logic (MELL) [3, 42, 38]. While the notion of a coalgebra modality is strictly weaker than that of a linear exponential comonad – which is precisely a *monoidal coalgebra modality* [4] – coalgebra modalities provide a sufficient context in which to axiomatize differentiation.

A **comonad** [36] on a category \mathbb{X} will be denoted as a triple $(!, \delta, \varepsilon)$ with endofunctor $! : \mathbb{X} \rightarrow \mathbb{X}$ and natural transformations $\delta_A : !(A) \rightarrow !(A)$ and $\varepsilon_A : !(A) \rightarrow A$. A **!-coalgebra** will be denoted as a pair (A, ω) with underlying object A and !-coalgebra structure $\omega : A \rightarrow !(A)$. The category of !-coalgebras and !-coalgebra morphisms is called the **coEilenberg-Moore category** [36] of the comonad $(!, \delta, \varepsilon)$ and will be denoted $\mathbb{X}^!$. Coalgebra modalities are comonads such that every cofree !-coalgebra comes equipped with a natural cocommutative comonoid structure.

► **Definition 8.** A *coalgebra modality* [7] on a symmetric monoidal category is a quintuple $(!, \delta, \varepsilon, \Delta, \mathbf{e})$ consisting of a comonad $(!, \delta, \varepsilon)$ equipped with two natural transformations $\Delta_A : !(A) \rightarrow !(A) \otimes !(A)$ and $\mathbf{e}_A : !(A) \rightarrow K$ such that for each object A , $(!(A), \Delta_A, \mathbf{e}_A)$ is a cocommutative comonoid and δ_A is a comonoid morphism.

What can we say about the coEilenberg-Moore category of a coalgebra modality? It turns out that every !-coalgebra of a coalgebra modality comes equipped with a cocommutative comonoid structure [9]. Indeed if (A, ω) is a !-coalgebra, then the triple $(A, \Delta^\omega, \mathbf{e}^\omega)$ is a cocommutative comonoid where Δ^ω and \mathbf{e}^ω are defined as follows:

$$\Delta^\omega := A \xrightarrow{\omega} !(A) \xrightarrow{\Delta_A} !(A) \otimes !(A) \xrightarrow{\varepsilon_A \otimes \varepsilon_A} A \otimes A \quad \mathbf{e}^\omega := A \xrightarrow{\omega} !(A) \xrightarrow{\mathbf{e}_A} K$$

It is important to point out that $(A, \Delta^\omega, \mathbf{e}^\omega)$ is in general only a cocommutative comonoid in the base category \mathbb{X} and not in the coEilenberg-Moore category $\mathbb{X}^!$, since the latter does not necessarily have a monoidal product. Also notice that since δ_A is a comonoid morphism, when applying this construction to a cofree !-coalgebra $(!(A), \delta_A)$ we recover Δ_A and \mathbf{e}_A , that is, $\Delta^{\delta_A} = \Delta_A$ and $\mathbf{e}^{\delta_A} = \mathbf{e}_A$.

► **Definition 9.** In a symmetric monoidal category with finite products \times and terminal object 1 , a coalgebra modality has **Seely isomorphisms** [3, 4, 43] if the map $\chi_1 : !(1) \rightarrow K$ and natural transformation $\chi : !(A \times B) \rightarrow !(A) \otimes !(B)$ defined respectively as

$$!(1) \xrightarrow{\mathbf{e}} K \quad !(A \times B) \xrightarrow{\Delta} !(A \times B) \otimes !(A \times B) \xrightarrow{!(\pi_0) \otimes !(\pi_1)} !(A) \otimes !(B)$$

are isomorphisms, so $!(1) \cong K$ and $!(A \times B) \cong !(A) \otimes !(B)$.

Coalgebra modalities with Seely isomorphisms can equivalently be defined as **monoidal coalgebra modalities** [3, 4], which are coalgebra modalities equipped with a natural transformation $\mathbf{m}_{A,B} : !(A) \otimes !(B) \rightarrow !(A \otimes B)$ and a map $\mathbf{m}_K : K \rightarrow !(K)$ making ! a symmetric monoidal comonad such that Δ and \mathbf{e} are both monoidal transformations and !-coalgebra morphisms. Furthermore for a monoidal coalgebra modality, the monoidal product of the base category becomes a finite product in the coEilenberg-Moore category [42]. Explicitly, the terminal object is the !-coalgebra (K, \mathbf{m}_K) while the product of !-coalgebras (A, ω) and (B, ω') is $(A, \omega) \otimes (B, \omega') := (A \otimes B, (\omega \otimes \omega') \mathbf{m}_{A,B})$.

► **Example 10.** Here are some examples of coalgebra modalities. Many other examples of coalgebra modalities (with and without the Seely isomorphisms) can be found in [5].

1. There is no shortage of examples of coalgebra modalities since every categorical model of MELL admits a coalgebra modality which has the Seely isomorphism. For example, Hyland and Schalk provide a nice list of such examples in [26, Section 2.4].
2. Let k be a field and let VEC_k be the category of k -vector spaces, which is a symmetric monoidal category with respect to the standard tensor product of k -vector spaces. For every k -vector space V , there exists a cofree cocommutative k -coalgebra [44] over V , denoted $!(V)$, where a detailed construction can be found in [11, 26, 44]. In particular, if k has characteristic 0² and if $X = \{x_i \mid i \in I\}$ is a basis of V , then $!(V) \cong \bigoplus_{v \in V} k[X]$ as k -coalgebras (where $k[X]$ is the polynomial ring over k generated by the set X). This induces a coalgebra modality $!$ on VEC_k which furthermore has the Seely isomorphisms ($!(V \times W) \cong !(V) \otimes !(W)$ and $!(0) \cong k$), and by [40] we know that the coEilenberg-Moore category of $!$ is isomorphic to the category of cocommutative k -coalgebras (which are the cocommutative comonoids in VEC_k). By applying results in [40], one can generalize this example to the category of modules over an arbitrary commutative unital ring.

The dual notion of a coalgebra modality is an algebra modality. Since we will be working with algebra modalities in Section 5, we provide the definition of an algebra modality in detail. A **monad** [36], the dual notion a comonad, on a category \mathbb{X} will be denoted as a triple (S, μ, η) consisting of an endofunctor $S : \mathbb{X} \rightarrow \mathbb{X}$ and natural transformations $\mu_A : S^2(A) \rightarrow S(A)$ and $\eta_A : S(A) \rightarrow A$. An S -algebra will be denoted as a pair (A, ν) with underlying object A and structure map $\nu : S(A) \rightarrow A$. The category of S -algebras and S -algebra morphisms is called the **Eilenberg-Moore category** [36] of the monad (S, μ, η) and is denoted \mathbb{X}^S .

► **Definition 11.** An *algebra modality* [9] on a symmetric monoidal category is a quintuple $(S, \mu, \eta, \nabla, \mathbf{u})$ consisting of a monad (S, μ, η) equipped with two natural transformations $\nabla_A : S(A) \otimes S(A) \rightarrow S(A)$ and $\mathbf{e} : K \rightarrow S(A)$ such that for each object A , $(S(A), \nabla_A, \mathbf{u}_A)$ is a commutative monoid and μ_A is a monoid morphism.

Since algebra modalities are dual to coalgebra modalities, it follows that every S -algebra comes equipped with a commutative monoid structure [9], which we again point out is in the base category and not the Eilenberg-Moore category. Explicitly, given an S -algebra (A, ν) of an algebra modality $(S, \mu, \eta, \nabla, \mathbf{u})$, the triple $(A, \nabla^\nu, \mathbf{u}^\nu)$ is a commutative monoid where ∇^ν and \mathbf{u}^ν are defined as follows:

$$\nabla^\nu := A \otimes A \xrightarrow{\eta_A \otimes \eta_A} S(A) \otimes S(A) \xrightarrow{\nabla_A} S(A) \xrightarrow{\nu_A} A \quad \mathbf{u}^\nu := K \xrightarrow{\mathbf{u}_A} S(A) \xrightarrow{\nu} A$$

Dual to coalgebra modalities with the Seely isomorphisms, in the case of a symmetric monoidal category with finite coproducts \oplus and initial object 0 , if the algebra modality has the Seely isomorphisms, i.e. $S(0) \cong K$ and $S(A \oplus B) \cong S(A) \otimes S(B)$, then the monoidal product becomes a coproduct in Eilenberg-Moore category.

► **Example 12.** Here are some examples of algebra modalities. Many other examples of algebra modalities (with and without the Seely isomorphisms) can be found in [5].

² In [11], Clift and Murfet work, for simplicity, with an algebraically closed field of characteristic 0. However, as they point out, the assumption that the field is algebraically closed is not necessary in the construction.

1. Let k be a field. In analogy with Example 10.2, for every k -vector space V there exists a free commutative k -algebra over V , denoted $\mathbf{Sym}(V)$, which is also called the symmetric algebra over V [30]. In particular, if $X = \{x_i \mid i \in I\}$ is a basis of V , then $\mathbf{Sym}(V) \cong k[X]$ as k -algebras (where $k[X]$ is the polynomial k -algebra generated by the set X). This induces an algebra modality \mathbf{Sym} on \mathbf{VEC}_k which furthermore has the Seelye isomorphisms (so that $\mathbf{Sym}(V \times W) \cong \mathbf{Sym}(V) \otimes \mathbf{Sym}(W)$ and $\mathbf{Sym}(0) \cong k$) and whose Eilenberg-Moore category is isomorphic to the category of commutative k -algebras (which are the commutative monoids in \mathbf{VEC}_k). This example generalizes to the category of modules over an arbitrary commutative unital semiring.
2. Let \mathbb{R} be the field of real numbers. C^∞ -rings [29, 39] are defined as the algebras of the Lawvere theory whose morphisms are smooth maps between Cartesian spaces \mathbb{R}^n , so a C^∞ -ring can be defined equivalently as a set A equipped with a family of functions $\Phi_f : A^n \rightarrow A$ indexed by the smooth functions $f : \mathbb{R}^n \rightarrow \mathbb{R}$, satisfying certain equations. For example, if M is a smooth manifold, then $C^\infty(M) = \{f : M \rightarrow \mathbb{R} \mid f \text{ is smooth}\}$ is a C^∞ -ring where for a smooth map $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $\Phi_f : C^\infty(M)^n \rightarrow C^\infty(M)$ is defined by post-composition by f . Every C^∞ -ring is a commutative \mathbb{R} -algebra and for every \mathbb{R} -vector space V there exists a free C^∞ -ring over V [28, Theorem 3.3][16], denoted as $\mathbf{S}^\infty(V)$. This induces an algebra modality \mathbf{S}^∞ on $\mathbf{VEC}_{\mathbb{R}}$ [16], where in particular for a finite dimensional vector space V of dimension n , one has that $\mathbf{S}^\infty(V) \cong C^\infty(\mathbb{R}^n)$. The Eilenberg-Moore category of \mathbf{S}^∞ is the category of C^∞ -rings [29, 39, 16]. It is important to note that this is an example of an algebra modality which does NOT have the Seelye isomorphisms. However, C^∞ -rings are mathematically important, as they provide the basis for well-adapted models of synthetic differential geometry [29, 39] and provide a natural setting for the adaptation of algebro-geometric methods to a smooth context.

4 Differential Categories

In this section we review (co)differential categories and, in particular, we will take a look at some well-known examples that correspond to differentiating polynomials and smooth maps. Differential categories were introduced by Blute, Cockett, and Seelye [7] to provide the categorical semantics of differential linear logic [18]. While a codifferential category is simply the dual of a differential category, we provide full definitions of each since we will be working with codifferential categories in Section 5 and in the appendix, and we will be working with differential categories in Section 6.

Two of the basic properties of the derivative from classical differential calculus require addition (or at least the number 0): the Leibniz rule and the constant rule. Therefore we must first discuss additive structure. Here we mean “additive” in the sense of [7], that is, enriched over commutative monoids. In particular this definition does not assume negatives nor does it assume biproducts, so this differs from other definitions of an additive category such as in [36]. That said, in Section 5 and Section 6 we will be working with (co)differential categories with biproducts.

► **Definition 13.** *An **additive category** is a category enriched in commutative monoids, that is, a category in which each hom-set is a commutative monoid, with addition operation $+$ and zero 0 , and in which composition preserves the additive structure in the sense that $k(f+g)h = (kfh) + (kgh)$ and $0f = 0 = f0$. An **additive symmetric monoidal category** [7] is a symmetric monoidal category that is also an additive category such that the monoidal product \otimes is compatible with the additive structure in the sense that $k \otimes (f + g) \otimes h = k \otimes f \otimes h + k \otimes g \otimes h$ and $f \otimes 0 \otimes g = 0$.*

Every category with finite biproducts is an additive category, and finite (co)products in an additive category are automatically finite biproducts. In fact, every additive category can be completed to a category with biproducts [36], where the completion is also an additive category, and similarly every additive symmetric monoidal category can be completed to an additive symmetric monoidal category with distributive biproducts. For this reason it is possible to argue (as in [19]) that one might as well always assume one has biproducts. However, it is important to bear in mind that only monoidal coalgebra modalities are guaranteed to lift to the biproduct completion. Thus, for a treatment of arbitrary coalgebra modalities this assumption cannot be made (see [5] for more details).

► **Definition 14.** A *differential category* [7] is an additive symmetric monoidal category with a coalgebra modality $(!, \delta, \varepsilon, \Delta, \mathbf{e})$ that comes equipped with a **deriving transformation**, that is, a natural transformation $\mathbf{d}_A : !(A) \otimes A \rightarrow !(A)$ such that the following equalities hold:

[d.1] Constant Rule: $\mathbf{d}_A \mathbf{e}_A = 0$

[d.2] Leibniz Rule: $\mathbf{d}_A \Delta_A = (\Delta_A \otimes 1_{!(A)})(1_{!(A)} \otimes \tau_{!(A), A})(\mathbf{d}_A \otimes 1_{!(A)}) + (\Delta_A \otimes 1_{!(A)})(1_{!(A)} \otimes \mathbf{d}_A)$

[d.3] Linear Rule: $\mathbf{d}_A \varepsilon_A = \mathbf{e}_A \otimes 1_A$

[d.4] Chain Rule: $\mathbf{d}_A \delta_A = (\Delta_A \otimes 1_A)(\delta_A \otimes \mathbf{d}_A) \mathbf{d}_{!(A)}$

[d.5] Interchange Rule³: $(\mathbf{d}_A \otimes 1_A) \mathbf{d}_A = (1 \otimes \tau_{A, A})(\mathbf{d}_A \otimes 1_A) \mathbf{d}_A$

CoKleisli maps of coalgebra modalities, that is, maps of type $f : !(A) \rightarrow B$, are to be thought of as *smooth* maps from A to B as they are, in a certain sense, infinitely differentiable. Indeed the *derivative* of $f : !(A) \rightarrow B$ is the map $\mathbf{D}[f] : !(A) \otimes A \rightarrow B$, defined as the composite $\mathbf{D}[f] := \mathbf{d}_A f$. The constant rule **[d.1]** amounts to the statement that the derivative of a constant map is zero. The second axiom **[d.2]** is the analogue of the classical Leibniz rule in differential calculus. For the third axiom, a subclass of smooth maps are the *linear* maps, which are coKleisli maps of the form $\varepsilon_{Ag} : !(A) \rightarrow B$ for some map $g : A \rightarrow B$. Then the linear rule **[d.3]** says that the derivative of a linear map is “constant” with respect to the point at which it is taken. The fourth axiom **[d.4]** is the chain rule regarding composition in the coKleisli category. The interchange rule **[d.5]**, is the independence of order of differentiation, which, naively put, says that differentiating with respect to x then y is the same as differentiation with respect to y then x . For more details and for string diagram representation of the axioms of a differential category, we refer the reader to [7, 5].

► **Definition 15.** A *differential storage category* [7] is a differential category with finite biproducts whose coalgebra modality has the Seelye isomorphisms.

For differential storage categories, the differential category structure can equivalently be axiomatized by a natural transformation $\eta_A : A \rightarrow !(A)$ known as a **coderelection** [7, 5, 19]. For monoidal coalgebra modalities, there is a bijective correspondence between coderelections and deriving transformations [5]. However, we will see below that the deriving transformation plays the more important role when discussing tangent category structure of (co-)Eilenberg-Moore categories of differential categories.

► **Example 16.** Here are some examples of differential storage categories. Many other examples of differential (storage) categories can be found in [7, 5, 18].

³ It should be noted that the interchange rule **[d.5]** was not part of the definition in [7] but was later added to ensure that the coKleisli category of a differential category (with finite products) was a Cartesian differential category [6].

17:10 Tangent Categories from the Coalgebras of Differential Categories

1. In [8], Blute, Ehrhard, and Tasson showed that the category of convenient vector spaces and bounded linear maps between them is a differential storage category. In particular, the coKleisli category is precisely the category of convenient vector spaces and *smooth* maps between them. The differential structure is induced by the limit definition of the derivative in locally convex vector spaces.
2. In [18], Ehrhard provides a differential storage category (amongst others) whose objects are *linearly topologized* vector spaces generated by finiteness spaces. The differential structure corresponds to differentiating multivariable power series.
3. In [11], Clift and Murfet study the differential category structure induced by cofree cocommutative coalgebras. So if k is a field of characteristic 0, \mathbf{VEC}_k is a differential storage category with the coalgebra modality $!$ defined in Example 10.2. Recalling that for a vector space V with basis X , $!(V) \cong \bigoplus_{v \in V} k[X]$, the deriving transformation can be expressed as:

$$d_V : \left(\bigoplus_{v \in V} k[X] \right) \otimes V \rightarrow \bigoplus_{v \in V} k[X] \quad p_v(x_1, \dots, x_n) \otimes x_i \mapsto p_v(x_1, \dots, x_n) x_i$$

where $p_v(x_1, \dots, x_n)$ is a polynomial in distinct indeterminates $x_1, x_2, \dots, x_n \in X$ and lies in the v -th coproduct-component of $!V$, where $v \in V$. This example also generalizes for modules over a commutative unital semiring.

► **Definition 17.** A *codifferential category* [9] is an additive symmetric monoidal category with an algebra modality $(S, \mu, \eta, \nabla, \mathbf{u})$ that comes equipped with a **deriving transformation**⁴, that is, a natural transformation $d_A : S(A) \rightarrow S(A) \otimes A$ such that dual equalities of [d.1] to [d.5] hold, that is:

[cd.1] *Constant Rule:* $\mathbf{u}_A d_A = 0$

[cd.2] *Leibniz Rule:*

$$\nabla_A d_A = (d_A \otimes \mathbf{1}_{S(A)})(\mathbf{1}_{S(A)} \otimes \tau_{A, S(A)})(\nabla_A \otimes \mathbf{1}_{S(A)}) + (\mathbf{1}_{S(A)} \otimes d_A)(\nabla_A \otimes \mathbf{1}_{S(A)})$$

[cd.3] *Linear Rule:* $\eta_A d_A = \mathbf{u}_A \otimes \mathbf{1}_A$

[cd.4] *Chain Rule:* $\mu_A d_A = d_{S(A)}(\mu_A \otimes d_A)(\nabla_A \otimes \mathbf{1}_A)$

[cd.5] *Interchange Rule:* $d_A(d_A \otimes \mathbf{1}_A) = d_A(d_A \otimes \mathbf{1}_A)(\mathbf{1} \otimes \tau_{A, A})$

► **Example 18.** Here are some examples of codifferential categories. Many other examples of codifferential categories can be found in [7, 5].

1. Let k be a field. Then \mathbf{VEC}_k is a codifferential storage category with the algebra modality \mathbf{Sym} defined in Example 12.1, and where the differential structure corresponds precisely to differentiation of polynomials. To see this, recall that for a k -vector space V with basis set X , $\mathbf{Sym}(V) \cong k[X]$. Therefore the deriving transformation can be expressed as $d_V : k[X] \rightarrow k[X] \otimes V$, which is given by taking a sum involving the partial derivatives:

$$d_V : k[X] \rightarrow k[X] \otimes V \quad p(x_1, \dots, x_n) \mapsto \sum_{i=1}^n \frac{\partial p}{\partial x_i}(x_1, \dots, x_n) \otimes x_i$$

So \mathbf{VEC}_k^{op} is a differential storage category. See [7, 5] for full details on this example. This example generalizes to the category of modules over a commutative unital semiring.

⁴ As in the literature, we keep the same terminology and notation for a deriving transformation in the context of a codifferential category

2. Other than the codifferential category structure given by Sym from the previous example, $\text{VEC}_{\mathbb{R}}$ also has a codifferential category structure with respect to the algebra modality \mathbb{S}^∞ defined in Example 12.2. The deriving transformation is induced by differentiating smooth functions. In particular for \mathbb{R}^n , $\mathbb{S}^\infty(\mathbb{R}^n) = C^\infty(\mathbb{R}^n)$ and the deriving transformation $d_{\mathbb{R}^n} : C^\infty(\mathbb{R}^n) \rightarrow C^\infty(\mathbb{R}^n) \otimes \mathbb{R}^n$ is defined as a sum involving the partial derivatives:

$$d_{\mathbb{R}^n} : C^\infty(\mathbb{R}^n) \rightarrow C^\infty(\mathbb{R}^n) \otimes \mathbb{R}^n \quad f \mapsto \sum_{i=1}^n \frac{\partial f}{\partial x_i} \otimes x_i$$

Hence $\text{VEC}_{\mathbb{R}}^{\text{op}}$ is a differential category. See [16] for full details on this example.

5 Tangent Structure and Codifferential Categories

The goal of this section is to prove that the Eilenberg-Moore category of a codifferential category with finite biproducts is a tangent category. To achieve this we need to first introduce the concept of a *tangent monad*, in order to lift tangent structure to Eilenberg-Moore categories.

Let (\mathbb{S}, μ, η) be a monad on a category \mathbb{X} and let $\mathbb{T} : \mathbb{X} \rightarrow \mathbb{X}$ be an endofunctor. Recall that a **distributive law** [45] of $\mathbb{T} : \mathbb{X} \rightarrow \mathbb{X}$ over (\mathbb{S}, μ, η) is a natural transformation $\lambda_M : \mathbb{S}(\mathbb{T}(M)) \rightarrow \mathbb{T}(\mathbb{S}(M))$ such that the following diagrams commute:

$$\begin{array}{ccc} \mathbb{S}^2\mathbb{T}(M) \xrightarrow{\mathbb{S}(\lambda_M)} \mathbb{S}\mathbb{T}\mathbb{S}(M) \xrightarrow{\lambda_{\mathbb{S}(M)}} \mathbb{T}\mathbb{S}^2(M) & & \mathbb{T}(M) \xrightarrow{\eta_{\mathbb{T}(M)}} \mathbb{S}\mathbb{T}(M) \\ \mu_{\mathbb{T}(M)} \downarrow & & \downarrow \lambda_M \\ \mathbb{S}\mathbb{T}(M) \xrightarrow{\lambda_M} \mathbb{T}\mathbb{S}(M) & & \mathbb{T}(\eta_M) \searrow \downarrow \lambda_M \\ & & \mathbb{T}\mathbb{S}(M) \end{array}$$

Distributive laws of this sort allow us to lift \mathbb{T} to the Eilenberg-Moore category of (\mathbb{S}, μ, η) [45], noting that this is an instance of a more general result of Appelgate that is stated in [27]. Explicitly, the endofunctor $\bar{\mathbb{T}} : \mathbb{X}^{\mathbb{S}} \rightarrow \mathbb{X}^{\mathbb{S}}$, called the lifting of \mathbb{T} , is defined on objects by $\bar{\mathbb{T}}(A, \mathbb{S}(A) \xrightarrow{\nu} A) := (\mathbb{T}(A), \mathbb{S}\mathbb{T}(A) \xrightarrow{\lambda_A} \mathbb{T}\mathbb{S}(A) \xrightarrow{\mathbb{T}(\nu)} \mathbb{T}(A))$ and on maps by $\bar{\mathbb{T}}(f) := \mathbb{T}(f)$.

► **Definition 19.** A *tangent monad* on a tangent category (\mathbb{X}, \mathbb{T}) is a quadruple $(\mathbb{S}, \eta, \mu, \lambda)$ consisting of a monad (\mathbb{S}, η, μ) equipped with a distributive law λ of the tangent functor \mathbb{T} over $(\mathbb{S}, \mu, \delta)$ such that the following diagrams commute:

$$\begin{array}{ccc} \mathbb{S}\mathbb{T}(M) & & \mathbb{S}\mathbb{T}_2(M) \xrightarrow{\mathbb{S}(\sigma_M)} \mathbb{S}\mathbb{T}(M) & & \mathbb{S}(M) \xrightarrow{\mathbb{S}(z_M)} \mathbb{S}\mathbb{T}(M) \\ \lambda_M \downarrow & \searrow \mathbb{S}(p_M) & \downarrow \lambda_M & & \downarrow \lambda_M \\ \mathbb{T}\mathbb{S}(M) \xrightarrow{p_{\mathbb{S}(M)}} \mathbb{S}(M) & & \mathbb{T}_2\mathbb{S}(M) \xrightarrow{\sigma_{\mathbb{S}(M)}} \mathbb{T}\mathbb{S}(M) & & \mathbb{T}\mathbb{S}(M) \end{array}$$

$$\begin{array}{ccc} \mathbb{S}\mathbb{T}(M) \xrightarrow{\mathbb{S}(\ell_M)} \mathbb{S}\mathbb{T}^2(M) & & \mathbb{S}\mathbb{T}^2(M) \xrightarrow{\mathbb{S}(c_M)} \mathbb{S}\mathbb{T}^2(M) \\ \lambda_M \downarrow & \searrow \lambda_{\mathbb{T}(M)} & \downarrow \lambda_{\mathbb{T}(M)} \\ \mathbb{T}\mathbb{S}(M) \xrightarrow{\ell_{\mathbb{S}(M)}} \mathbb{T}^2\mathbb{S}(M) & & \mathbb{T}\mathbb{S}(M) \xrightarrow{c_{\mathbb{S}(M)}} \mathbb{T}^2\mathbb{S}(M) \end{array}$$

17:12 Tangent Categories from the Coalgebras of Differential Categories

Equivalently a tangent monad is a monad in the 2-category of tangent categories [20] since the above diagrams imply that (S, λ) is a tangent morphism [12].

► **Proposition 20.** *The Eilenberg-Moore category of a tangent monad is a tangent category such that the forgetful functor preserves the tangent structure strictly.*

Proof. Let (S, η, μ, λ) be a tangent monad on a tangent category (\mathbb{X}, \mathbb{T}) . Since λ is a distributive law, this induces a lifting functor $\bar{\mathbb{T}} : \mathbb{X}^S \rightarrow \mathbb{X}^S$. Then we can define $\bar{p}_{(A, \nu)} := p_A$, $\bar{z}_{(A, \nu)} := z_A$, $\bar{\ell}_{(A, \nu)} := \ell_A$, $\bar{c}_{(A, \nu)} := c_A$. That \bar{p} , \bar{z} , $\bar{\ell}$, and \bar{c} are all S -algebra morphisms follows from naturality of p, z, ℓ, c and the respective diagrams of a tangent monad. To define the addition map $\bar{\sigma}$, we must first address limits in \mathbb{X}^S . It is well known that any given diagram has a limit in the Eilenberg-Moore category as soon as it has a limit in the base category [36]. Therefore the tangent limits of (\mathbb{X}, \mathbb{T}) easily lift to \mathbb{X}^S , where in particular $\bar{\mathbb{T}}_2(A, S(A) \xrightarrow{\nu} A) := (\mathbb{T}_2(A), S\mathbb{T}_2(A) \xrightarrow{S(\rho_0)\lambda, S(\rho_1)\lambda} \mathbb{T}_2S(A) \xrightarrow{\mathbb{T}_2(\nu)} \mathbb{T}_2(A))$. Then we have that $\bar{\sigma}_{(A, \nu)} := \sigma_A$. It follows that $\bar{\mathbb{T}} := (\bar{\mathbb{T}}, \bar{p}, \bar{\sigma}, \bar{z}, \bar{\ell}, \bar{c})$ is a tangent structure on \mathbb{X}^S , which by definition is preserved strictly by the forgetful functor. We conclude that $(\mathbb{X}^S, \bar{\mathbb{T}})$ is a tangent category. ◀

The converse of Proposition 20 is also true, that is, if the Eilenberg-Moore category of a monad admits a tangent structure that is strictly preserved by the forgetful functor, then said monad is a tangent monad. In fact, in analogy with results for other kinds of distributive laws [45], tangent monads are in bijective correspondence with liftings of the tangent structure in this sense. Also note that by the universal property of the pullback there are distributive laws $\lambda_{n, M} : S\mathbb{T}_n(M) \rightarrow \mathbb{T}_nS(M)$ for each $n \in \mathbb{N}$.

To provide a tangent structure on the Eilenberg-Moore category of a codifferential category, we will define a tangent monad structure on the algebra modality itself. However we first need to address which tangent structure of the base category we will be lifting to the Eilenberg-Moore category. This is where finite biproducts come into play. Recall that a category with **finite biproducts** [36] can be described as an additive category with a zero object $0 = 1$ such that for each pair of objects A and B , there is an object $A \oplus B$ and maps $\iota_0 : A \rightarrow A \oplus B$, $\iota_1 : B \rightarrow A \oplus B$, $\pi_0 : A \oplus B \rightarrow A$, and $\pi_1 : A \oplus B \rightarrow B$, satisfying the well-known identities. This makes $A \oplus B$ both a product and a coproduct of A and B .

Every category \mathbb{X} with finite biproducts admits a tangent structure whose tangent functor is the diagonal functor, that is, the tangent functor \mathbb{T} is defined on objects as $\mathbb{T}(A) := A \oplus A$ and on maps as $\mathbb{T}(f) := f \oplus f$. The projection is $p_A := A \oplus A \xrightarrow{\pi_0} A$, the zero is $z_A := A \xrightarrow{\iota_0} A \oplus A$, the vertical lift is $\ell_A := A \oplus A \xrightarrow{\iota_0 \oplus \iota_1} A \oplus A \oplus A$, and the canonical flip is $c_A := A \oplus A \oplus A \oplus A \xrightarrow{1 \oplus \tau^{\oplus 1}} A \oplus A \oplus A \oplus A$ where $\tau_{A, B}^{\oplus} : A \oplus B \cong B \oplus A$ is the canonical symmetry isomorphism of the biproduct. Therefore it follows that for every $n \in \mathbb{N}$, $\mathbb{T}_n(A) := \bigoplus_{i=0}^n A$. For $n = 2$, $\mathbb{T}_2(A) := A \oplus A \oplus A$ and the addition map is $\sigma_A := A \oplus A \oplus A \xrightarrow{1 \oplus (\pi_0 + \pi_1)} A \oplus A$. We denote this tangent structure as \mathbb{B} (for biproduct). That (\mathbb{X}, \mathbb{B}) is a tangent category follows from that fact that every category with finite biproducts is in fact a Cartesian differential category (see Example 3.2).

► **Proposition 21.** *Let \mathbb{X} be a codifferential category with algebra modality $(S, \eta, \mu, \nabla, \mathbf{u})$ and deriving transformation \mathbf{d} , and suppose that \mathbb{X} admits finite biproducts \oplus . Define the natural transformation $\lambda_A : S(A \oplus A) \rightarrow S(A) \oplus S(A)$ as the unique map that makes the following*

diagram commute:

$$\begin{array}{ccccc}
 S(A \oplus A) & \xrightarrow{d_A} & S(A \oplus A) \otimes (A \oplus A) & \xrightarrow{S(\pi_0) \otimes \pi_1} & S(A) \otimes A & \xrightarrow{1_{S(A)} \otimes \eta_A} & S(A) \otimes S(A) \\
 \swarrow S(\pi_0) & & \downarrow \lambda_A & & & & \downarrow \nabla_A \\
 S(A) & \xleftarrow{\pi_0} & S(A) \oplus S(A) & \xrightarrow{\pi_1} & S(A) & & S(A)
 \end{array}$$

Equivalently, using the additive structure, $\lambda := S(\pi_0)\iota_0 + d(S(\pi_0) \otimes \pi_1)(1 \otimes \eta)\nabla\iota_1$. Then (S, μ, η, λ) is a tangent monad on (\mathbb{X}, \mathbb{B}) .

Proof. See the extended version of this paper [15]. ◀

An immediate consequence of Proposition 20 and Proposition 21 is that $(\mathbb{X}^S, \overline{\mathbb{B}})$ is a tangent category. Summarizing, we obtain one of the main results of this paper:

► **Theorem 22.** *The Eilenberg-Moore category of a codifferential category with finite bi-products is a tangent category.*

In particular, the result of applying the tangent functor of $(\mathbb{X}^S, \overline{\mathbb{B}})$ to an S -algebra can be simplified to $\overline{T}(A, \nu) := (A \oplus A, S(\pi_0)\nu\iota_0 + d(S(\pi_0) \otimes \pi_1)(\nu \otimes 1)\nabla\iota_1)$, which is an instance of the S -algebra structure defined in [9, Theorem 4.1]. Denote the S -algebra structure of $\overline{T}(A, \nu)$ as $\nu^b : S(A \oplus A) \rightarrow A \oplus A$. By [9, Proposition 5.4], the induced commutative monoid structure on $\overline{T}(A, \nu)$, generalizes that of the ring of dual numbers from Example 3.3:

$$\nabla^{\nu^b} = (\pi_0 \otimes \pi_0)\nabla^{\nu}\iota_0 + [(\pi_0 \otimes \pi_1) + (\pi_1 \otimes \pi_0)]\nabla^{\nu}\iota_1 \quad \mathbf{u}^{\nu^b} = \mathbf{u}^{\nu}\iota_0$$

Thus, the above tangent structure on the Eilenberg-Moore category of a codifferential category further highlights the relation between tangent structure and Weil algebras [32].

► **Example 23.** We conclude this section with some of the resulting tangent categories from our main examples of codifferential categories:

1. For a field k , when applying the constructions of this section to Example 18.1, one recovers precisely the tangent category from Example 3.3 induced by dual numbers, recalling that $\text{VEC}_k^{\text{Sym}} \cong \text{CALG}_k$.
2. For Example 18.2, the resulting tangent structure on the category of C^∞ -rings is particularly important, since the ring of dual numbers plays a key role in models of synthetic differential geometry based on C^∞ -rings [29, 39].
3. For a field k , one can also apply the constructions of this section to Example 16.3, which implies that the opposite category of cocommutative k -coalgebras is a tangent category.

6 Representable Tangent Structure and Differential Categories

The goal of this section is to show that the coEilenberg-Moore category of a differential category with biproducts is a tangent category, provided that it has certain equalizers. We will also explain that in the case of a differential storage category, the coEilenberg-Moore category is in fact a *representable* tangent category. To achieve this, we wish to apply Theorem 4 to the tangent structure that we constructed in the previous section, which resides on the Eilenberg-Moore category of a codifferential category.

In a category \mathbb{X} with finite biproducts, the pullback powers of the projection $p_A := \pi_0$ are $T_n(A) = \bigoplus_{i=0}^n A$. By the universality of the product and couniversality of the coproduct, each T_n is its own adjoint, that is, T_n is a left adjoint to T_n . However in the Eilenberg-Moore

category, \bar{T}_n is not necessarily its own adjoint (in fact it is far from it in any of the examples in this paper). Therefore we cannot use results about lifting adjunctions to Eilenberg-Moore category on the nose such as in [25]. Instead we employ Johnstone's left adjoint lifting theorem [27, Theorem 2], which is a special case of the adjoint lifting theorem of Butler that can be found in [2, Theorem 7.4], and it is at this point that we require the mild further assumption that the Eilenberg-Moore category admits reflexive coequalizers. First recall that a reflexive pair is a pair of parallel maps $f, g : A \rightarrow B$ with a common section, that is, there is a map $h : B \rightarrow A$ such that $hf = 1_B = hg$. A reflexive coequalizer is a coequalizer of a reflexive pair, and a category is said to have reflexive coequalizers if it has coequalizers of all reflexive pairs. A famous result of Linton's is that for a monad on a cocomplete category, the Eilenberg-Moore category is cocomplete if and only if the Eilenberg-Moore category admits all reflexive coequalizers [33].

► **Proposition 24.** [27, Theorem 2] *Let λ be a distributive law of a functor $R : \mathbb{X} \rightarrow \mathbb{X}$ over a monad (S, μ, η) , and suppose that R has a left adjoint L . If \mathbb{X}^S admits reflexive coequalizers then the lifting of R , $\bar{R} : \mathbb{X}^S \rightarrow \mathbb{X}^S$, has a left adjoint $G : \mathbb{X}^S \rightarrow \mathbb{X}^S$ such that $G(S(A), \mu_A) = (SL(A), \mu_{L(A)})$.*

Applying Proposition 24 to the Eilenberg-Moore category of a codifferential category, we obtain the following result:

► **Proposition 25.** *Let \mathbb{X} be a codifferential category with algebra modality $(S, \eta, \mu, \nabla, \mathbf{u})$ and deriving transformation \mathbf{d} , and suppose that \mathbb{X} admits finite biproducts and \mathbb{X}^S admits reflexive coequalizers. Then for each $n \in \mathbb{N}$, $\bar{T}_n : \mathbb{X}^S \rightarrow \mathbb{X}^S$ has a left adjoint.*

Applying Theorem 4 to the above proposition, we obtain the main result of this paper:

► **Theorem 26.** *If the coEilenberg-Moore category of a differential category with finite biproducts admits coreflexive equalizers (the dual of reflexive coequalizers), then the coEilenberg-Moore category is a tangent category.*

For differential storage categories, in order to show that the coEilenberg-Moore category is a representable tangent category, we will need to look at the construction of Section 5 for codifferential categories with comonoidal algebra modalities. So let \mathbb{X} be a codifferential category with a comonoidal algebra modality $(S, \mu, \eta, \nabla, \mathbf{u}, \mathbf{n}, \mathbf{n}_K)$ such that \mathbb{X} also admits finite biproducts, noting that \mathbf{n} and \mathbf{n}_K denote the comonoidal structure on S (Section 3). Note that in \mathbb{X} , it follows from distributivity between the biproduct and monoidal product (which is automatic in any additive symmetric monoidal category) that for every $n \in \mathbb{N}$ one has that $T_n(A) = \bigoplus_{i=0}^n A \cong \bigoplus_{i=0}^n (A \otimes K) \cong A \otimes \bigoplus_{i=0}^n K = A \otimes T_n(K)$. In particular when $n = 1$, $T(A) \cong A \otimes (K \oplus K)$. Recall that for a comonoidal algebra modality, \otimes is a coproduct in the Eilenberg-Moore category and there is a map $\mathbf{n}_K : S(K) \rightarrow K$ making (K, \mathbf{n}_K) into an S -algebra and an initial object. Then for every $n \in \mathbb{N}$ one has the following isomorphisms in the Eilenberg-Moore category \mathbb{X}^S : $\bar{T}_n(A, \nu) \cong (A, \nu) \otimes \bar{T}_n(K, \mathbf{n}_K)$, and in particular, for $n = 1$, $\bar{T}(A, \nu) \cong (A, \nu) \otimes \bar{T}(K, \mathbf{n}_K) = (A, \nu) \otimes (K \oplus K, \mathbf{n}_K^b)$. If \mathbb{X}^S admits reflexive coequalizers then by Proposition 25, each functor $\bar{T}_n \cong - \otimes \bar{T}_n(K, \mathbf{n}_K)$ has a left adjoint. Dualizing, this implies that $- \otimes \bar{T}_n(K, \mathbf{n}_K) : (\mathbb{X}^S)^{op} \rightarrow (\mathbb{X}^S)^{op}$ has a right adjoint and therefore that $\bar{T}_n(K, \mathbf{n}_K)$ is an exponent object in $(\mathbb{X}^S)^{op}$. Therefore, in view of Theorem 26, $(\mathbb{X}^S)^{op}$ is a representable tangent category whose infinitesimal object is $\bar{T}(K, \mathbf{n}_K) = (K \oplus K, \mathbf{n}_K^b)$.

Let us restate this result in terms of differential storage categories. Let \mathbb{X} be a differential storage category with coalgebra modality $(!, \delta, \varepsilon, \Delta, \mathbf{e})$ equipped with deriving transformation $\mathbf{d}_A : !(A) \otimes A \rightarrow !(A)$ and such that \mathbb{X} has finite biproducts \oplus . Recall that the monoidal

structure on $!$ includes a map $m_K : K \rightarrow !(K)$ that makes (K, m_K) into a $!$ -coalgebra. If $\mathbb{X}^!$ admits coreflexive equalizers, then $\mathbb{X}^!$ is a representable tangent category whose infinitesimal object is $(K \oplus K, m_K^\sharp)$, where $m_K^\sharp : K \oplus K \rightarrow !(K \oplus K)$ is defined as the unique map that makes the following diagram commute (using the couniversal property of the coproduct):

$$\begin{array}{ccccc}
 K & \xrightarrow{\iota_0} & K \oplus K & \xleftarrow{\iota_1} & K & \xrightarrow{\cong} & K \otimes K \\
 \downarrow m_K & & \downarrow m_K^\sharp & & & & \downarrow m_K \otimes 1_K \\
 & & & & & & !(K) \otimes K \\
 & & & & & & \downarrow !(\iota_0) \otimes \iota_1 \\
 !(K) & \xrightarrow{!(\iota_0)} & !(K \oplus K) & \xleftarrow{d_{K \oplus K}} & !(K \oplus K) \otimes (K \oplus K) & &
 \end{array}$$

We summarize this result for differential storage categories to obtain the final main result of this paper:

► **Theorem 27.** *If the coEilenberg-Moore category of a differential storage category admits coreflexive equalizers, then the coEilenberg-Moore category is a representable tangent category.*

► **Example 28.** We conclude this section by looking briefly at some examples.

1. For a field k and Example 18.1, recall once again that $\text{VEC}_k^{\text{Sym}} \cong \text{CALG}_k$. It is well known that CALG_k is complete and cocomplete, and therefore $\text{VEC}_k^{\text{Sym}}$ admits reflexive coequalizers. Applying Theorem 27 to this example, one obtains precisely the tangent structure on $\text{CALG}_k^{\text{op}}$ from Example 7.3 (and described in full detail in [12, Proposition 5.16]), where we recall that the infinitesimal object is the ring of dual numbers $k[\epsilon]$. It is interesting to note that for polynomial rings $k[X]$ we have in $\text{CALG}_k^{\text{op}}$ that $k[X]^{k[\epsilon]} \cong k[X] \otimes k[X]$. In [35, 34], the third author generalized the tangent structure on the opposite of the category of commutative k -algebras to the setting of certain codifferential categories, using universal derivations [9], which generalize Kähler differentials.
2. Similarly for Example 16.3 (again for a field k) recall that $\text{VEC}_k^!$ is isomorphic to the category of cocommutative k -coalgebras, which is both complete and cocomplete [40, 1]. Therefore, $\text{VEC}_k^!$ admits coreflexive equalizers and so by applying Theorem 27, $\text{VEC}_k^!$ is a representable tangent category. Most interesting is that the infinitesimal object in this case is again the ring of dual numbers $k[\epsilon]$ but seen as a cocommutative k -coalgebra with comultiplication defined on the basis elements as $1 \mapsto 1 \otimes 1$ and $\epsilon \mapsto 1 \otimes \epsilon + \epsilon \otimes 1$. The coalgebra $k[\epsilon]$ played an important role in [11].
3. For C^∞ -rings and Example 18.2, the Eilenberg-Moore category is the category of C^∞ -rings. As it is the category of algebras for a Lawvere theory, the category of C^∞ -rings is both complete and complete and therefore, in particular, admits reflexive coequalizers. Hence it follows that the opposite of the category of C^∞ -rings is a tangent category. In particular, for a smooth manifold M , the image of $C^\infty(M)$ under the tangent functor in this case is precisely $C^\infty(\mathbb{T}(M))$, where $\mathbb{T}(M)$ is the standard tangent bundle over M .

7 Conclusion

Given that differential categories involve a comonad it seems obvious from a categorical perspective that one should consider the coEilenberg-Moore category of coalgebras. That these coEilenberg-Moore categories are tangent categories, assuming the existence of coreflexive equalizers, provides an important way of generating tangent categories that is already “baked in” to algebraic geometry and synthetic differential geometry. As there are many examples of differential categories from various fields, this opens the door to studying new and interesting tangent categories.

However, viewing these structures at this level of generality raises a number of further questions. Indeed, the fact that one has a tangent category begs the question of how various devices from abstract differential geometry (such as vector fields, Lie algebras, connections, solutions to differential equations, etc.) manifest in these settings. For example when and how do “curve objects” and “line objects” appear in these settings? In any tangent category, one can also consider *differential objects*, which are objects A such that $\mathbb{T}(A) \cong A \times A$. Can one characterize the differential objects in a coEilenberg-Moore category of a differential category? The cofree !-coalgebras $(!(A), \delta_A)$ are always differential objects, but when are these *exactly* the differential objects?

There is now much work to be done to examine the more detailed ramifications of the constructions in this paper and to place specific results in a more general geography.

References

- 1 A Agore. Limits of coalgebras, bialgebras and Hopf algebras. *Proceedings of the American Mathematical Society*, 139(3):855–863, 2011.
- 2 Michael Barr and Charles Wells. Toposes, triples and theories. *Repr. Theory Appl. Categ.*, pages x+288, 2005. Corrected reprint of the 1985 original.
- 3 Gavin M Bierman. What is a categorical model of intuitionistic linear logic? In *International Conference on Typed Lambda Calculi and Applications*, pages 78–93. Springer, 1995.
- 4 R Blute, J R B Cockett, and Robert A G Seely. Cartesian differential storage categories. *Theory and Applications of Categories*, 30(18):620–686, 2015.
- 5 R. F. Blute, J. R. B. Cockett, J.-S. P. Lemay, and R. A. G. Seely. Differential Categories Revisited. *Applied Categorical Structures*, 2019. doi:10.1007/s10485-019-09572-y.
- 6 RF Blute, J Robin B Cockett, and Robert A G Seely. Cartesian differential categories. *Theory and Applications of Categories*, 22(23):622–672, 2009.
- 7 Richard F Blute, J Robin B Cockett, and Robert A G Seely. Differential categories. *Mathematical Structures in Computer Science*, 16(06):1049–1083, 2006.
- 8 Richard F Blute, Thomas Ehrhard, and Christine Tasson. A convenient differential category. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, LIII:211–232, 2012.
- 9 Richard F Blute, Rory B B Lucyshyn-Wright, and Keith O’Neill. Derivations in codifferential categories. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 57:243–280, 2016.
- 10 Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. Categorical models for simply typed resource calculi. *Electronic Notes in Theoretical Computer Science*, 265:213–230, 2010.
- 11 James Clift and Daniel Murfet. Cofree coalgebras and differential linear logic. *arXiv preprint arXiv:1701.01285*, 2017.
- 12 J Robin B Cockett and Geoff S H Cruttwell. Differential Structure, Tangent Structure, and SDG. *Applied Categorical Structures*, 22(2):331–417, 2014.
- 13 J Robin B Cockett and Geoff S H Cruttwell. Differential Bundles and Fibrations for Tangent Categories. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 59:10–92, 2018.
- 14 J Robin B Cockett, Geoff S H Cruttwell, and Jonathan D Gallagher. Differential restriction categories. *Theory and Applications of Categories*, 25(21):537–613, 2011.
- 15 Robin Cockett, Jean-Simon Pacaud Lemay, and Rory B. B. Lucyshyn-Wright. Tangent Categories from the Coalgebras of Differential Categories, 2019. arXiv:1910.05617.
- 16 G S H Cruttwell, J-S P Lemay, and R B B Lucyshyn-Wright. Integral and differential structure on the free C^∞ -ring modality. *arXiv preprint arXiv:1902.04555*, 2019.
- 17 Thomas Ehrhard. On Köthe sequence spaces and linear logic. *Mathematical Structures in Computer Science*, 12(5):579–623, 2002.
- 18 Thomas Ehrhard. An introduction to Differential Linear Logic: proof-nets, models and antiderivatives. *Mathematical Structures in Computer Science*, 28:995–1060, 2018.

- 19 M.P. Fiore. Differential structure in models of multiplicative biadditive intuitionistic linear logic. In *International Conference on Typed Lambda Calculi and Applications*, pages 163–177. Springer, 2007.
- 20 Richard Garner. An embedding theorem for tangent categories. *Advances in Mathematics*, 323:668–687, 2018.
- 21 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.
- 22 Jonathan S Golan. *Semirings and their Applications*. Springer Science & Business Media, 2013.
- 23 Li Guo. *An introduction to Rota-Baxter algebra*, volume 2. International Press Somerville, 2012.
- 24 Robin Hartshorne. *Algebraic geometry*, volume 52. Springer Science & Business Media, 2013.
- 25 Claudio Hermida and Bart Jacobs. Structural induction and coinduction in a fibrational setting. *Information and Computation*, 145(2):107–152, 1998.
- 26 Martin Hyland and Andrea Schalk. Glueing and orthogonality for models of linear logic. *Theoretical Computer Science*, 294(1-2):183–231, 2003.
- 27 P T Johnstone. Adjoint lifting theorems for categories of algebras. *The Bulletin of the London Mathematical Society*, 7(3):294–297, 1975.
- 28 Gerd Kainz, Andreas Kriegl, and P Michor. C^∞ -algebras from the functional analytic view point. *Journal of Pure and Applied Algebra*, 46(1):89–107, 1987.
- 29 Anders Kock. *Synthetic differential geometry*, volume 333. Cambridge University Press, 2006.
- 30 Serge Lang. Algebra, revised 3rd ed. *Graduate Texts in Mathematics*, 211, 2002.
- 31 Jeffrey M Lee. *Manifolds and Differential Geometry*, volume 107. American Mathematical Soc., 2009.
- 32 Poon Leung. Classifying tangent structures using Weil algebras. *Theory and Applications of Categories*, 32(9):286–337, 2017.
- 33 Fred EJ Linton. Coequalizers in categories of algebras. In *Seminar on triples and categorical homology theory*, pages 75–90. Springer, 1969.
- 34 Rory B B Lucyshyn-Wright. Categories with representable tangent structure and models of SDG associated to codifferential categories. *Foundational Methods in Computer Science*, 2014.
- 35 Rory B B Lucyshyn-Wright. Models of synthetic differential geometry associated to codifferential categories. *Category Theory 2014 conference*, Cambridge, UK, 2014.
- 36 Saunders Mac Lane. *Categories for the working mathematician*. Springer-Verlag, New York, Berlin, Heidelberg, 1971, revised 2013.
- 37 Giulio Manzonetto. What is a Categorical Model of the Differential and the Resource λ -Calculi? *Mathematical Structures in Computer Science*, 22(3):451–520, 2012.
- 38 Paul-André Mellies. Categorical semantics of linear logic. *Panoramas et Syntheses*, 27:15–215, 2009.
- 39 Ieke Moerdijk and Gonzalo E Reyes. *Models for smooth infinitesimal analysis*. Springer Science & Business Media, 2013.
- 40 Hans-E Porst. ON CORINGS AND COMODULES. *Archivum Mathematicum*, 42(4), 2006.
- 41 J Rosický. Abstract tangent functors. *Diagrammes*, 12(3):1–11, 1984.
- 42 Andrea Schalk. What is a categorical model of linear logic. *Manuscript, available from <http://www.cs.man.ac.uk/~schalk/work.html>*, 2004.
- 43 Robbert A G Seely. *Linear logic, *-autonomous categories and cofree coalgebras*, volume 92. American Mathematical Society, 1989.
- 44 Moss E Sweedler. *Hopf Algebras*. W. A. Benjamin, Inc., 1969.
- 45 Robert Wisbauer. Algebras versus coalgebras. *Applied Categorical Structures*, 16(1):255–295, 2008.

Reverse Derivative Categories*

Robin Cockett

University of Calgary,
Department of Computer Science, Canada
robin@ucalgary.ca

Jonathan Gallagher

Dalhousie University, Department of
Mathematics and Statistics, Canada
jonathan.gallagher@dal.ca

Benjamin MacAdam

University of Calgary,
Department of Computer Science, Canada
benjamin.macadam@ucalgary.ca

Dorette Pronk

Dalhousie University, Department of
Mathematics and Statistics, Canada
dorette.pronk@dal.ca

Geoffrey Cruttwell

Mount Allison University, Department of
Mathematics and Computer Science, Canada
gcruttwell@mta.ca

Jean-Simon Pacaud Lemay

University of Oxford,
Department of Computer Science, UK
jean-simon.lemay@kellogg.ox.ac.uk

Gordon Plotkin

Google Research
gdp@inf.ed.ac.uk

Abstract

The reverse derivative is a fundamental operation in machine learning and automatic differentiation [1, 12]. This paper gives a direct axiomatization of a category with a reverse derivative operation, in a similar style to that given by [2] for a forward derivative. Intriguingly, a category with a reverse derivative also has a forward derivative, but the converse is not true. In fact, we show explicitly what a forward derivative is missing: a reverse derivative is equivalent to a forward derivative with a dagger structure on its subcategory of linear maps. Furthermore, we show that these linear maps form an additively enriched category with dagger biproducts.

2012 ACM Subject Classification Theory of computation \rightarrow Semantics and reasoning; Theory of computation \rightarrow Program semantics

Keywords and phrases Reverse Derivatives, Cartesian Reverse Differential Categories, Categorical Semantics, Cartesian Differential Categories, Dagger Categories, Automatic Differentiation

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.18

Related Version A full version of the paper is available at <https://arxiv.org/abs/1910.07065>.

Funding *Robin Cockett*: Partially supported by NSERC (Canada).

Geoffrey Cruttwell: Partially supported by NSERC (Canada).

Jonathan Gallagher: Supported in part by NSERC and AARMS (Canada).

Jean-Simon Pacaud Lemay: Supported by Kellogg College, the Clarendon Fund, and the Oxford Google-DeepMind Graduate Scholarship (UK).

Benjamin MacAdam: Partially supported by NSERC (Canada).

Gordon Plotkin: Supported by ESPRC (UK).

Dorette Pronk: Partially supported by NSERC (Canada).

Acknowledgements We thank Robert Seely for participating in the discussion on reverse differentiation with us.

* This paper is the result of a joint working session that the authors participated in at the Foundational Methods in Computer Science workshop in June, 2019.



1 Introduction

The use of derivatives and differentiation in programming and machine learning is becoming ubiquitous. As a result, there has been an increased interest in axiomatic setups for differentiation; in particular, categorical models for differentiation have become more central. There are two types of derivative operations used in programming: the *forward* derivative and the *reverse* derivative. From the programmer's perspective, it is much more common for the reverse derivative to play the central role due to its increased efficiency and improved accuracy when computing with functions from \mathbb{R}^n to \mathbb{R} (due to the so called cheap gradient principle). The importance of this principle was already recognized by Linnainmaa in 1976 [16] and was specifically used for back-propagation in multi-layer networks and deep learning. This was further spelled out in detail in [18]. Also, Tensorflow, Google's new interface for expressing machine learning algorithms, uses the reverse mode of automatic differentiation as the basic building block minimizing cost functions [1].

The categorical approaches to differentiation to date have all exclusively focused on the abstract properties of the forward derivative [2]. This thus leaves a significant gap which needs to be filled: an axiomatic categorical setting for reverse differentiation. The main goal of this paper is to introduce such a structure and explore some of its properties and consequences.

A “Cartesian reverse differential category” (a category equipped with a reverse derivative operation as introduced in this paper) is already a Cartesian differential category (the standard axiomatics for a category with a forward derivative). We show that a category equipped with a reverse derivative also has a forward derivative (i.e., it has a Cartesian differential structure). Moreover, a reverse differential category has a fibered dagger structure on its subcategory of linear maps, a structure which does not automatically exist in a Cartesian differential category. Suitably axiomatized, we show that having such a dagger structure is enough to ensure that a Cartesian differential category structure gives a reverse differential category. These results provide a starting point to build categorical semantics of differential programming languages [17], as they provide axiomatically enough structure to handle both forward and reverse derivatives.

The paper is structured as follows. In section 2, we recall the basic notation and definitions of a Cartesian differential category (“a category equipped with a forward derivative”). We do this first to acclimatize the reader to the general style of this categorical definition, and to recall the structure of Cartesian left additive categories, which are necessary to define both forward and reverse differential categories. In section 3, we introduce our definition of a reverse differential category. We explore some of the important consequences of the definition noted above: (a) Cartesian differential structure, (b) how to define and work with linear maps in this setting, and (c) a dagger structure on the linear maps. In section 4, we show how to go back: given a Cartesian differential category with a “contextual dagger”, we build a Cartesian reverse differential category. There is much more work to be done with this structure and these ideas: in section 5, we describe some of the ways in which this work can be extended, including allowing partial functions.

As far as we are aware, this paper represents the first categorical axiomatization of the reverse derivative. However, [11] does have some related ideas. There, the relationship between the reverse derivative and coproducts was noticed, and the author specified an internal category which satisfies some of the axioms of a Cartesian differential category in a functional programming language. This work expands that observation by developing the dagger biproduct structure using the reverse derivative and relating this to the dual of the simple slice fibration.

Full proofs of the theorems in this paper can be found in an expanded version [10], and we have indicated in this version, the corresponding theorem number in the longer version.

2 Forward derivatives

The standard setting for a “category with a forward derivative” is a Cartesian differential category, first introduced in [2]. Following that paper, we write composition in diagrammatic order, so that f , followed by g , is written as fg .

2.1 Cartesian left additive categories

A Cartesian differential category first consists of a Cartesian left additive category, and so we begin by recalling this notion. Recall that a category \mathbb{X} is said to be **Cartesian** when there are chosen binary products \times , with projection maps π_i and pairing operation $\langle -, - \rangle$, and a chosen terminal object $\mathbf{1}$, with unique maps $!$ to the terminal object.

► **Definition 1.** *A left additive category [2, Definition 1.1.1] is a category \mathbb{X} such that each hom-set is a commutative monoid, with addition operation $+$ and zero maps 0 , such that composition on the left preserves the additive structure in the sense that $x(f + g) = xf + xg$ and $x0 = 0$. Maps h which preserve the additive structure by composition on the right ($(x + y)h = xh + yh$ and $0h = 0$) are called **additive**. A **Cartesian left additive category** [2, Definition 1.2.1] is a left additive category \mathbb{X} which is Cartesian and such that all projection maps π_i are additive¹.*

Cartesian left additive categories can alternatively be defined as Cartesian categories in which each object A canonically bears the structure of a commutative monoid with addition $+_A : A \times A \rightarrow A$ and zero $0_A : \mathbf{1} \rightarrow A$.

► **Example 2.** Here are examples of Cartesian left additive categories that we will consider throughout this paper:

1. Any category with finite biproducts is a Cartesian left additive category where every map is additive. And conversely, in a Cartesian left additive category where every map is additive, the finite product is a finite biproduct [2, Proposition 1.2.2].
2. Let R be a commutative rig (also known as a commutative semiring). Let POLY_R be the category of polynomials with coefficients in R ; that is, the category whose objects are the natural numbers $n \in \mathbb{N}$ and where a map $n \xrightarrow{P} m$ is an m -tuple of polynomials $P := \langle p_1(\vec{x}), \dots, p_m(\vec{x}) \rangle$, where $p_i(\vec{x}) \in R[x_1, \dots, x_n]$ (the polynomial ring in n -variables over R). POLY_R is a Cartesian left additive category where composition is given by the standard composition of polynomials, the product on objects is given by the sum of natural numbers, and the additive structure is given by the sum of polynomials.
3. Let \mathbb{R} be the set of real numbers and let **Smooth** be the category of smooth real functions, that is, the category whose objects are again the natural numbers $n \in \mathbb{N}$ and where a map $n \xrightarrow{F} m$ is a smooth function $\mathbb{R}^n \xrightarrow{F} \mathbb{R}^m$. **Smooth** is a Cartesian left additive category where composition is given by the standard composition of smooth functions, the product on objects is given by the sum of natural numbers, and the additive structure is given by the sum of smooth functions. Note that a smooth map $\mathbb{R}^n \xrightarrow{F} \mathbb{R}^m$ is actually an m -tuple of smooth functions $F = \langle f_1, \dots, f_m \rangle$, where $\mathbb{R}^n \xrightarrow{f_i} \mathbb{R}$ and therefore $\text{POLY}_{\mathbb{R}}$ is a sub-Cartesian left additive category of **Smooth**.

¹ Note that this is a slight variation on the definition of a Cartesian left additive category found in [2], but it is indeed equivalent.

18:4 Reverse Derivative Categories

As not every map in a Cartesian left additive category is additive, the product \times is not a coproduct, thus is *not* a biproduct. However, it is still possible to define injection maps. So in a Cartesian left additive category, define $\iota_0 := \langle 1, 0 \rangle : A \rightarrow A \times B$ and $\iota_1 := \langle 0, 1 \rangle : B \rightarrow A \times B$. For maps $A \xrightarrow{f} C$ and $B \xrightarrow{g} C$, we define $\langle f|g \rangle := \pi_0 f + \pi_1 g : A \times B \rightarrow C$, and finally for maps $A \xrightarrow{h} B$ and $C \xrightarrow{k} D$ we write $h \oplus k := \langle h\iota_0|k\iota_1 \rangle : A \times B \rightarrow C \times D$. Although this notation is suggestive, we again stress this is not part of a coproduct or biproduct structure. However, in what follows we will define the category of linear maps where the above will witness a biproduct structure on that category. We leave the following lemma as an easy exercise to the reader:

► **Lemma 3.** *In a Cartesian left additive category, $f\iota_0 + g\iota_1 = \langle f, g \rangle$ and $h \oplus k = h \times k$.*

2.2 Cartesian differential categories

This section reviews Cartesian differential categories which provide the semantics for forward differentiation [2].

► **Definition 4.** *A Cartesian differential category [2] is a Cartesian left additive category with a combinator D , called the **differential combinator**, which written as an inference rule is given by:*

$$\frac{A \xrightarrow{f} B}{A \times A \xrightarrow{D[f]} B}$$

where $D[f]$ is called the derivative of f , and such that the following equalities hold²:

[CDC.1] $D[f + g] = D[f] + D[g]$ and $D[0] = 0$;

[CDC.2] $\langle a, b + c \rangle D[f] = \langle a, b \rangle D[f] + \langle a, c \rangle D[f]$ and $\langle a, 0 \rangle D[f] = 0$;

[CDC.3] $D[1] = \pi_1$, $D[\pi_0] = \pi_1\pi_0$, and $D[\pi_1] = \pi_1\pi_1$;

[CDC.4] $D[\langle f, g \rangle] = \langle D[f], D[g] \rangle$;

[CDC.5] $D[\langle f|g \rangle] = \langle \pi_0 f, D[f] \rangle D[g]$;

[CDC.6] $\langle \langle a, b \rangle, \langle 0, c \rangle \rangle D[D[f]] = \langle a, c \rangle D[f]$;

[CDC.7] $\langle \langle a, b \rangle, \langle c, d \rangle \rangle D[D[f]] = \langle \langle a, c \rangle, \langle b, d \rangle \rangle D[D[f]]$.

For an in-depth commentary on these axioms, we invite the reader to see the original Cartesian differential category paper [2]. Briefly, **[CDC.1]** is that the derivative of a sum is the sum of the derivatives, **[CDC.2]** states that derivatives are additive in their second argument, **[CDC.3]** says that the identity and projection maps are linear (more on what this means soon), **[CDC.4]** is that the derivative of a pairing is the pairing of the derivatives, **[CDC.5]** is the famous chain rule, **[CDC.6]** says that the derivative is linear in its second argument, and finally **[CDC.7]** is the symmetry of the mixed partial derivatives.

► **Example 5.** Here are some well-known examples of Cartesian differential categories.

1. Every category with finite biproducts is a Cartesian differential category where for a map $A \xrightarrow{f} B$, its derivative $A \oplus A \xrightarrow{D[f]} B$ is defined as $D[f] := A \oplus A \xrightarrow{\pi_1} A \xrightarrow{f} B$.

² Note that the order of variables is different here than in [2]; here, we write the vector variable in the second component, as this more closely aligns with standard differential calculus notation.

2. Let R be a commutative rig. POLY_R is a Cartesian differential category whose differential combinator is given by the standard differentiation of polynomials. By [CDC.4], since every map in POLY_R is a tuple, it is sufficient to define the derivative of maps $n \xrightarrow{p} 1$, which are polynomials $p(\vec{x}) \in R[x_1, \dots, x_n]$. Then its derivative $n \times n \xrightarrow{D[p]} 1$, viewed as polynomials $D[p](\vec{x}, \vec{y}) \in R[x_1, \dots, x_n, y_1, \dots, y_n]$, is defined by the sum of partial derivatives of $p(\vec{x})$:

$$D[p](\vec{x}, \vec{y}) := \sum_{i=1}^n \frac{\partial p}{\partial x_i}(\vec{x}) y_i$$

For example, consider the polynomial $p(x_1, x_2) = x_1^2 + 3x_1x_2 + 5x_2$, so $2 \xrightarrow{p} 1$, then $4 \xrightarrow{D[p]} 1$ is $D[p](x_1, x_2, y_1, y_2) = (2x_1 + 3x_2)y_1 + (3x_1 + 5)y_2$. On the other hand, for a map $n \xrightarrow{P} m$, which is a tuple $P := \langle p_1(\vec{x}), \dots, p_m(\vec{x}) \rangle$, its derivative is the tuple $D[P] := \langle D[p_1](\vec{x}, \vec{y}), \dots, D[p_m](\vec{x}, \vec{y}) \rangle$.

3. The category Smooth is a Cartesian differential category where for a map $n \xrightarrow{F} m$, which is a smooth function $\mathbb{R}^n \xrightarrow{F} \mathbb{R}^m$, its derivative $\mathbb{R}^n \times \mathbb{R}^n \xrightarrow{D[F]} \mathbb{R}^m$ is defined as

$$D[F](\vec{x}, \vec{v}) := J_F(\vec{x}) \cdot v$$

where $J_F(\vec{x})$ is the Jacobian of F at \vec{x} and where \cdot is matrix multiplication. Of course, similar to the previous example, as every F can be viewed as a tuple, by [CDC.4], it would have also been sufficient to define the differential combinator for smooth maps $\mathbb{R}^n \xrightarrow{f} \mathbb{R}$. In this case, $J_f(x)$ is better known as the gradient of f , $\nabla(f)(x) := \langle \frac{\partial f}{\partial x_1}(\vec{x}), \dots, \frac{\partial f}{\partial x_n}(\vec{x}) \rangle$, and so $\mathbb{R}^n \times \mathbb{R}^n \xrightarrow{D[F]} \mathbb{R}$ is:

$$D[f](\vec{x}, \vec{y}) := \nabla(f)(\vec{x}) \cdot \vec{y} = \sum_{i=1}^n \frac{\partial f}{\partial x_i}(\vec{x}) y_i$$

This clearly shows that $\text{POLY}_{\mathbb{R}}$ is a sub-Cartesian differential category of Smooth .

We now provide a few lemmas that give alternative views on the axioms of a Cartesian differential category; these will be helpful when comparing this structure to a reverse differential category. Note that while the first lemma shows that [CDC.4] is actually redundant, to keep the numbering of the equations consistent with past literature on Cartesian differential categories, we chose to include it in the definition.

► **Lemma 6.** [15, Lemma 2.8] *In a Cartesian differential category, [CDC.4] is redundant.*

► **Lemma 7.** [9, Proposition 4.2] *In a Cartesian left additive category:*

1. *If a combinator D satisfies [CDC.1-5,7], the axiom [CDC.6] is equivalent to:*

$$\langle 1 \times \pi_0, 0 \times \pi_1 \rangle D[D[f]] = (1 \times \pi_1)D[f]$$

2. *If a combinator D satisfies [CDC.1-6], the axiom [CDC.7] is equivalent to*

$$\text{ex}D[D[f]] = D[D[f]]$$

where $\text{ex} : (A \times B) \times (C \times D) \rightarrow (A \times C) \times (B \times D)$ is the **exchange** natural isomorphism defined as $\text{ex} := \langle \pi_0 \times \pi_0, \pi_1 \times \pi_1 \rangle$.

18:6 Reverse Derivative Categories

In a Cartesian differential category, there are two important notions: that of partial derivatives and that of linear maps. Beginning with partial derivatives, if $A \times B \xrightarrow{f} C$ then the partial derivative of f with respect to B is defined as follows:

$$D_B[f] := A \times (B \times B) \xrightarrow{\langle 1 \times \pi_0, 0 \times \pi_1 \rangle} (A \times B) \times (A \times B) \xrightarrow{D[f]} C$$

This partial derivative definition induces a Cartesian differential category on the simple slice categories. Recall that the simple slice category of \mathbb{X} with respect to A , denoted $\mathbb{X}[A]$, is the category with the same objects as \mathbb{X} and where a map from $B \rightarrow C$ in $\mathbb{X}[A]$ is a map $f : A \times B \rightarrow C$ in \mathbb{X} ; that is, in terms of homsets, $\mathbb{X}[A](B, C) = \mathbb{X}(A \times B, C)$, and composition of is given by $\langle f, \pi_1 \rangle g$.

► **Proposition 8.** [2, Corollary 4.5.2] *Let \mathbb{X} be a Cartesian differential category and A any object. Then $\mathbb{X}[A]$ is a Cartesian differential category and the derivative of $f : A \times B \rightarrow C$ is $D_B[f]$.*

Linear maps play a central role in the theory of Cartesian differential categories.

► **Definition 9.** *A map f in a Cartesian differential category is **linear** when $D[f] = \pi_1 f$. Similarly, a map $A \times B \xrightarrow{f} C$ is **linear in B** if the following diagram commutes:*

$$\begin{array}{ccc} A \times (B \times B) & \xrightarrow{D_B[f]} & C \\ & \searrow 1 \times \pi_1 & \nearrow f \\ & A \times B & \end{array}$$

Note that a map $A \times B \xrightarrow{f} C$ is linear in B if and only if when regarded as a map $B \xrightarrow{f} C$ in $\mathbb{X}[A]$, it is linear with respect to the derivative in $\mathbb{X}[A]$.

► **Example 10.** Let us consider the linear maps in our examples of Cartesian differential categories from Example 5:

1. In a category with finite biproducts, every map is linear by definition of the differential combinator.
2. Let R be a commutative rig. In POLY_R , a map $n \xrightarrow{p} 1$ is linear if and only if $p(\vec{x}) = \sum_{i=1}^n r_i x_i$ for some $r_i \in R$. And it follows that $n \xrightarrow{P} m$, with $P = \langle p_1(\vec{x}), \dots, p_n(\vec{x}) \rangle$, is linear if and only if each $p_i(\vec{x})$ is. In other words, $n \xrightarrow{P} m$ is linear in the Cartesian differential category sense if and only if it induces an R -linear map $R^n \rightarrow R^m$.
3. Similar to the previous example, in Smooth the linear maps in the Cartesian differential category sense are precisely the linear maps in the ordinary sense. Explicitly, $n \xrightarrow{F} m$ is linear if and only if $\mathbb{R}^n \xrightarrow{F} \mathbb{R}^m$ is a linear transformation.

For a Cartesian differential category \mathbb{X} , we can also form its subcategory of linear maps $\text{Lin}(\mathbb{X})$, and since every linear map is additive [2], it follows that:

► **Proposition 11** ([2, Corollary 2.2.3]). *For a Cartesian differential category \mathbb{X} , its subcategory of linear maps $\text{Lin}(\mathbb{X})$ has finite biproducts.*

Finally, we conclude this section with the observation that linearity can also be expressed in terms of injection maps:

► **Lemma 12.** *In a Cartesian differential category,*

- *A map $A \xrightarrow{f} B$ is linear if and only if $\iota_1 D[f] = f$.*
- *A map $A \times B \xrightarrow{f} C$ is linear in B if and only if $(\iota_0 \times \iota_1) D[f] = f$.*

3 Reverse derivatives

In this section we introduce our definition of a Cartesian reverse differential category. The types of axioms are similar to those for Cartesian differential categories; however, after the first two, the forms the axioms take are quite different.

► **Definition 13.** A Cartesian left additive category \mathbb{X} has **reverse derivatives** in case there is a combinator R , called the **reverse differential combinator**, which written as an inference rule is given by:

$$\frac{A \xrightarrow{f} B}{A \times B \xrightarrow{R[f]} A}$$

where $R[f]$ is called the reverse derivative of f , and such that the following coherences are satisfied:

[RD.1] $R[f + g] = R[f] + R[g]$ and $R[0] = 0$;

[RD.2] $\langle a, b + c \rangle R[f] = \langle a, b \rangle R[f] + \langle a, c \rangle R[f]$ and $\langle a, 0 \rangle R[f] = 0$

[RD.3] $R[1] = \pi_1$, while for the projections, the following diagrams commute:

$$\frac{A \times B \xrightarrow{\pi_0} A}{(A \times B) \times A \xrightarrow{R[\pi_0]} A \times B} \quad \begin{array}{ccc} (A \times B) \times A & \xrightarrow{R[\pi_0]} & A \times B \\ & \searrow \pi_1 & \nearrow \iota_0 \\ & A & \end{array}$$

$$\frac{A \times B \xrightarrow{\pi_1} B}{(A \times B) \times B \xrightarrow{R[\pi_1]} A \times B} \quad \begin{array}{ccc} (A \times B) \times B & \xrightarrow{R[\pi_1]} & A \times B \\ & \searrow \pi_1 & \nearrow \iota_1 \\ & B & \end{array}$$

[RD.4] For a tupling of maps f and g , the following equality holds:

$$\frac{A \xrightarrow{f} B}{A \times B \xrightarrow{R[f]} A} \quad \frac{A \xrightarrow{g} C}{A \times C \xrightarrow{R[g]} A} \quad \frac{A \xrightarrow{\langle f, g \rangle} B \times C}{A \times (B \times C) \xrightarrow{R[\langle f, g \rangle]} A}$$

$$R[\langle f, g \rangle] = (1 \times \pi_0)R[f] + (1 \times \pi_1)R[g]$$

While for the unique map to the terminal object: $!_A : A \rightarrow \mathbf{1}$, the following equality holds:

$$R[!_A] = 0$$

[RD.5] For composable maps f and g , the following diagram commutes:

$$\frac{A \xrightarrow{f} B}{A \times B \xrightarrow{R[f]} A} \quad \frac{B \xrightarrow{g} C}{B \times C \xrightarrow{R[g]} B} \quad \frac{A \xrightarrow{fg} C}{A \times C \xrightarrow{R[fg]} A}$$

$$\begin{array}{ccc} A \times C & \xrightarrow{R[fg]} & A \\ \langle \pi_0, \langle \pi_0 f, \pi_1 \rangle \rangle \downarrow & & \uparrow R[f] \\ A \times (B \times C) & \xrightarrow{1 \times R[g]} & A \times B \end{array}$$

[RD.6] $\langle 1 \times \pi_0, 0 \times \pi_1 \rangle (\iota_0 \times 1)R[R[R[f]]]\pi_1 = (1 \times \pi_1)R[f]$

[RD.7] $(\iota_0 \times 1)R[R[(\iota_0 \times 1)R[R[f]]\pi_1]]\pi_1 = \text{ex}(\iota_0 \times 1)R[R[(\iota_0 \times 1)R[R[f]]\pi_1]]\pi_1$

A **Cartesian reverse differential category** is a Cartesian left additive category with a reverse differential combinator.

The axioms of the reverse differential combinator mirror those of a differential combinator. **[RD.1]** states that the reverse derivative of a sum is the sum of the reverse derivatives while **[RD.2]** says that the reverse derivative is additive in its second argument. **[RD.3]** and **[RD.4]** respectively explain what the reverse derivatives of the identity, projection, and tuples are. **[RD.5]** is the reverse derivative version of the chain rule. Lastly, **[RD.6]** expresses that the reverse derivative is linear in its second argument and **[RD.7]** gives the symmetry of the mixed partial reverse derivatives.

► **Example 14.** Here are some examples of reverse differential categories:

1. Let R be a commutative rig. POLY_R is a reverse differential category whose reverse differential combinator R is again defined using partial derivatives of polynomials. For a map $n \xrightarrow{P} m$, $P := \langle p_1(\vec{x}), \dots, p_m(\vec{x}) \rangle$ with $p_i(\vec{x}) \in R[x_1, \dots, x_n]$, its reverse derivative $n \times m \xrightarrow{R[P]} n$ is the tuple:

$$R[P] := \left\langle \sum_{i=1}^m \frac{\partial p_i}{\partial x_1}(\vec{x})y_i, \dots, \sum_{i=1}^m \frac{\partial p_i}{\partial x_n}(\vec{x})y_i \right\rangle$$

where each component of $R[P]$ is a polynomial in $R[x_1, \dots, x_n, y_1, \dots, y_m]$. For example, consider from before the polynomial $p(x_1, x_2) = x_1^2 + 3x_1x_2 + 5x_2$, then $3 \xrightarrow{R[p]} 2$ is the tuple of polynomials in 3 variables, $R[p] = \langle (2x_1 + 3x_2)y, (3x_1 + 5)y \rangle$.

2. **Smooth** is a reverse differential category whose reverse differential combinator is defined using the transpose of the Jacobian. For a map $n \xrightarrow{F} m$, that is, a smooth function $\mathbb{R}^n \xrightarrow{F} \mathbb{R}^m$, its reverse derivative $n \times m \xrightarrow{R[F]} n$ is the smooth map $\mathbb{R}^n \times \mathbb{R}^m \xrightarrow{R[F]} \mathbb{R}^n$ defined as:

$$R[F](\vec{x}, \vec{y}) := (J_f(x))^T \cdot \vec{y}$$

In particular for a smooth map $\mathbb{R}^n \xrightarrow{f} \mathbb{R}$, its reverse derivative $\mathbb{R}^n \times \mathbb{R} \xrightarrow{R[f]} \mathbb{R}^n$ is calculated out to be:

$$R[f](\vec{x}, y) := \left\langle \frac{\partial f}{\partial x_1}(\vec{x})y, \dots, \frac{\partial f}{\partial x_n}(\vec{x})y \right\rangle$$

And as before, $\text{POLY}_{\mathbb{R}}$ is a sub-reverse differential category of **Smooth**.

The following lemma captures some basic properties of the reverse derivative.

► **Lemma 15.** *In a Cartesian reverse differential category, the following equalities holds:*

1. $R[fg] = \langle \pi_0, \langle \pi_0 f, \pi_1 \rangle R[g] \rangle R[f]$;
2. $R[\iota_0] = \pi_1 \pi_0$ and $R[\iota_1] = \pi_1 \pi_1$;
3. $R[\pi_0 f] = (\pi_0 \times 1)R[f]\iota_0$ and $R[\pi_1 f] = (\pi_1 \times 1)R[f]\iota_1$;
4. $R[f\pi_0] = (1 \times \iota_0)R[f]$ and $R[f\pi_1] = (1 \times \iota_1)R[f]$;
5. $R[f \times g] = \text{ex}(R[f] \times R[g])$;
6. $R[\iota_0 f] = (\iota_0 \times 1)R[f]\pi_0$ and $R[\iota_1 f] = (\iota_1 \times 1)R[f]\pi_1$;
7. $R[f\iota_0] = (1 \times \pi_0)R[f]$ and $R[f\iota_1] = (1 \times \pi_1)R[f]$;
8. $R[\langle f|g \rangle] = \langle D[f\iota_0]R[g\iota_1] \rangle$;
9. $R[f \oplus g] = \text{ex}(R[f] \times R[g])$;

Proof. See the extended version [10] Lemma 15. ◀

3.1 Forward Differential Structure

Here we explain how every reverse derivative operator induces a forward derivative operator, that is, how every Cartesian reverse differential category is a Cartesian differential category. The trick was noticed in [6]: the reverse derivative in **Smooth** is the transpose of the Jacobian, which is linear, hence applying the reverse derivative again allows one to reconstruct the forward derivative. We formalize this in an arbitrary Cartesian reverse differential category as follows. Consider the resulting type of applying the reverse differential combinator twice:

$$\frac{\frac{A \xrightarrow{f} B}{A \times B \xrightarrow{R[f]} A}}{(A \times B) \times A \xrightarrow{R[R[f]]} (A \times B)}$$

► **Theorem 16.** *If \mathbb{X} is a Cartesian reverse differential category, then \mathbb{X} is a Cartesian differential category with differential combinator D defined as follows (for any map $A \xrightarrow{f} B$):*

$$D[f] := A \times A \xrightarrow{\langle (1, 0) \times 1 \rangle} (A \times B) \times A \xrightarrow{R[R[f]]} A \times B \xrightarrow{\pi_1} B$$

Proof. See [10] Theorem 16. ◀

► **Example 17.** For both $\text{POLY}_{\mathbb{R}}$ and **Smooth**, applying Theorem 16 to their respective reverse differential operators defined in Example 14 results precisely in their differential combinators defined in Example 5. This follows from the fact that there is a bijective correspondence between a reverse differential combinator and a differential combinator with an involution operation, which we will discuss in Section 4.

3.2 Dagger Structure and Linear Maps

We now investigate the subcategory of linear maps of the induced Cartesian differential category structure from Theorem 16 of a Cartesian reverse differential category. In particular we will show that the subcategory of linear maps has a dagger structure.

► **Definition 18.** *A \dagger -category [19] is a category \mathbb{X} with a stationary on objects involution $\mathbb{X}^{op} \xrightarrow{(\cdot)^\dagger} \mathbb{X}$. A \dagger -category that also has finite biproducts \oplus , with projection maps π_i and injection maps ι_i , is said to have \dagger -biproducts [19] when $\pi_i^\dagger = \iota_i$ (or equivalently if $\iota_i^\dagger = \pi_i$).*

Note that having \dagger -biproducts implies that $0^\dagger = 0$ and $(f + g)^\dagger = f^\dagger + g^\dagger$. At this point we can also point out that in the same way that every category with finite biproducts is a Cartesian differential category, we have the following basic example of a reverse differential category:

► **Example 19.** Every \dagger -category with finite \dagger -biproducts is a reverse differential category where for a map $A \xrightarrow{f} B$, $A \oplus B \xrightarrow{R[f]} A$ is defined as $R[f] := A \oplus B \xrightarrow{\pi_1} B \xrightarrow{f^\dagger} A$. As a particular example, let R be a commutative rig and let $\text{MAT}(R)$ be the category of matrices over R , that is, the category whose objects are the natural numbers $n \in \mathbb{N}$ and where a map $n \xrightarrow{A} m$ is an $n \times m$ -matrix A with coefficients in R . $\text{MAT}(R)$ admits finite biproducts where on objects $n \oplus m := n + m$ and where the projection and injection maps are the obvious matrices. $\text{MAT}(R)$ also admits a \dagger defined as the transpose of matrices and this makes $\text{MAT}(R)$ into a \dagger -category with finite \dagger -biproducts.

18:10 Reverse Derivative Categories

For any map $A \xrightarrow{f} B$ in a reverse differential category, we can define a map of opposite type $B \xrightarrow{f^\dagger} A$ by $f^\dagger := \iota_1 R[f]$. As the following example shows, however, in general this operation is neither functorial nor involutive.

► **Example 20.** With our standard example $2 \xrightarrow{p} 1$ in POLY_R , $p(x_1, x_2) = x_1^2 + 3x_1x_2 + 5x_2$, one computes that $1 \xrightarrow{p^\dagger} 2$ is the tuple of 1 variable polynomials $p^\dagger = \langle 0, 5x \rangle$.

However, as we shall see, \dagger is well behaved for linear maps.

► **Lemma 21.** *With the preceding definition of \dagger in a reverse differential category, one has that $\pi_i^\dagger = \iota_i$ and $\iota_i^\dagger = \pi_i$.*

► **Lemma 22.** *In a Cartesian reverse differential category, for any map $A \xrightarrow{f} B$, the following are equivalent:*

1. f is linear (Definition 9) with respect to the differential combinator of Theorem 16;
2. $\iota_1(\iota_0 \times 1)R[R[f]]\pi_1 = f$;
3. $f^{\dagger\dagger} = f$.

Proof. That $1 \Leftrightarrow 2$ follows from the fact that by definition, the left hand side of 2 can be re-expressed as $\iota_1(\iota_0 \times 1)R[R[f]]\pi_1 = \langle 0, 1 \rangle D[f]$, and so 2 holds precisely when $\langle 0, 1 \rangle D[f] = f$, which by Lemma 12 is equivalent to $D[f] = \pi_1 f$, that is, that f is linear. Next we show that $2 \Leftrightarrow 3$. First note that $\iota_1(\iota_0 \times 1) = \iota_1(\iota_1 \times 1)$ since:

$$\iota_1(\iota_0 \times 1) = \langle 0, 1 \rangle \langle \langle 1, 0 \rangle \times 1 \rangle = \langle 0, 1 \rangle = \langle 0, 1 \rangle \langle \langle 0, 1 \rangle \times 1 \rangle = \iota_1(\iota_1 \times 1)$$

And then by Lemma 15.6, we have the following equality:

$$f^{\dagger\dagger} = \iota_1 R[\iota_1 R[f]] = \iota_1(\iota_1 \times 1)R[R[f]]\pi_1 = \iota_1(\iota_0 \times 1)R[R[f]]\pi_1$$

Then it immediately follows that $f^{\dagger\dagger} = f$ if and only if $f = \iota_1(\iota_0 \times 1)R[R[f]]\pi_1$. ◀

► **Lemma 23.** *In a Cartesian reverse differential category, for any $A \xrightarrow{f} B$, its reverse derivative $A \times B \xrightarrow{R[f]} A$ is linear in B (Definition 9) with respect to the differential combinator of Theorem 16. Furthermore, the following diagram commutes:*

$$\begin{array}{ccc} ((A \times B) \times A) \times (A \times B) & \xrightarrow{R^{(3)}[f]} & (A \times B) \times A \\ \langle \iota_0, 0 \rangle \times \iota_1 \uparrow & & \downarrow \pi_1 \\ A \times B & \xrightarrow{R[f]} & A \end{array}$$

Proof. That $A \times B \xrightarrow{R[f]} A$ is linear in B follows immediately from the [RD.6] (we leave it as an exercise to re-express [RD.6] in terms of partial derivatives). Commutativity of the diagram follows by applying Lemma 12 to $R[f]$. ◀

► **Proposition 24.** *For a Cartesian reverse differential category \mathbb{X} , the category of linear maps of the induced Cartesian differential category structure from Theorem 16, $\text{Lin}(\mathbb{X})$, is a \dagger -category with finite \dagger -biproducts.*

Proof. By Proposition 11, we already know that $\text{Lin}(\mathbb{X})$ has finite biproducts. We need to show that $\text{Lin}(\mathbb{X})$ also has a \dagger . Lemma 22 shows that the linear maps are precisely those

for which $f^{\dagger\dagger} = f$, and thus if f is linear then f^{\dagger} is linear. Therefore \dagger is well-defined and involutive. We now show that \dagger is a contravariant functor. First that \dagger preserves the identity:

$$1^{\dagger} = \iota_1 R[1] = \iota_1 \pi_1 = 1$$

Next, that \dagger preserves composition (recall that if f is linear, then $0f = 0$):

$$\begin{aligned} (fg)^{\dagger} &= \iota_1 R[fg] = \iota_1 \langle \pi_0, \langle \pi_0 f, \pi_1 \rangle R[g] \rangle R[f] = \langle \iota_1 \pi_0, \langle \iota_1 \pi_0 f, \iota_1 \pi_1 \rangle R[g] \rangle R[f] \\ &= \langle 0, \langle 0f, 1 \rangle R[g] \rangle R[f] = \langle 0, \langle 0, 1 \rangle R[g] \rangle R[f] = \langle 0, 1 \rangle R[g] \langle 0, 1 \rangle R[f] = g^{\dagger} f^{\dagger} \end{aligned}$$

Note in the above that functoriality only relies on f preserving 0. Thus $\text{Lin}(\mathbb{X})$ is a \dagger -category. Lastly by Lemma 21, $\text{Lin}(\mathbb{X})$ also has \dagger -biproducts. \blacktriangleleft

4 From forward derivatives to reverse derivatives

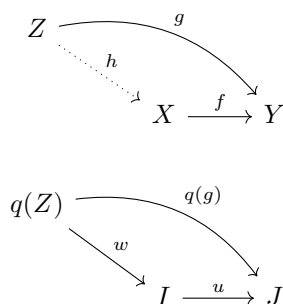
In the previous section, we showed that a Cartesian reverse differential category gives rise to a Cartesian differential category in which the subcategory of linear maps has a dagger biproduct structure. For the converse we need to develop Cartesian differential categories where every simple slice linear map category is a dagger category with dagger biproducts. The conceptual structure behind this is what we call a dagger fibration with fibered dagger biproducts. We will show that when a Cartesian differential category's linear map fibration is such a dagger fibration then the category is also a Cartesian reverse differential category.

4.1 Review of Fibrations and the Dual Fibration

We first recall the notion of fibration (for example, see [13, Section 1.1]) and the lesser-known idea of the dual of a fibration. These will be helpful concepts in which to frame our characterization of reverse differential categories (Theorem 42) and to describe how the reverse derivative is functorial (Proposition 31).

► **Definition 25.** Suppose that $q : \mathbb{X} \rightarrow \mathbb{B}$ is a functor.

1. Say that a morphism $f : X \rightarrow Y$ in \mathbb{X} is **over** a morphism $u : I \rightarrow J$ in \mathbb{B} if $q(f) = u$.
2. Say that a morphism $f : X \rightarrow Y$ in \mathbb{X} is **Cartesian over** $u : I \rightarrow J$ in \mathbb{B} if f is over u , and for every $g : Z \rightarrow Y$ in \mathbb{X} such that $q(g) = wu$ for some $w : q(Z) \rightarrow I$, there is a unique $h : Z \rightarrow X$ in \mathbb{X} over w such that $hf = g$:



3. Say that q is a **fibration** if for every Y in \mathbb{X} and every $u : I \rightarrow q(Y)$ in \mathbb{B} , there is a Cartesian morphism $f : X \rightarrow Y$ in \mathbb{X} above u .
4. Say that an arrow $f : X \rightarrow Y$ in \mathbb{X} is **vertical** if f is over an identity map.

18:12 Reverse Derivative Categories

5. For an object I in \mathbb{B} , the **fibred category** of q over I , denoted $q^{-1}(I)$, is the category whose objects are those objects of \mathbb{X} for which $q(X) = I$, and whose arrows are vertical morphisms between them.

► **Example 26.** If \mathbb{X} is a Cartesian category, then the **simple fibration** [13, Definition 1.3.1] $\widetilde{\mathbb{X}} \xrightarrow{\pi} \mathbb{X}$ is described as follows: the total category $\widetilde{\mathbb{X}}$ has objects pairs of objects of \mathbb{X} and a map $(I, A) \xrightarrow{(f, g)} (J, B)$ is given by a pair of maps of type $I \xrightarrow{f} J$ and $I \times A \xrightarrow{g} B$. The identity of (I, A) is $(1_A, \pi_1)$ while the composition of maps $(I, A) \xrightarrow{(f, g)} (J, B)$ and $(J, B) \xrightarrow{(f', g')} (K, C)$ is defined as: $(I \xrightarrow{f} J \xrightarrow{f'} K, I \times A \xrightarrow{\langle \pi_0 f, g \rangle} J \times B \xrightarrow{g'} C)$. The fibration $\widetilde{\mathbb{X}} \xrightarrow{\pi} \mathbb{X}$ is the functor which on objects is $\pi(I, A) = I$ and on maps is $\pi(f, g) := f$. The vertical arrows in $\widetilde{\mathbb{X}}$ are precisely those of the form $(I, A) \xrightarrow{(1, g)} (I, B)$ while the Cartesian arrows are those of the form $(I, A) \xrightarrow{(f, \pi_1)} (J, A)$.

► **Example 27.** If \mathbb{X} is a Cartesian differential category, we denote by $\widetilde{\text{Lin}}(\mathbb{X})$ the **simple linear fibration**, whose objects are pairs of objects in \mathbb{X} and whose maps $(I, A) \xrightarrow{f, g} (J, B)$ are pairs of maps $I \xrightarrow{f} J$ and $I \times A \xrightarrow{g} B$ where g is linear in B . Composition and identities of $\widetilde{\text{Lin}}(\mathbb{X})$ are defined as for the simple fibration. The fiber over A of this fibration is denoted $\widetilde{\text{Lin}}(\mathbb{X})[A]$. Note that by [2, Proposition 1.5.4], every fiber of $\widetilde{\text{Lin}}(\mathbb{X})$ has biproducts.

► **Definition 28.** Suppose that $\mathbb{X} \xrightarrow{q} \mathbb{B}$ is a fibration. The **dual fibration** of q [5, 14] is a fibration $\mathbb{X}^* \xrightarrow{q^*} \mathbb{B}$ whose total category \mathbb{X}^* has the same objects as \mathbb{X} and where a map $X \rightarrow Y$ in \mathbb{X}^* is an equivalence class of spans

$$\begin{array}{ccc} S & \xrightarrow{c} & Y \\ v \downarrow & & \\ X & & \end{array}$$

where v is vertical and c is Cartesian (over $q(c)$) under the equivalence relation $(v, c) \sim (v', c')$ when there is a vertical isomorphism α that makes the following diagram commute.

$$\begin{array}{ccccc} & & S' & & \\ & v' \swarrow & \downarrow \alpha & \searrow c' & \\ X & \xleftarrow{v} & S & \xrightarrow{c} & Y \end{array}$$

To compose such spans, note that given a cospan $S \xrightarrow{c} X' \xleftarrow{v'} S'$ with c cartesian and v' vertical, that there is a cartesian arrow over $q(c)$ with codomain S' , and this induces uniquely a v'' making the relevant square commute, and we get a span $S \xleftarrow{v''} S'' \xrightarrow{\hat{c}} S'$ with v'' vertical and \hat{c} cartesian; this span is used to form the composite of the spans $(v, c)(v', c')$. For more details, see [14]. The fibration q^* is defined on objects as $q^*(A) := q(A)$, and defined on maps as $q^*(v, c) := q(X) = q(S) \xrightarrow{q(c)} q(Y)$.

► **Example 29.** The dual of the simple fibration, $\widetilde{\mathbb{X}}^*$, can be described as the category with objects pairs of objects of \mathbb{X} and with maps $(I, A) \xrightarrow{(f, g)} (J, B)$ where $I \xrightarrow{f} J$ and a $I \times B \xrightarrow{g} A$. The identity on (I, A) is $(1, \pi_1)$, while composition of maps $(I, A) \xrightarrow{(f, g)} (J, B)$ and $(J, B) \xrightarrow{(f', g')} (K, C)$ is defined to be

$$(I \xrightarrow{ff'} K, I \times C \xrightarrow{\langle \pi_0, \langle \pi_0 f, \pi_1 \rangle \rangle} (I \times (J \times C)) \xrightarrow{1 \times g'} I \times B \xrightarrow{g} A).$$

► **Example 30.** The dual of the linear fibration, $\widehat{\text{Lin}(\mathbb{X})}^*$, has again objects (I, A) but now maps $(I, A) \xrightarrow{(f, g)} (J, B)$ consist of pairs of a map $I \xrightarrow{f} J$ and a map $I \times B \xrightarrow{g} A$ such that g is linear in B .

The dual of the linear fibration allows us to describe how the reverse derivative is functorial:

► **Proposition 31.** For a Cartesian reverse differential category \mathbb{X} , there is a product-preserving functor $\mathbb{X} \rightarrow \widehat{\text{Lin}(\mathbb{X})}^*$ defined on objects as $A \mapsto (A, A)$ and on maps as $f \mapsto (f, R[f])$.

Proof. This follows from [RD.3] and [RD.5]. ◀

► **Lemma 32.** A fiber of the dual fibration is isomorphic to the opposite category of the associated fiber of the starting fibration; that is, for any A in \mathbb{B} , $q^{*-1}(A) \simeq (q^{-1}(A))^{op}$ and moreover the isomorphism is stationary on objects.

Proof. See [10] Lemma 32. ◀

Note that \mathbb{X} and \mathbb{X}^{**} are also isomorphic as fibrations over \mathbb{B} ; see [14, Proposition 3.4].

4.2 Dagger fibrations

We now introduce the notion of a dagger fibration. First recall that a *morphism of fibrations* (over a fixed base) is a commuting triangle:

$$\begin{array}{ccc} \mathbb{X} & \xrightarrow{h} & \mathbb{Y} \\ & \searrow p & \swarrow q \\ & \mathbb{B} & \end{array}$$

where h carries Cartesian maps to Cartesian maps.

► **Definition 33.** A **dagger fibration** is given by a fibration $\mathbb{X} \xrightarrow{q} \mathbb{B}$ with a morphism of fibrations $\mathbb{X} \xrightarrow{(_)^\dagger} \mathbb{X}^*$ such that

$$\begin{array}{ccccc} & & 1_{\mathbb{X}} & & \\ & \searrow & \curvearrowright & \swarrow & \\ \mathbb{X} & \xrightarrow{(_)^\dagger} & \mathbb{X}^* & \xrightarrow{(_)^\dagger} & \mathbb{X}^{**} = \mathbb{X} \\ & \searrow q & \downarrow q^* & \swarrow q & \\ & & \mathbb{B} & & \end{array}$$

and such that \dagger is stationary on objects. A dagger fibration has a **dagger cleavage** when $(_)^\dagger$ sends cloven cartesian arrows to cloven cartesian arrows.

Our main example of a dagger fibration will be the linear fibration of a Cartesian reverse differential category. We begin by defining the required dagger (this is a more general form of the dagger discussed earlier in Section 3.2):

► **Definition 34.** In a Cartesian reverse differential category \mathbb{X} , for a map $C \times A \xrightarrow{f} B$, define the **contextual** \dagger of f , $C \times B \xrightarrow{f^\dagger[C]} A$, as follows:

$$f^\dagger[C] := C \times B \xrightarrow{\iota_0 \times 1} (C \times A) \times B \xrightarrow{R[f]} C \times A \xrightarrow{\pi_1} A$$

18:14 Reverse Derivative Categories

► **Lemma 35.** *In a Cartesian reverse differential category, for any map $C \times A \xrightarrow{f} B$, the following are equivalent:*

1. f is linear in A (Definition 9) with respect to the differential combinator of Theorem 16;
2. $(\iota_0 \times \iota_1)\text{exD}[f] = f$;
3. $f^{\dagger[C]\dagger[C]} = f$.

Proof. $1 \Leftrightarrow 2$ follows from Lemma 12. To show that $2 \Leftrightarrow 3$ requires a bit more work, but the proof is essentially the same as in Lemma 22. ◀

► **Corollary 36.** *Let \mathbb{X} be a Cartesian reverse differential category and let $I \times A \xrightarrow{g} B$ be linear in A . Then $I \times B \xrightarrow{g^{\dagger[I]}} A$ is linear in B .*

► **Theorem 37.** *If \mathbb{X} is a Cartesian reverse differential category, then its associated linear fibration is a dagger fibration, with dagger as in Definition 34.*

Proof. See [10] Theorem 37. ◀

► **Lemma 38.** *If $\mathbb{X} \xrightarrow{q} \mathbb{B}$ is a dagger fibration with a dagger cleavage, then each fiber $q^{-1}(A)$ is a \dagger -category, and reindexing preserves the dagger.*

Proof. See [10] Lemma 38. ◀

4.3 Characterization of Cartesian reverse differential categories

We have seen in the previous sections that a Cartesian reverse differential category is a Cartesian differential category whose associated linear fibration is a dagger fibration in which each fibre has \dagger -biproducts. In this final section, we show that this collection of structures characterizes Cartesian reverse differential categories.

► **Definition 39.** *Let \mathbb{X} be a Cartesian differential category. We say that \mathbb{X} has a **contextual linear dagger** when the linear fibration is a dagger fibration*

$$\begin{array}{ccc} \widetilde{\text{Lin}}(\mathbb{X}) & \xrightarrow{(_)^\dagger} & \widetilde{\text{Lin}}(\mathbb{X})^* \\ & \searrow \pi & \swarrow \pi^* \\ & \mathbb{X} & \end{array}$$

and each fiber category $\text{Lin}(\mathbb{X})[A]$ has \dagger -biproducts.

By Lemma 38, every fiber of such a fibration is a \dagger -category, and reindexing functors preserve the dagger. We denote the \dagger in the fiber $\text{Lin}(\mathbb{X})[A]$ by $(_)^\dagger[A]$. In particular we note that $(_)^\dagger[A]$ preserves the additive structure. Before giving the main theorems of this section, we will need the following lemma:

► **Lemma 40.** *Let \mathbb{X} be a Cartesian differential category with a contextual linear dagger. For any map $A \xrightarrow{f} B$ the following diagram commutes.*

$$\begin{array}{ccc} (A \times B) \times A & \xrightarrow{\text{D}[\text{D}[f]^\dagger[A]^\dagger[A \times B]]} & A \times B \\ \langle 1, 0 \rangle \times 1 \uparrow & & \downarrow \pi_1 \\ A \times A & \xrightarrow{\text{D}[f]} & B \end{array}$$

Proof. See [10] Lemma 40. ◀

► **Theorem 41.** *A Cartesian differential category \mathbb{X} with a contextual linear dagger is a Cartesian reverse differential category with reverse differential combinator R defined as follows (for a map $A \xrightarrow{f} B$):*

$$R[f] := A \times B \xrightarrow{D[f]^\dagger[A]} B$$

Proof. See [10] Theorem 41. ◀

We conclude with the main result of this paper:

► **Theorem 42.** *A Cartesian reverse differential category is precisely a Cartesian differential category with a contextual linear dagger.*

Proof. See [10] Theorem 42. ◀

5 Concluding remarks

This paper begins the story of categories with a reverse derivative; however, there is much more that needs to be done in this area. Perhaps the most important next step is to add partiality into this setting. One way to add partiality to categories is via a restriction structure [8]. The paper [7] showed how to combine a Cartesian differential structure with a restriction structure to obtain “differential restriction categories.” This provides an axiomatization for categories of smooth partial maps. A key next step is then to combine reverse differential categories with restriction structure, and check that many of the results that held for differential restriction categories hold for “reverse differential restriction categories”. Such a structure would bring us even closer to a true categorical semantics for differential programming.

Another important aspect to develop will be the term logic for reverse differential categories. The term logic for Cartesian differential categories greatly facilitates the ability to establish and prove results in that abstract setting; a term logic for reverse differential categories is similarly important.

Tensors are another important aspect of differential programming, and form the foundations on which modern, large scale machine learning platforms are based [1]. In [3], monoidal structure was described in a way that interacts well with differentiation. In particular, $V \otimes W$ is the object for which bilinear maps $V \times W \rightarrow U$ correspond to linear maps $V \otimes W \rightarrow U$. Developing a similar structure for the reverse derivative will thus also be important. More generally, there should be a notion of (monoidal) reverse differential category. These should provide additional examples of Cartesian reverse differential categories: just as the coKleisli category of a (monoidal) differential category [4] is a Cartesian differential category, so should the coKleisli category of a monoidal reverse differential category be a Cartesian reverse differential category.

Finally, an important generalization of Cartesian differential categories are tangent categories [9], a categorical setting for differential geometry which axiomatizes the existence of a “tangent bundle” for each object. Every Cartesian differential category gives rise to a tangent category. A reverse derivative category should give a “category with a cotangent bundle for each object”; defining such categories will be another important extension of this work.

References

- 1 Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz

- Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. URL: <https://www.tensorflow.org/>.
- 2 R. Blute, R. Cockett, and R. Seely. Cartesian Differential Categories. *Theory and Applications of Categories*, 22:622–672, 2009.
 - 3 R. Blute, R. Cockett, and R. Seely. Cartesian Differential Storage Categories. *Theory and Applications of Categories*, 30(18):620–686, 2015.
 - 4 R.F. Blute, J.R.B. Cockett, and R.A.G. Seely. Differential categories. *Mathematical structures in computer science*, 16(6):1049–1083, 2006.
 - 5 F. Borceaux. *Handbook of categorical algebra II*. Cambridge University Press, 2008.
 - 6 Bruce Christianson. A Leibniz notation for automatic differentiation. In *Recent Advances in Algorithmic Differentiation*, volume 87 of *Lecture Notes in Computational Science and Engineering*, pages 1–9. Springer, 2012.
 - 7 J.R.B. Cockett, G.S.H. Cruttwell, and J.D. Gallagher. Differential restriction categories. *Theory and applications of categories*, 25(21):537–613, 2011.
 - 8 J.R.B. Cockett and Stephen Lack. Restriction categories I: categories of partial maps. *Theoretical Computer Science*, 270(1):223–259, 2002. doi:10.1016/S0304-3975(00)00382-0.
 - 9 R. Cockett and G. Cruttwell. Differential structure, tangent structure, and SDG. *Applied Categorical Structures*, 22:331–417, 2014.
 - 10 Robin Cockett, Geoffrey Cruttwell, Jonathan Gallagher, Jean-Simon Lema, Benjamin Mac-Adam, Gordon Plotkin, and Dorette Pronk. Reverse derivative categories. *Arxiv*, 2019.
 - 11 Conal Elliott. The simple essence of automatic differentiation. *Proceedings of the ACM on Programming Languages*, 2(ICFP):70, 2018.
 - 12 Andreas Griewank. Who invented the reverse mode of differentiation. *Documenta Mathematica, Extra Volume ISMP*, pages 389–400, 2012.
 - 13 B. Jacobs. *Categorical logic and type theory*. Number 141 in *Studies in logic and the foundations of mathematics*. Elsevier, 1999.
 - 14 Anders Kock. The dual fibration in elementary terms. *arXiv e-prints*, page arXiv:1501.01947, January 2015. arXiv:1501.01947.
 - 15 J-S P. Lema. A Tangent Category Alternative to the Faa Di Bruno Construction. *Theory and Applications of Categories*, 33(35):1072–1110, 2018.
 - 16 S. Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16:146–160, 1976.
 - 17 G. Plotkin. A Simple Differential Programming Language. MFPS 2018 Keynote Address, June 2018.
 - 18 David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by backpropagating errors. *Cognitive modeling*, 5:3, 1988. URL: www.cs.toronto.edu/hinton/naturebp.pdf.
 - 19 Peter Selinger. Dagger Compact Closed Categories and Completely Positive Maps. *Electron. Notes Theor. Comput. Sci.*, 170:139–163, March 2007. doi:10.1016/j.entcs.2006.12.018.

Internal Calculi for Separation Logics

Stéphane Demri

LSV, CNRS, ENS Paris-Saclay, Université Paris-Saclay, France

Etienne Lozes

Université Côte d’Azur, CNRS, I3S, France

Alessio Mansutti

LSV, CNRS, ENS Paris-Saclay, Université Paris-Saclay, France

Abstract

We present a general approach to axiomatise separation logics with heaplet semantics with no external features such as nominals/labels. To start with, we design the first (internal) Hilbert-style axiomatisation for the quantifier-free separation logic $SL(*, -*)$. We instantiate the method by introducing a new separation logic with essential features: it is equipped with the separating conjunction, the predicate $1s$, and a natural guarded form of first-order quantification. We apply our approach for its axiomatisation. As a by-product of our method, we also establish the exact expressive power of this new logic and we show PSPACE-completeness of its satisfiability problem.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Separation logic, internal calculus, adjunct/quantifier elimination

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.19

Acknowledgements We would like to thank the anonymous reviewers for their suggestions and remarks that help us to improve the quality of this paper.

1 Introduction

The virtue of axiomatising program logics. Designing a Hilbert-style axiomatisation for your favourite logic is usually quite challenging. This does not lead necessarily to optimal decision procedures, but the completeness proof usually provides essential insights to better understand the logic at hand. That is why many logics related to program verification have been axiomatised, often requiring non-trivial completeness proofs. By way of example, there exist axiomatisations for the linear-time μ -calculus [28, 19], the modal μ -calculus [39] or for the alternating-time temporal logic ATL [23]. Concerning the separation logics that extend Hoare-Floyd logic to verify programs with mutable data structures (see e.g. [34, 38, 27, 33, 37]), a Hilbert-style axiomatisation of Boolean BI has been introduced in [21], but remained at the abstract level of Boolean BI. More recently, HyBBI [8], a hybrid version of Boolean BI has been introduced in order to axiomatise various classes of separation logics; HyBBI naturally considers classes of abstract models (typically preordered partial monoids) but it does not fit exactly the heaplet semantics of separation logics. Furthermore, the addition of nominals (in the sense of hybrid modal logics, see e.g. [1]) extends substantially the object language. Other frameworks to axiomatise classes of abstract separation logics can be found in [18] and in [25], respectively with labelled tableaux calculi and with sequent-style proof systems.

Our motivations. Since the birth of separation logics, there has been a lot of interest in the study of decidability and computational complexity issues, see e.g. [3, 10, 11, 7, 15, 32], and comparatively a bit less attention to the design of proof systems, and even less with the puristic approach that consists in discarding any external feature such as nominals or labels in the calculi. The well-known advantages of such an approach include an exhaustive understanding of the expressive power of the logic and discarding the use of any external



© Stéphane Demri, Etienne Lozes, and Alessio Mansutti;
licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 19; pp. 19:1–19:18



Leibniz International Proceedings in Informatics

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

artifact referring to semantical objects. For instance, a complete tableaux calculus with labels for quantifier-free separation logic is designed in [22] –with an extension of the calculus to handle quantifiers, whereas Hilbert-style calculi for abstract separation logics with nominals are defined in [8] (see also in [26] a proof system for a first-order abstract separation logic with an abstracted version of the points-to predicate). Similarly, display calculi for bunched logics are provided in [5] and such calculi extend Gentzen-style proof systems by allowing new structural connectives. In this paper, we advocate a puristic approach and aim at designing Hilbert-style proof systems for quantifier-free separation logic $\text{SL}(*, \multimap)$ (which includes the separating conjunction $*$ and implication \multimap , as well as all Boolean connectives) and more generally for other separation logics, while remaining within the very logical language. Consequently, in this work we only focus on axiomatising the separation logics, and we have no claim for practical applications in the field of program verification. Aiming at internal calculi is a non-trivial task as the general frameworks for abstract separation logics make use of labels, see e.g. [18, 25]. We cannot fully rely on label-free calculi for BI, see e.g. [36, 21], as separation logics are usually understood as Boolean BI interpreted on models of heap memory and therefore require calculi that handle specifically the stack-and-heap models. Finally, we know translations from separation logics into logics/theories, see e.g. [9, 35, 4], but completeness cannot always be inherited by sublogics as the proof system should only use the sublogic and therefore their axiomatisation may lead to different methods.

Our contribution. Though our initial motivation is to design an internal Hilbert-style axiomatisation for $\text{SL}(*, \multimap)$, we go beyond this, and we propose a method to axiomatise other separation logics assuming that key properties are satisfied. Hence, we consider a broader perspective and we use our approach on two separation logics: quantifier-free separation logic and a new separation logic that admits a form of guarded first-order quantification. Our results are not limited to (internal) axiomatisation, as we provide a complexity analysis based on the properties of the derivations in the proof system. Let us be a bit more precise.

In Section 3, we provide the first Hilbert-style proof system for $\text{SL}(*, \multimap)$ that uses axiom schemas and rules involving only formulae of this logic. Each formula of $\text{SL}(*, \multimap)$ is equivalent to a Boolean combination of *core formulae*: simple formulae of the logic expressing elementary properties about the models [30]. Though core formulae (also called *test formulae*) have been handy in several occasions for establishing complexity results for separation logics, see e.g. [14, 15, 20], in the paper, these formulae are instrumental for the axiomatisation. Indeed, we distinguish the axiomatisation of Boolean combinations of core formulae from the transformation of formulae into such Boolean combinations. Thus, we show how to introduce axioms to transform every formula into a Boolean combination of core formulae, together with axioms to deal with these simple formulae. Schematically, for a valid formula φ , we conclude $\vdash \varphi$ from $\vdash \varphi'$ and $\vdash \varphi' \Leftrightarrow \varphi$, where φ' is a Boolean combination of core formulae. Another difficulty arises as we have to design an axiomatisation for such Boolean combinations. So, the calculus is divided in three parts: the axiomatisation of Boolean combinations of core formulae, axioms and inference rules to simulate a bottom-up elimination of separating connectives, and finally axioms and inference rules from propositional calculus and Boolean BI. Such an approach that consists in first axiomatising a syntactic fragment of the whole logic (in our case, the core formulae), is best described in [19] (see also [39, 40, 31, 13]).

In Section 4, our intention is to add standard features to the logic such as first-order quantification and inductive predicates, and to apply our method for axiomatisation. As $\text{SL}(*, \multimap, \text{ls})$ (i.e. $\text{SL}(*, \multimap)$ enriched with the predicate ls) is already non-finitely axiomatisable [16], we need to fine-tune the logical formalism. That is why, we introduce a new

separation logic $\text{SL}(*, \exists: \rightsquigarrow)$ that admits the separating conjunction $*$ (no -) and a guarded form of first-order quantification. In the formula $\exists \mathbf{z}: \langle \mathbf{x} \rightsquigarrow \mathbf{y} \rangle \varphi$, the variable \mathbf{z} is existentially quantified over the set of locations in the minimal non-empty path from \mathbf{x} to \mathbf{y} , if any. The logic $\text{SL}(*, \exists: \rightsquigarrow)$ contains the symbolic heap fragment [2, 11] but also richer logics such as $\text{SL}(*, \text{reach}^+)$ from [15]. Hence, the logic $\text{SL}(*, \exists: \rightsquigarrow)$ captures the list segment predicate ls but also allows us to quantify in a guarded form over locations in a minimal path, which makes it a promising language. We provide an internal Hilbert-style axiomatisation for $\text{SL}(*, \exists: \rightsquigarrow)$, illustrating the flexibility of our method. It requires the design of an adequate family of core formulae that captures $\text{SL}(*, \exists: \rightsquigarrow)$. The axiomatisation of Boolean combinations of core formulae reveals to be challenging, and the elimination of guarded quantification or separating conjunction happens also to require complex developments. We analyse the derivations from the calculus to establish a small model property for the logic and, together with a symbolic model-checking algorithm, prove that the satisfiability problem for $\text{SL}(*, \exists: \rightsquigarrow)$ is in PSPACE.

2 Preliminaries

Quantifier-free separation logic $\text{SL}(*, \text{-})$. We present the quantifier-free separation logic $\text{SL}(*, \text{-})$, that includes standard features such as the separating conjunction $*$ and the separating implication - . Let $\text{VAR} = \{\mathbf{x}, \mathbf{y}, \dots\}$ be a countably infinite set of *program variables*. The formulae φ of $\text{SL}(*, \text{-})$ and its atomic formulae π are built from the grammars below (where $\mathbf{x}, \mathbf{y} \in \text{VAR}$ and the connectives $\Rightarrow, \Leftrightarrow$ and \vee are defined as usually).

$$\pi ::= \mathbf{x} = \mathbf{y} \mid \mathbf{x} \hookrightarrow \mathbf{y} \mid \text{emp} \quad \varphi ::= \pi \mid \neg \varphi \mid \varphi \wedge \varphi \mid \varphi * \varphi \mid \varphi \text{-} * \varphi.$$

In the heaplet semantics, the formulae of $\text{SL}(*, \text{-})$ are interpreted on *memory states* that are pairs (s, h) where $s : \text{VAR} \rightarrow \text{LOC}$ is a variable valuation (the *store*) from the set of program variables to a countably infinite set of *locations* $\text{LOC} = \{\ell_0, \ell_1, \ell_2, \dots\}$ whereas $h : \text{LOC} \rightarrow_{\text{fin}} \text{LOC}$ is a partial function with finite domain (the *heap*). We write $\text{dom}(h)$ to denote its domain and $\text{ran}(h)$ to denote its range. A *memory cell* of h is understood as a pair of locations (ℓ, ℓ') such that $\ell \in \text{dom}(h)$ and $\ell' = h(\ell)$. As usual, the heaps h_1 and h_2 are said to be *disjoint*, written $h_1 \perp h_2$, if $\text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset$; when this holds, we write $h_1 + h_2$ to denote the heap corresponding to the disjoint union of the graphs of h_1 and h_2 , hence $\text{dom}(h_1 + h_2) = \text{dom}(h_1) \uplus \text{dom}(h_2)$. Moreover, we write $h' \sqsubseteq h$ to denote that $\text{dom}(h') \subseteq \text{dom}(h)$ and for all locations $\ell \in \text{dom}(h')$, we have $h'(\ell) = h(\ell)$. Given a heap h , we define a family of $(h^\delta)_{\delta \in \mathbb{N}}$ of partial functions such that h^0 is the identity function on LOC , $h^1 = h$ and for all $\delta \geq 2$ and $\ell \in \text{LOC}$, we have $h^\delta(\ell) \stackrel{\text{def}}{=} h(h^{\delta-1}(\ell))$, assuming that $h^{\delta-1}(\ell)$ is defined and belongs to $\text{dom}(h)$, otherwise $h^\delta(\ell)$ is undefined. The satisfaction relation \models is defined as follows (omitting standard clauses for \neg, \wedge):

$$\begin{aligned} (s, h) \models \mathbf{x} = \mathbf{y} &\stackrel{\text{def}}{\Leftrightarrow} s(\mathbf{x}) = s(\mathbf{y}) & (s, h) \models \text{emp} &\stackrel{\text{def}}{\Leftrightarrow} \text{dom}(h) = \emptyset \\ (s, h) \models \mathbf{x} \hookrightarrow \mathbf{y} &\stackrel{\text{def}}{\Leftrightarrow} s(\mathbf{x}) \in \text{dom}(h) \text{ and } h(s(\mathbf{x})) = s(\mathbf{y}) \\ (s, h) \models \varphi_1 * \varphi_2 &\stackrel{\text{def}}{\Leftrightarrow} \exists h_1, h_2. h_1 \perp h_2, (h_1 + h_2) = h, (s, h_1) \models \varphi_1 \text{ and } (s, h_2) \models \varphi_2 \\ (s, h) \models \varphi_1 \text{-} * \varphi_2 &\stackrel{\text{def}}{\Leftrightarrow} \forall h_1. (h_1 \perp h \text{ and } (s, h_1) \models \varphi_1) \text{ implies } (s, h + h_1) \models \varphi_2. \end{aligned}$$

We denote with \perp the contradiction $\mathbf{x} \neq \mathbf{x}$, and with \top its negation $\neg \perp$. The septraction operator \oplus (kind of dual of -), defined by $\varphi \oplus \psi \stackrel{\text{def}}{=} \neg(\varphi \text{-} * \neg \psi)$, has the following semantics:

$$(s, h) \models \varphi \oplus \psi \Leftrightarrow \text{there is a heap } h' \text{ such that } h \perp h', (s, h') \models \varphi, \text{ and } (s, h + h') \models \psi.$$

Moreover, we introduce the following (important) shortcuts:

- $\text{alloc}(\mathbf{x})$ which is satisfied by (s, h) iff $s(\mathbf{x}) \in \text{dom}(h)$. It is defined as $(\mathbf{x} \hookrightarrow \mathbf{x}) \text{-} * \perp$.

- $\mathbf{size} \geq \beta$ which is satisfied by (s, h) iff $\text{card}(\text{dom}(h)) \geq \beta$, where $\beta \in \mathbb{N}$ and $\text{card}(X)$ denotes the cardinality of the set X . This shortcut is inductively defined as $\mathbf{size} \geq 0 \stackrel{\text{def}}{=} \top$, $\mathbf{size} \geq 1 \stackrel{\text{def}}{=} \neg \mathbf{emp}$ and, for each $\beta \in \mathbb{N}$, $\mathbf{size} \geq \beta+2 \stackrel{\text{def}}{=} \neg \mathbf{emp} * \mathbf{size} \geq \beta+1$.

We use $\mathbf{size}=\beta$ as a shorthand for $\mathbf{size} \geq \beta \wedge \neg \mathbf{size} \geq \beta+1$. A formula φ is *valid* if $(s, h) \models \varphi$ for all (s, h) (and we write $\models \varphi$). For a complete description of separation logic, see e.g. [38].

Hilbert-style proof systems. A *Hilbert-style proof system* \mathcal{H} is defined as a set of *derivation step schemata* $((\Phi_1, \dots, \Phi_n), \Psi)$ with $n \geq 0$, where $\Phi_1, \dots, \Phi_n, \Psi$ are *formula schemata*. When $n \geq 1$, $((\Phi_1, \dots, \Phi_n), \Psi)$ is called an *inference rule*, otherwise it is an *axiom*. As usual, formula schemata generalise the notion of formulae by allowing metavariables for formulae (typically φ, ψ, χ), for program variables (typically $\mathbf{x}, \mathbf{y}, \mathbf{z}$) or for any type of syntactic objects in formulae, depending on the context. The set of formulae *derivable* from \mathcal{H} is the least set S such that for all $((\Phi_1, \dots, \Phi_n), \Psi) \in \mathcal{H}$ and for all substitutions σ such that $\Phi_1\sigma, \dots, \Phi_n\sigma \in S$, $\Psi\sigma \in S$. We write $\vdash_{\mathcal{H}} \varphi$ if φ is derivable from \mathcal{H} . A proof system \mathcal{H} is *sound* if all derivable formulae are valid. \mathcal{H} is *complete* if all valid formulae are derivable. \mathcal{H} is *strongly complete* iff for all sets of formulae Γ and formulae φ , we have $\Gamma \models \varphi$ (semantical entailment) iff $\vdash_{\mathcal{H} \cup \Gamma} \varphi$.

Interestingly enough, there is no strongly complete proof system for separation logic, as strong completeness implies compactness and separation logic is not compact. Indeed, $\{\mathbf{size} \geq \beta \mid \beta \in \mathbb{N}\}$ is unsatisfiable, as heaps have finite domains, but all finite subsets of it are satisfiable. Even for the weaker notion of completeness, deriving an Hilbert-style axiomatisation for $\text{SL}(*, \text{-}*)$ remains challenging. Indeed, the satisfiability problem for $\text{SL}(*, \text{-}*)$ reduces to its validity problem, making $\text{SL}(*, \text{-}*)$ an unusual logic from a proof-theoretical point of view. Let us develop a bit further this point. Let φ be a formula with program variables in $\mathbf{X} \subseteq_{\text{fin}} \text{VAR}$, and let \approx be an equivalence relation on \mathbf{X} . The formula $\psi_{\approx} \stackrel{\text{def}}{=} (\mathbf{emp} \wedge \bigwedge_{x \approx y} \mathbf{x} = \mathbf{y} \wedge \bigwedge_{x \not\approx y} \mathbf{x} \neq \mathbf{y}) \Rightarrow (\varphi \oplus \top)$ can be shown to be valid iff for every store s agreeing on \approx , there is a heap h such that $(s, h) \models \varphi$. It is known that for all stores s, s' agreeing on \approx , and every heap h , (s, h) and (s', h) satisfy the same set of formulae having variables from \mathbf{X} . Since the antecedent of ψ_{\approx} is satisfiable, we conclude that ψ_{\approx} is valid iff there are a store s agreeing on \approx and a heap h such that $(s, h) \models \varphi$. To check whether φ is satisfiable, it is sufficient to find an equivalence relation \approx on \mathbf{X} such that ψ_{\approx} is valid. As the number of equivalence relations on \mathbf{X} is finite, we obtain a Turing reduction from satisfiability to validity. Consequently, it is not possible to define sound and complete axiom systems for any extension of $\text{SL}(*, \text{-}*)$ admitting an undecidable validity problem (as long as there is a reduction from satisfiability to validity, as above). A good example is $\text{SL}(*, \text{-}*, \mathbf{1s})$ [16] (extension of $\text{SL}(*, \text{-}*)$ with $\mathbf{1s}$). Indeed, in order to obtain a sound and complete axiom system, the validity problem has to be recursively enumerable (r.e.). However, this would imply that the satisfiability problem is also r.e.. As φ is not valid iff $\neg\varphi$ is satisfiable, we then conclude that the set of valid formulae is recursive, hence decidable, a contradiction.

It is worth also noting that quantifier-free $\text{SL}(*, \text{-}*)$ axiomatised below admits a PSPACE-complete validity problem, see e.g. [10], and should not be confused with propositional separation logic with the stack-heap models shown undecidable in [6, Corollary 5.1] (see also [12]), in which there are propositional variables interpreted by sets of memory states.

3 Hilbert-style proof system for $\text{SL}(*, \text{-}*)$

We define a proof system for $\text{SL}(*, \text{-}*)$, namely $\mathcal{H}_{\text{C}}(*, \text{-}*)$, by relying on its *core formulae*: simple $\text{SL}(*, \text{-}*)$ formulae capturing essential properties of the models, see e.g. [29, 41]. It is known that every $\text{SL}(*, \text{-}*)$ formula is logically equivalent to a Boolean combination of

(System 1) \mathcal{H}_C : Axioms for Boolean combinations of core formulae	
(A_1^C) $x = x$	(A_4^C) $x \hookrightarrow y \wedge x \hookrightarrow z \Rightarrow y = z$
(A_2^C) $\varphi \wedge x = y \Rightarrow \varphi[y \leftarrow x]$	(I_5^C) $\text{size} \geq \beta + 1 \Rightarrow \text{size} \geq \beta$
(A_3^C) $x \hookrightarrow y \Rightarrow \text{alloc}(x)$	(I_6^C) $\bigwedge_{x \in X} (\text{alloc}(x) \wedge \bigwedge_{y \in X \setminus \{x\}} x \neq y) \Rightarrow \text{size} \geq \text{card}(X)$
(System 2) Axioms and inference rule for the separating conjunction	
(A_7^*) $(\varphi * \psi) \Leftrightarrow (\psi * \varphi)$	(A_{14}^*) $e * \top \Rightarrow e$ $\left[e \text{ is } \neg \text{emp}, x = y, x \neq y \text{ or } x \hookrightarrow y \right]$
(A_8^*) $(\varphi * \psi) * \chi \Leftrightarrow \varphi * (\psi * \chi)$	(A_{15}^*) $\neg \text{alloc}(x) * \neg \text{alloc}(x) \Rightarrow \neg \text{alloc}(x)$
(I_9^*) $(\varphi \vee \psi) * \chi \Rightarrow (\varphi * \chi) \vee (\psi * \chi)$	(A_{16}^*) $(\text{alloc}(x) \wedge \neg x \hookrightarrow y) * \top \Rightarrow \neg x \hookrightarrow y$
(I_{10}^*) $(\perp * \varphi) \Leftrightarrow \perp$	(A_{17}^*) $\text{alloc}(x) \Rightarrow (\text{alloc}(x) \wedge \text{size} = 1) * \top$
(A_{11}^*) $\varphi \Leftrightarrow \varphi * \text{emp}$	(A_{18}^*) $\neg \text{emp} \Rightarrow \text{size} = 1 * \top$
(I_{12}^*) $\text{alloc}(x) * \top \Rightarrow \text{alloc}(x)$	(A_{19}^*) $\neg \text{size} \geq \beta_1 * \neg \text{size} \geq \beta_2 \Rightarrow \neg \text{size} \geq \beta_1 + \beta_2 - 1$
(I_{13}^*) $(\text{alloc}(x) * \text{alloc}(x)) \Leftrightarrow \perp$	(A_{20}^*) $\text{alloc}(x) \wedge \text{alloc}(y) \wedge x \neq y \Rightarrow \text{size} \geq 2$
$*\text{-Intro}$: $\frac{\varphi \Rightarrow \chi}{\varphi * \psi \Rightarrow \chi * \psi}$	where $a \dot{-} b = a - b$ if $a \geq b$, 0 otherwise.
(System 3) Axioms and inference rules for the separating implication	
(A_{21}^*) $(\text{size} = 1 \wedge \bigwedge_{x \in X} \neg \text{alloc}(x)) \oplus \top \left[\bigwedge X \subseteq_{\text{fin}} \text{VAR} \right]$	$*\text{-Adj}$: $\frac{\varphi * \psi \Rightarrow \chi}{\varphi \Rightarrow (\psi * \chi)}$ -*Adj : $\frac{\varphi \Rightarrow (\psi * \chi)}{\varphi * \psi \Rightarrow \chi}$
(A_{22}^*) $\neg \text{alloc}(x) \Rightarrow ((x \hookrightarrow y \wedge \text{size} = 1) \oplus \top)$	
(A_{23}^*) $\neg \text{alloc}(x) \Rightarrow ((\text{alloc}(x) \wedge \text{size} = 1 \wedge \bigwedge_{y \in X} \neg x \hookrightarrow y) \oplus \top) \left[\bigwedge X \subseteq_{\text{fin}} \text{VAR} \right]$	
1	$\text{emp} \Rightarrow \neg \text{size} \geq 1$ $(\neg \neg \text{E})$ and def. of $\text{size} \geq 1$
2	$\text{alloc}(x) \wedge \text{size} = 1 \Rightarrow \neg \text{size} \geq 2$ $(\wedge \text{Er})$
3	$\text{emp} * (\text{alloc}(x) \wedge \text{size} = 1) \Rightarrow \neg \text{size} \geq 1 * \neg \text{size} \geq 2$ $*\text{-Itr}$, 1, 2
4	$\neg \text{size} \geq 1 * \neg \text{size} \geq 2 \Rightarrow \neg \text{size} \geq 2$ (A_{19}^*)
5	$(\text{emp} * (\text{alloc}(x) \wedge \text{size} = 1)) \Rightarrow \neg \text{size} \geq 2$ $\Rightarrow \text{-Tr}$, 3, 4
6	$\text{emp} \Rightarrow ((\text{alloc}(x) \wedge \text{size} = 1) * \neg \text{size} \geq 2)$ $*\text{-Adj}$ rule, 5

■ **Figure 1** Proof of $\text{emp} \Rightarrow ((\text{alloc}(x) \wedge \text{size} = 1) * \neg \text{size} \geq 2)$.

core formulae [29]. However, as every core formula is an $\text{SL}(*, *)$ formula, we stay in the original language and we can derive an axiomatisation of $\text{SL}(*, *)$ by extending the axiom system of propositional calculus with three sets of axioms and inference rules: the axioms and inference rules of the propositional logic of core formulae (System 1), the axioms and inference rules witnessing that every formula of the form $\varphi_1 * \varphi_2$, where φ_1, φ_2 are Boolean combinations of core formulae is logically equivalent to a Boolean combination of core formulae (System 2), and the axioms and inference rules to eliminate formulae whose outermost connective is the separating implication $*$ (System 3). The *core formulae* are expressions of the form $x = y$, $\text{alloc}(x)$, $x \hookrightarrow y$ and $\text{size} \geq \beta$, where $x, y \in \text{VAR}$ and $\beta \in \mathbb{N}$. As previously shown, these formulae are from $\text{SL}(*, *)$ and are used in the axiom system as abbreviations. Given $X \subseteq_{\text{fin}} \text{VAR}$ and $\alpha \in \mathbb{N}$, we define $\text{Core}(X, \alpha)$ as the set $\{x = y, \text{alloc}(x), x \hookrightarrow y, \text{size} \geq \beta \mid x, y \in X, \beta \in [0, \alpha]\}$. $\text{Bool}(\text{Core}(X, \alpha))$ is the set of Boolean combinations of formulae from $\text{Core}(X, \alpha)$, whereas $\text{Conj}(\text{Core}(X, \alpha))$ is the set of conjunctions of literals built upon $\text{Core}(X, \alpha)$ (a literal being a core formula or its negation). Given $\varphi = L_1 \wedge \dots \wedge L_n \in \text{Conj}(\text{Core}(X, \alpha))$, every L_i being a literal, $\text{Lt}(\varphi) \stackrel{\text{def}}{=} \{L_1, \dots, L_n\}$. $\psi \subseteq_{\text{Lt}} \varphi$ stands for $\text{Lt}(\psi) \subseteq \text{Lt}(\varphi)$. We write $\chi \subseteq_{\text{Lt}} \{\varphi \mid \psi\}$, $\{\varphi \mid \psi\} \subseteq_{\text{Lt}} \chi$ and $\chi \subseteq_{\text{Lt}} \{\varphi; \psi\}$ for “ $\chi \subseteq_{\text{Lt}} \varphi$ or $\chi \subseteq_{\text{Lt}} \psi$ ”, “ $\varphi \subseteq_{\text{Lt}} \chi$ or $\psi \subseteq_{\text{Lt}} \chi$ ”, and “ $\chi \subseteq_{\text{Lt}} \varphi$ and $\chi \subseteq_{\text{Lt}} \psi$ ”, respectively.

Example. To show the flavour of the axioms and the rules, Figure 1 displays a proof in $\mathcal{H}_C(*, *)$. In the proof, a line “ $j \mid \chi$ A, i_1, \dots, i_k ” states that χ is a theorem denoted by the index j and derivable by the axiom or the rule A . If A is a rule, the indices $i_1, \dots, i_k < j$ denote

the theorems used as premises in order to derive χ . The example uses the rule ***-Adj**, which together with ***-Adj** states that the $*$ and \ast are adjoint operators, and the axiom **(A₁₉^{*})**, stating that $\text{card}(\text{dom}(h)) \leq \beta_1 + \beta_2$ holds whenever a heap h can be split into two subheaps that have less than $\beta_1 + 1$ and $\beta_2 + 1$ memory cells, respectively. We also use the following theorems and rules, which can be shown derivable/admissible in the forthcoming calculus:

$$(\wedge\text{Er}) \quad \psi \wedge \varphi \Rightarrow \varphi \quad (\neg\neg\text{E}) \quad \neg\neg\varphi \Rightarrow \varphi \quad \ast\text{-Itr}: \frac{\varphi \Rightarrow \varphi' \quad \psi \Rightarrow \psi'}{\varphi \ast \psi \Rightarrow \varphi' \ast \psi'} \quad \Rightarrow\text{-Tr}: \frac{\varphi \Rightarrow \chi \quad \chi \Rightarrow \psi}{\varphi \Rightarrow \psi}$$

3.1 A simple calculus for the core formulae

To axiomatise $\text{SL}(*, \ast)$, we start by introducing the proof system \mathcal{H}_C (presented in System 1) dedicated to Boolean combinations of core formulae. \mathcal{H}_C and all the subsequent proof systems contain the axiom schemata and modus ponens for the propositional calculus. The axioms $I_i^?$ in System n are necessary for the fragment the System n governs, but are admissible when the axioms/rules from the System $n+1$ are present. In **(A₂^C)**, $\varphi[y \leftarrow x]$ is the formula obtained from φ by replacing with x every occurrence of y . Let (s, h) be a memory state. The axioms state that $=$ is an equivalence relation (first two axioms), $h(s(x)) = s(y)$ implies $s(x) \in \text{dom}(h)$ (axiom **(A₃^C)**) and that h is a (partial) function (axiom **(A₄^C)**). Furthermore, there are two intermediate axioms about size formulae: **(I₅^C)** states that if $\text{dom}(h)$ has at least $\beta + 1$ elements, then it has at least β elements, whereas **(I₆^C)** states that if there are β distinct memory cells corresponding to program variables, then indeed $\text{dom}(h) \geq \beta$. It is easy to check that \mathcal{H}_C is sound (right-to-left direction of Theorem 2, below). In order to establish its completeness with respect to $\text{Bool}(\text{Core}(\mathbf{X}, \alpha))$, we first establish that \mathcal{H}_C is complete for a fragment of $\text{Bool}(\text{Core}(\mathbf{X}, \alpha))$, made of *core types*. Let $\mathbf{X} \subseteq_{\text{fin}} \text{VAR}$, $\alpha \in \mathbb{N}^+$ and $\hat{\alpha} = \alpha + \text{card}(\mathbf{X})$. We write $\text{CoreTypes}(\mathbf{X}, \alpha)$ to denote the set of *core types* defined by $\{\varphi \in \text{Conj}(\text{Core}(\mathbf{X}, \hat{\alpha})) \mid \forall \psi \in \text{Core}(\mathbf{X}, \hat{\alpha}), \{\psi \mid \neg\psi\} \subseteq_{\text{Lt}} \varphi, \text{ and } (\psi \wedge \neg\psi) \not\subseteq_{\text{Lt}} \varphi\}$. Every formula in this set is a conjunction having exactly one literal built upon ψ for every $\psi \in \text{Core}(\mathbf{X}, \hat{\alpha})$.

► **Lemma 1.** *Let $\varphi \in \text{CoreTypes}(\mathbf{X}, \alpha)$. We have $\neg\varphi$ is valid iff $\vdash_{\mathcal{H}_C} \neg\varphi$.*

By classical reasoning, one can show that every $\varphi \in \text{Bool}(\text{Core}(\mathbf{X}, \alpha))$ is provably equivalent to a disjunction of core types. Together with Lemma 1, this implies that \mathcal{H}_C is complete.

► **Theorem 2.** (Adequacy) *A Boolean combination of core formulae φ is valid iff $\vdash_{\mathcal{H}_C} \varphi$.*

3.2 A constructive elimination of $*$ to axiomatise $\text{SL}(*, \text{alloc})$

We enrich \mathcal{H}_C by adding axioms and inference rule that handle $*$ (System 2). The axioms deal with the commutative monoid properties of $(*, \text{emp})$ and its distributivity over \vee (as for Boolean BI, see e.g. [21]). In **(A₁₄^{*})**, the notation $\varphi \llbracket \mathcal{B} \rrbracket$ refers to the axiom schema φ assuming that the Boolean condition \mathcal{B} holds. The rule ***-Intro** states that logical equivalence is a congruence for $*$. This allows us to remove the intermediate axioms **(I₅^C)** and **(I₆^C)** from the proof system. Hence, we call $\mathcal{H}_C(*)$ the proof system obtained from \mathcal{H}_C by adding all schemata from System 2 and removing **(I₅^C)** and **(I₆^C)**. It is easy to check that $\mathcal{H}_C(*)$ is sound. More importantly, $\mathcal{H}_C(*)$ enjoys the $*$ elimination property with respect to core types.

► **Lemma 3.** *Let φ and ψ in $\text{CoreTypes}(\mathbf{X}, \alpha)$. There is a conjunction of core formulae literals $\chi \in \text{Conj}(\text{Core}(\mathbf{X}, 2\alpha))$ such that $\vdash_{\mathcal{H}_C(*)} \varphi * \psi \Leftrightarrow \chi$.*

Proof. (sketch) Let $\varphi, \psi \in \text{CoreTypes}(\mathbf{X}, \alpha)$. If φ is unsatisfiable, then $\vdash_{\mathcal{H}_C} \varphi \Rightarrow \perp$, by Lemma 1. By the rule ***-Intro** and the axiom **(I₁₀^{*})**, we get $\vdash_{\mathcal{H}_C(*)} \varphi * \psi \Rightarrow \perp$ and we take $\chi = \perp$.

Assume now both φ and ψ to be satisfiable. Then $\varphi * \psi$ can be shown provably equivalent to:

$$\begin{aligned} & \bigwedge \{x \sim y \subseteq_{\text{Lt}} \{\varphi \mid \psi\} \mid \sim \in \{=, \neq\}\} \quad \wedge \quad \bigwedge \{\text{alloc}(x) \subseteq_{\text{Lt}} \{\varphi \mid \psi\}\} \\ & \wedge \bigwedge \{x \leftrightarrow y \subseteq_{\text{Lt}} \{\varphi \mid \psi\}\} \quad \wedge \quad \bigwedge \{\neg \text{alloc}(x) \subseteq_{\text{Lt}} \{\varphi; \psi\}\} \\ & \wedge \bigwedge \{\perp \mid \text{alloc}(x) \subseteq_{\text{Lt}} \{\varphi; \psi\}\} \quad \wedge \quad \bigwedge \{\neg x \leftrightarrow y \mid \text{alloc}(x) \wedge \neg x \leftrightarrow y \subseteq_{\text{Lt}} \{\varphi \mid \psi\}\} \\ & \wedge \bigwedge \left\{ \text{size} \geq \beta_1 + \beta_2 \mid \begin{array}{l} \text{size} \geq \beta_1 \subseteq_{\text{Lt}} \varphi \\ \text{size} \geq \beta_2 \subseteq_{\text{Lt}} \psi \end{array} \right\} \quad \wedge \quad \bigwedge \left\{ \neg \text{size} \geq \beta_1 + \beta_2 + 1 \mid \begin{array}{l} \neg \text{size} \geq \beta_1 \subseteq_{\text{Lt}} \varphi \\ \neg \text{size} \geq \beta_2 \subseteq_{\text{Lt}} \psi \end{array} \right\} \end{aligned}$$

This equivalence is reminiscent to the one in [20, Lemma 3] that is proved semantically. In a way, because $\mathcal{H}_C(*)$ will reveal to be complete, the restriction of the proof of [20, Lemma 3] to $\text{SL}(*, \text{alloc})$ can actually be replayed completely syntactically within $\mathcal{H}_C(*)$. ◀

By the distributivity axiom (**I₉^{*}**), this result is extended from core types to arbitrary Boolean combinations of core formulae. $\mathcal{H}_C(*)$ is therefore complete for $\text{SL}(*, \text{alloc})$, i.e. the logic obtained from $\text{SL}(*, *)$ by removing $*$ and adding the formulae $\text{alloc}(x)$ (only core formulae requiring $*$). Then, to prove that a formula $\varphi \in \text{SL}(*, \text{alloc})$ is valid, we repeatedly apply the $*$ elimination bottom-up obtaining a Boolean combination of core formulae ψ that is equivalent to φ . We rely on the completeness of \mathcal{H}_C (Theorem 2) to prove that ψ is valid.

► **Theorem 4.** *A formula φ in $\text{SL}(*, \text{alloc})$ is valid iff $\vdash_{\mathcal{H}_C(*)} \varphi$.*

3.3 A constructive elimination of $*$ to axiomatise $\text{SL}(*, *)$

The proof system $\mathcal{H}_C(*, *)$ is defined as $\mathcal{H}_C(*)$ augmented with the axioms and inference rules from System 3 dedicated to separating implication. The axioms involving \oplus (kind of dual of $*$ introduced in Section 2) express that it is always possible to extend a given heap with an extra cell, and that the address and the content of this cell can be fixed arbitrarily (provided it is not already allocated). The adjunction rules are from the Hilbert-style axiomatisation of Boolean BI [21, Section 2]. One can observe that the axioms (**I₉^{*}**), (**I₁₀^{*}**), (**I₁₂^{*}**) and (**I₁₃^{*}**) are derivable in $\mathcal{H}_C(*, *)$. It is easy to check that $\mathcal{H}_C(*, *)$ is sound. Analogously, $\mathcal{H}_C(*, *)$ enjoys the $*$ elimination property, stated below by means of \oplus .

► **Lemma 5.** *Let φ and ψ in $\text{CoreTypes}(\mathbf{X}, \alpha)$. There is a conjunction of core formulae literals $\chi \in \text{Conj}(\text{Core}(\mathbf{X}, \alpha))$ such that $\vdash_{\mathcal{H}_C(*, *)} (\varphi \oplus \psi) \Leftrightarrow \chi$.*

Proof. (sketch) If either φ or ψ is unsatisfiable, then one can show that $\vdash_{\mathcal{H}_C(*, *)} \varphi \oplus \psi \Rightarrow \perp$. Otherwise, $\varphi \oplus \psi$ can be shown provably equivalent to

$$\begin{aligned} & \bigwedge \{x \sim y \subseteq_{\text{Lt}} \{\varphi \mid \psi\} \mid \sim \in \{=, \neq\}\} \quad \wedge \quad \bigwedge \{\neg \text{alloc}(x) \subseteq_{\text{Lt}} \psi\} \quad \wedge \quad \bigwedge \{\neg x \leftrightarrow y \subseteq_{\text{Lt}} \psi\} \\ & \wedge \bigwedge \left\{ \text{alloc}(x) \mid \begin{array}{l} \neg \text{alloc}(x) \subseteq_{\text{Lt}} \varphi \\ \text{alloc}(x) \subseteq_{\text{Lt}} \psi \end{array} \right\} \quad \wedge \quad \bigwedge \left\{ x \leftrightarrow y \mid \begin{array}{l} \neg \text{alloc}(x) \subseteq_{\text{Lt}} \varphi \\ x \leftrightarrow y \subseteq_{\text{Lt}} \psi \end{array} \right\} \quad \wedge \quad \bigwedge \{\neg \text{alloc}(x) \mid \text{alloc}(x) \subseteq_{\text{Lt}} \varphi\} \\ & \wedge \bigwedge \left\{ \text{size} \geq \beta_2 + 1 + \beta_1 \mid \begin{array}{l} \neg \text{size} \geq \beta_1 \subseteq_{\text{Lt}} \varphi \\ \text{size} \geq \beta_2 \subseteq_{\text{Lt}} \psi \end{array} \right\} \quad \wedge \quad \bigwedge \left\{ \perp \mid \begin{array}{l} x \leftrightarrow y \subseteq_{\text{Lt}} \varphi \\ \neg x \leftrightarrow y \subseteq_{\text{Lt}} \psi \end{array} \right\} \quad \wedge \quad \bigwedge \left\{ \perp \mid \begin{array}{l} \text{alloc}(x) \wedge \neg x \leftrightarrow y \subseteq_{\text{Lt}} \varphi \\ x \leftrightarrow y \subseteq_{\text{Lt}} \psi \end{array} \right\} \\ & \wedge \bigwedge \left\{ \neg \text{size} \geq \beta_2 + \beta_1 \mid \begin{array}{l} \text{size} \geq \beta_1 \subseteq_{\text{Lt}} \varphi \\ \neg \text{size} \geq \beta_2 \subseteq_{\text{Lt}} \psi \end{array} \right\} \quad \wedge \quad \bigwedge \left\{ \perp \mid \begin{array}{l} \text{alloc}(x) \subseteq_{\text{Lt}} \varphi \\ \neg \text{alloc}(x) \subseteq_{\text{Lt}} \psi \end{array} \right\} \end{aligned}$$

where $a \dot{-} b$ stands for $a - b$ if $a \geq b$, 0 otherwise. Again, this equivalence is reminiscent to the one in [20, Lemma 4] proved semantically. Herein, the proof is completely syntactical. ◀

Again, this result for core types can be extended to arbitrary Boolean combinations of core formulae, as we show that the distributivity of \oplus over disjunctions is provable in $\mathcal{H}_C(*, *)$. As a consequence of this development, we achieve one of the main results of the paper.

► **Theorem 6.** *$\mathcal{H}_C(*, *)$ is sound and complete for $\text{SL}(*, *)$.*

What's next? To provide further evidence that our method is robust, we shall apply it to axiomatise other separation logics, for instance by adding the list segment predicate \mathbf{ls} [2] (or inductive predicates in general) or first-order quantification. Of course, the set of valid formulae must be r.e., which discards any attempt with $\mathbf{SL}(*, \neg, \mathbf{ls})$ or with the first-order version of $\mathbf{SL}(*, \neg)$ [15, 4]. In Section 4, we introduce an extension of $\mathbf{SL}(*, \mathbf{ls})$ and we axiomatise it with our method, whose main ingredients are recalled below.

3.4 Ingredients of the method

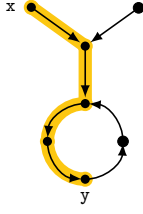
The Hilbert-style axiomatisation of $\mathbf{SL}(*, \neg)$ has culminated with Theorem 6 that states the adequateness of $\mathcal{H}_C(*, \neg)$. Below, we would like to recapitulate the key ingredients of the proposed method, not only to provide a vade-mecum for axiomatising other separation logics (which we illustrate on the newly introduced logic $\mathbf{SL}(*, \exists: \rightsquigarrow)$ in Section 4), but also to identify the essential features and where variations are still possible.

Core formulae. To axiomatise $\mathbf{SL}(*, \neg)$ internally, the core formulae have played an essential role. The main properties of these formulae is that their Boolean combinations capture the full logic $\mathbf{SL}(*, \neg)$ [29] and all the core formulae can be expressed in $\mathbf{SL}(*, \neg)$. Generally speaking, our axiom system naturally leads to a form of constructive completeness, as advocated in [19, 31]: the axiomatisation provides proof-theoretical means to transform any formula into an equivalent Boolean combination of core formulae, and it contains also a part dedicated to the derivation of valid Boolean combinations of core formulae (understood as a syntactical fragment of $\mathbf{SL}(*, \neg)$). What is specific to each logic is the design of the set of core formulae and in the case of $\mathbf{SL}(*, \neg)$, this was already known since [29].

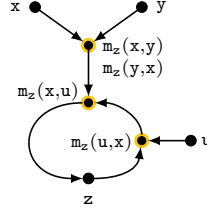
Big-step vs. small-step axiom schemas. $\mathcal{H}_C(*, \neg)$ simulates the bottom-up elimination of separating connectives (see Lemmata 3 and 5) when the arguments are two Boolean combinations of core formulae. To do so, $\mathcal{H}_C(*, \neg)$ contains axiom schemas that perform such an elimination in multiple “small-step” derivations, e.g. by deriving a single $\mathbf{alloc}(x)$ predicate from $\mathbf{alloc}(x) * \top$ (axiom (\mathbf{I}_{12}^*)). Alternatively, it would have been possible to include “big-step” axiom schemas that, given the two Boolean combinations of core formulae, derive the equivalent formula in one single derivation step. Instances of this are given in the proof sketch of Lemma 3, and later in Section 4 (axiom $(*_{48})$). The main difference is that small-step axioms provide a simpler understanding of the key properties of the logic.

4 How to axiomatise internally the separation logic $\mathbf{SL}(*, \exists: \rightsquigarrow)$

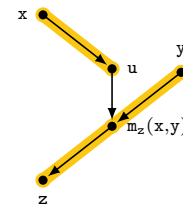
Though core formulae are handful for several existing separation logics, see e.g. recently [15, 32, 20], we would like to test our method with first-order quantification and reachability predicates, standard features in specifications. However, $\mathbf{SL}(*, \neg, \mathbf{ls})$ is already known to be non-finitely axiomatisable, see the developments in Section 2. So, we need to downgrade our ambitions and we suggest to consider a new logic with guarded quantification and \mathbf{ls} and this is $\mathbf{SL}(*, \exists: \rightsquigarrow)$ presented below. Note that the idea of having guarded quantification with second-order features is not new, see e.g. in [24] extensions of the guarded fragment of first-order logic with fixed points, but herein, this is done in the framework of separation logics and their axiomatisation. In short, we introduce the new separation logic $\mathbf{SL}(*, \exists: \rightsquigarrow)$ that admits the connective $*$, the list segment predicate \mathbf{ls} (implicitly) and a guarded form of first-order quantification involving \mathbf{ls} . It contains the symbolic heap fragment [2, 11] but



■ Figure 2 Path quantifier.



■ Figure 3 Meet points.



■ Figure 4 Sees predicates.

also richer logics such as $\text{SL}(*, \text{reach}^+)$ (see e.g. [15]). As a by-product of our completeness proof, we are able to characterise the complexity of the satisfiability problem for $\text{SL}(*, \exists: \rightsquigarrow)$.

4.1 A guarded logic with ls : $\text{SL}(*, \exists: \rightsquigarrow)$

Formulae of $\text{SL}(*, \exists: \rightsquigarrow)$ are defined according to the grammar below (where $x, y, z \in \text{VAR}$):

$$\varphi := x = y \mid x \hookrightarrow y \mid \text{emp} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi * \varphi \mid \exists z: \langle x \rightsquigarrow y \rangle \varphi$$

All the syntactic ingredients are standard except the quantifier (denoted with $\exists: \rightsquigarrow$). Intuitively (the formal definition is provided below), $\exists z: \langle x \rightsquigarrow y \rangle \varphi$ is a guarded form of quantification that is intended to hold true whenever y is reachable from x in at least one step, and there is a location ℓ along the minimal path between x and y so that the formula φ holds whenever ℓ is assigned to z . Figure 2 highlights the possible assignments of z (arrows represent the heap). Given a heap h and $\ell_1, \ell_2 \in \text{LOC}$, we define $h[\ell_1, \ell_2]$ as the set of locations in the shortest path from ℓ_1 to ℓ_2 (ℓ_2 possibly excluded). Formally:

$$h[\ell_1, \ell_2] \stackrel{\text{def}}{=} \left\{ \ell \in \text{LOC} \mid \begin{array}{l} \text{there are } \delta_1 \geq 0 \text{ and } \delta_2 \geq 1 \text{ such that } h^{\delta_1}(\ell_1) = \ell, \\ h^{\delta_2}(\ell) = \ell_2 \text{ and, for every } \delta \in [1, \delta_1 + \delta_2 - 1], h^\delta(\ell_1) \neq \ell_2 \end{array} \right\}$$

For example, $h[\ell, \ell] = \emptyset$ holds iff ℓ is not in a cycle. Otherwise, $h[\ell, \ell]$ contains all the locations in the cycle containing ℓ . By definition, the minimal paths are preserved when considering heap extensions. Then, the satisfaction relation \models is completed with

$$(s, h) \models \exists z: \langle x \rightsquigarrow y \rangle \varphi \stackrel{\text{def}}{\iff} h[s(x), s(y)] \neq \emptyset \text{ and } \exists \ell \in h[s(x), s(y)] \cup \{s(y)\} \text{ s.t. } (s[z \leftarrow \ell], h) \models \varphi.$$

We define $\forall z: \langle x \rightsquigarrow y \rangle \varphi \stackrel{\text{def}}{=} \neg \exists z: \langle x \rightsquigarrow y \rangle \neg \varphi$. In a separation logic *lingua* admitting first-order quantification of program variables over the set of locations LOC , and a predicate $\text{reach}^+(x, y)$ (reachability in at least one step, as in [15]), the formula $\exists z: \langle x \rightsquigarrow y \rangle \varphi$ is equivalent to

$$\text{reach}^+(x, y) \wedge \exists z \varphi \wedge (z = x \vee z = y \vee ((\text{reach}^+(x, z) \wedge \neg \text{reach}^+(x, y)) * \text{reach}^+(z, y))).$$

Obviously, $\text{SL}(*, \exists: \rightsquigarrow)$ does not allow unrestricted first-order quantification but it can faithfully define the reachability predicates classically studied in separation logic [15, 38]. $\text{reach}^+(x, y)$ is definable as $\exists z: \langle x \rightsquigarrow y \rangle \top$, and allows us to define $\text{ls}(x, y)$ and $\text{reach}(x, y)$ as shown in [15]: $\text{ls}(x, y) \stackrel{\text{def}}{=} (x = y \wedge \text{emp}) \vee (x \neq y \wedge \text{reach}^+(x, y) \wedge \neg(\neg \text{emp} * \text{reach}^+(x, y)))$, whereas $\text{reach}(x, y) \stackrel{\text{def}}{=} x = y \vee \text{reach}^+(x, y)$. There are two features of $\text{SL}(*, \exists: \rightsquigarrow)$, we would like to emphasize. First, it is possible to enforce a heap domain of exponential size. Indeed, we define the formula $R^n(x, y)$ of size linear in n , but enforcing the existence of a path of length at least 2^n between two distinct locations corresponding to x and y , respectively. $R^0(x, y) \stackrel{\text{def}}{=} x \neq y \wedge \exists z: \langle x \rightsquigarrow y \rangle \top$, whereas for $n \geq 0$, $R^{n+1}(x, y)$ is defined as

$$x \neq y \wedge \exists z: \langle x \rightsquigarrow y \rangle \forall z': \langle x \rightsquigarrow y \rangle \forall z'': \langle x \rightsquigarrow y \rangle ((z' = x \wedge z'' = z) \vee (z' = z \wedge z'' = y) \Rightarrow R^n(z', z'')).$$

Nevertheless, in Section 4.6 we show how the satisfiability and validity problems for $\text{SL}(*, \exists: \rightsquigarrow)$ are in PSPACE. Another interesting feature of $\text{SL}(*, \exists: \rightsquigarrow)$ is illustrated by its ability to state that from two locations corresponding to program variables (say x, y), it is possible to reach a

different location, which in turn reaches another location corresponding to a program variable (say z). This can be done with the formula $\exists w: \langle x \rightsquigarrow z \rangle (\text{reach}^+(y, w) \wedge \bigwedge_{v \in \{x, y, z\}} w \neq v)$. Thus, the logic is able to express that two paths meet at a specific location. This naturally leads to the notion of meet-points, introduced next in order to define the core formulae for $\text{SL}(*, \exists: \rightsquigarrow)$.

4.2 Core formulae are back!

In order to axiomatise internally $\text{SL}(*, \exists: \rightsquigarrow)$ with our method, we need to possess a set of core formulae that captures $\text{SL}(*, \exists: \rightsquigarrow)$. Below, we design such core formulae and establish its appropriateness. They make intensive use of meet-point terms, a concept introduced in [15] but that will play a crucial role herein. Informally, given a memory state (s, h) , a meet-point between $s(x)$ and $s(y)$ leading to $s(z)$ is a location ℓ such that (I) ℓ reaches $s(z)$, (II) both locations $s(x)$ and $s(y)$ reach ℓ , and (III) there is no location ℓ' satisfying these properties and reachable from $s(x)$ in strictly fewer steps. A meet-point term of the form $m_z(x, y)$, where $x, y, z \in \text{VAR}$, is then an expression that, given a memory state (s, h) , is intended to be interpreted by a meet-point between $s(x)$ and $s(y)$ leading to $s(z)$ (if it exists). Figure 3 shows some of the meet-points between x and other program variables, highlighting their distribution in a memory state. In particular, notice how in the figure, $m_z(x, u)$ is different from $m_z(u, x)$, which happens because of the condition (III) and as the two corresponding locations are in a cycle. We call this type of meet-points *asymmetric*. We now formalise these concepts. Given $X \subseteq \text{VAR}$, we write $\mathbb{T}(X)$ to denote the set $X \cup \{m_z(x, y) \mid x, y, z \in X\}$. Elements of $\mathbb{T}(\text{VAR})$ are called *terms*. Expressions $m_z(x, y)$ are syntactic constructs called *meet-point terms*. Terms are denoted with $\mathfrak{t}, \mathfrak{t}_1, \mathfrak{t}_2, \dots$, when we do not need to distinguish between variables and meet-point terms. To give a semantics to these objects, we interpret the terms by means of the interpretation function $\llbracket \cdot \rrbracket_{s, h}: \mathbb{T}(\text{VAR}) \rightarrow \text{LOC}$ s.t. $\llbracket x \rrbracket_{s, h} \stackrel{\text{def}}{=} s(x)$ for $x \in \text{VAR}$, and $\llbracket m_z(x, y) \rrbracket_{s, h}$ is defined and takes the value ℓ iff there are $\delta_1, \delta_2 \geq 0$ s.t.

- $h^{\delta_1}(s(x)) = h^{\delta_2}(s(y)) = \ell$ and there is $\delta \geq 0$ such that $h^\delta(\ell) = s(z)$;
- for every $\delta'_1 \in [0, \delta_1 - 1]$ and $\delta'_2 \geq 0$, $h^{\delta'_1}(s(x)) \neq h^{\delta'_2}(s(y))$.

One last object is needed in order to define the core formulae. Given a memory state (s, h) and a finite set of pairs of terms $\mathfrak{P} \subseteq_{\text{fin}} \mathbb{T}(\text{VAR}) \times \mathbb{T}(\text{VAR})$, we write $\text{Rem}_{s, h}^{\mathfrak{P}}$ to denote the subset of $\text{dom}(h)$ made of the locations that are not in the path between two locations corresponding to terms in a pair of \mathfrak{P} . Formally: $\text{Rem}_{s, h}^{\mathfrak{P}} \stackrel{\text{def}}{=} \text{dom}(h) \setminus \left(\bigcup_{(\mathfrak{t}_1, \mathfrak{t}_2) \in \mathfrak{P}} h[\llbracket \mathfrak{t}_1 \rrbracket_{s, h}, \llbracket \mathfrak{t}_2 \rrbracket_{s, h}] \right)$.

The *core formulae* are expressions of the form: $\mathfrak{t}_1 = \mathfrak{t}_2$, $\text{sees}_{\mathfrak{T}}(\mathfrak{t}_1, \mathfrak{t}_2) \geq \beta + 1$ and $\text{rem}_{\mathfrak{P}} \geq \beta$, where $\mathfrak{t}_1, \mathfrak{t}_2 \in \mathbb{T}(\text{VAR})$, $\mathfrak{T} \subseteq_{\text{fin}} \mathbb{T}(\text{VAR})$, $\mathfrak{P} \subseteq_{\text{fin}} \mathbb{T}(\text{VAR}) \times \mathbb{T}(\text{VAR})$ and $\beta \in \mathbb{N}$. We write $\text{sees}_{\mathfrak{T}}(\mathfrak{t}_1, \mathfrak{t}_2)$ for $\text{sees}_{\mathfrak{T}}(\mathfrak{t}_1, \mathfrak{t}_2) \geq 1$. The satisfaction relation \models is extended to core formulae:

- $(s, h) \models \mathfrak{t}_1 = \mathfrak{t}_2 \stackrel{\text{def}}{\iff} \llbracket \mathfrak{t}_1 \rrbracket_{s, h} = \llbracket \mathfrak{t}_2 \rrbracket_{s, h}$; ■ $(s, h) \models \text{rem}_{\mathfrak{P}} \geq \beta \stackrel{\text{def}}{\iff} \text{card}(\text{Rem}_{s, h}^{\mathfrak{P}}) \geq \beta$;
- $(s, h) \models \text{sees}_{\mathfrak{T}}(\mathfrak{t}_1, \mathfrak{t}_2) \geq \beta \stackrel{\text{def}}{\iff}$ there is $\delta \geq \beta$ such that $h^\delta(\llbracket \mathfrak{t}_1 \rrbracket_{s, h}) = \llbracket \mathfrak{t}_2 \rrbracket_{s, h}$ and for all $\delta' \in [1, \delta - 1]$, $h^{\delta'}(\llbracket \mathfrak{t}_1 \rrbracket_{s, h}) \notin \{\llbracket \mathfrak{t}_2 \rrbracket_{s, h}\} \cup \{\llbracket \mathfrak{t} \rrbracket_{s, h} \mid \mathfrak{t} \in \mathfrak{T}\}$.

As earlier in Section 3, we write $\text{Core}(X, \alpha)$ to denote the set of core formulae restricted to terms from $\mathbb{T}(X)$, where $X \subseteq_{\text{fin}} \text{VAR}$ and β is bounded above by α . In order to become more familiar with these core formulae, let us consider the memory state (s, h) outlined in Figure 4. Since both $s(x)$ and $s(y)$ reach $s(z)$, $\llbracket m_z(x, y) \rrbracket_{s, h}$ is defined, or alternatively $(s, h) \models m_z(x, y) = m_z(x, y)$. Therefore, we have that $(s, h) \models \text{sees}_{\emptyset}(x, m_z(x, y))$. We also note that $s(u)$ is a location in the minimal path from $s(x)$ to $\llbracket m_z(x, y) \rrbracket_{s, h}$. However, as $s(u)$ is distinct from these two locations, we conclude that $(s, h) \models \neg \text{sees}_{\{u\}}(x, m_z(x, y))$. Lastly, let us take for example the sets of locations corresponding to the two paths highlighted in yellow: $h[s(x), s(u)[$ and $h[s(y), s(z)[$. The location $s(u)$ does not belong to any of these sets.

As it is in $\text{dom}(h)$, we conclude that $(s, h) \models \text{rem}_{\{(x,u),(y,z)\}} \geq 1$.

Expressing core formulae in $\text{SL}(*, \exists: \rightsquigarrow)$. A crucial point for axiomatising $\text{SL}(*, \rightsquigarrow)$ is that every core formula is a mere abbreviation for a formula of the logic. This is the property that leads to an *internal* axiomatisation. The same holds for $\text{SL}(*, \exists: \rightsquigarrow)$ as one can show that every core formula can be defined in $\text{SL}(*, \exists: \rightsquigarrow)$ and, in the forthcoming axiomatisation, should be considered as an abbreviation. For example, the formula $\text{sees}_\theta(x, y) \geq \beta$ can be shown equivalent to $(\text{strict}(\text{reach}^+(x, y)) \wedge \text{size} \geq \beta) * \top$, where $\text{strict}(\varphi)$ is a shortcut for $\varphi \wedge \neg(\neg \text{emp} * \varphi)$ and states that φ holds in the current model, say (s, h) but does not hold in any submodel (i.e. in (s, h') where $h' \sqsubset h$). Similarly, $x = m_u(y, z)$ is equivalent to

$$\text{reach}(x, u) \wedge (\text{reach}(y, x) * \text{reach}(z, x)) \wedge (\text{reach}^+(x, x) \Rightarrow (\text{reach}(y, x) * \text{reach}^+(x, x))),$$

whereas $m_z(x, y) = m_w(u, v)$ is $\exists j: \langle x \rightsquigarrow z \rangle (m_z(x, y) = j \wedge j = m_w(u, v))$, where $j \notin \{x, y, z, u, v, w\}$.

► **Lemma 7.** *Every core formula is logically equivalent to a formula of $\text{SL}(*, \exists: \rightsquigarrow)$.*

4.3 Axiomatisation of the logic of core formulae

As done in Section 3, to axiomatise $\text{SL}(*, \exists: \rightsquigarrow)$ we start by extending the axiom system for the propositional calculus in order to obtain the proof system \mathcal{H}_C dedicated to Boolean combinations of core formulae. The axioms, presented in System 4, are divided into axioms for equalities between terms, whose name is of the form $=_i^C$; axioms essentially about the predicates sees , whose name is of the form s_i^C ; and axioms essentially about the predicates rem , whose name is of the form r_i^C . In order to obtain this axiom system, the two main difficulties (which lead to very technical formulae) are given by the distribution of meet-points within the memory state and the axiomatisation of the predicates sees . For the former, it is important to distinguish between symmetric and asymmetric meet-points. For this reason, System 4 uses the formulae $\text{def}(m_z(x, y)) \stackrel{\text{def}}{=} m_z(x, y) = m_z(x, y)$, which checks if a meet-point is defined, $\text{sym}(m_z(x, y)) \stackrel{\text{def}}{=} m_z(x, y) = m_z(y, x)$ for symmetric meet-points, and $\text{asym}(m_z(x, y)) \stackrel{\text{def}}{=} \text{def}(m_z(x, y)) \wedge \neg \text{sym}(m_z(x, y))$ for asymmetric ones. The definition of these formulae, as well as the ones below, is extended on a variable $x \in \text{VAR}$ simply by replacement with the meet-point $m_x(x, x)$ (the two terms are always equivalent, see the axiom $(=1^C)$). So, for example $\text{def}(x)$ is defined as $\text{def}(m_x(x, x))$. For sees predicates, an important distinction is given by terms corresponding to different locations in the same tree (no cycle is involved) and terms that correspond to different locations in the same cycle. Hence, we define the abbreviations $\text{before}(t_1, t_2)$ and $\text{samecycle}(t_1, t_2)$ with the following meanings:

$(s, h) \models \text{before}(t_1, t_2)$ iff $\llbracket t_1 \rrbracket_{s,h} \neq \llbracket t_2 \rrbracket_{s,h}$ and, there is a path from $\llbracket t_1 \rrbracket_{s,h}$ to $\llbracket t_2 \rrbracket_{s,h}$ s.t.
the only location on the path that may belong to a cycle is $\llbracket t_2 \rrbracket_{s,h}$.

$(s, h) \models \text{samecycle}(t_1, t_2)$ iff $\llbracket t_1 \rrbracket_{s,h} \neq \llbracket t_2 \rrbracket_{s,h}$ and there is a cycle with both $\llbracket t_1 \rrbracket_{s,h}$ and $\llbracket t_2 \rrbracket_{s,h}$.

They are defined as follows for meet-points (and extended for $x \in \text{VAR}$ as shown for $\text{def}(x)$)

- The formulae $\text{before}(m_z(x, y), m_v(x, u))$ and $\text{before}(m_z(y, x), m_v(x, u))$ are both defined as $\text{sym}(m_z(x, y)) \wedge \text{def}(m_v(x, y)) \wedge \text{def}(m_v(x, u)) \wedge m_z(x, y) \neq m_v(x, u) \wedge m_z(x, y) \neq m_v(y, u)$;
- $\text{before}(m_z(x, y), m_w(u, v)) \stackrel{\text{def}}{=} \bigvee_{a \in \{u, v\}} \text{before}(m_z(x, y), m_w(x, a)) \wedge m_w(x, a) = m_w(u, v)$;
- $\text{samecycle}(m_z(x, y), m_w(u, v)) \stackrel{\text{def}}{=} m_z(x, y) = m_w(x, u) \wedge m_w(u, v) = m_z(u, x) \wedge \text{asym}(m_z(x, u))$.

We write $\mathfrak{t} \in \mathfrak{T}$ (finite set of terms \mathfrak{T}) to denote $\bigvee_{t_2 \in \mathfrak{T}} \mathfrak{t} = t_2$. Like the axiom (A_2^C) , the axiom $(=3^C)$ performs a substitution of every occurrence of t_1 with t_2 . We have to be careful here: when substituting a variable x with a meet-point $m_u(y, z)$, we only substitute the occurrences of x that are not inside meet-point terms. For example, $\text{sees}_{\{x, m_x(x, x)\}}(x, m_x(x, x)) [x \leftarrow m_u(y, z)]$

19:12 Internal Calculi for Separation Logics

is equal to $\text{sees}_{\{\mathfrak{m}_u(y,z), \mathfrak{m}_x(x,x)\}}(\mathfrak{m}_u(y,z), \mathfrak{m}_x(x,x))$. By way of example, let us explain why all the instances of the axiom $(=\mathfrak{C})$ are valid. Suppose $(s, h) \models \text{def}(\mathfrak{m}_z(x, y)) \wedge \text{def}(\mathfrak{m}_u(x, y))$. Since $\llbracket \mathfrak{m}_z(x, y) \rrbracket_{s,h}$ is defined (say equal to ℓ), there are $\delta_1, \delta_2 \geq 0$ such that

- $h^{\delta_1}(s(x)) = h^{\delta_2}(s(y)) = \ell$ and there is $\delta \geq 0$ such that $h^\delta(\ell) = s(z)$;
- for every $\delta'_1 \in [0, \delta_1 - 1]$ and $\delta'_2 \geq 0$, $h^{\delta'_1}(s(x)) \neq h^{\delta'_2}(s(y))$.

Similarly, as $\llbracket \mathfrak{m}_u(x, y) \rrbracket_{s,h}$ is also defined (say equal to ℓ'), there are also $\gamma_1, \gamma_2 \geq 0$ such that

- $h^{\gamma_1}(s(x)) = h^{\gamma_2}(s(y)) = \ell'$ and there is $\delta' \geq 0$ such that $h^{\delta'}(\ell') = s(u)$;
- for every $\gamma'_1 \in [0, \gamma_1 - 1]$ and $\gamma'_2 \geq 0$, $h^{\gamma'_1}(s(x)) \neq h^{\gamma'_2}(s(y))$.

Combining the two types of inequality constraints, we can conclude that $\delta_1 = \gamma_1$ and therefore $\ell = \ell'$, i.e. $(s, h) \models \mathfrak{m}_z(x, y) = \mathfrak{m}_u(x, y)$. Soundness of \mathcal{H}_C is certainly not immediate but this can be done similarly to the above developments for the axiom $(=\mathfrak{C})$.

► **Lemma 8.** \mathcal{H}_C is sound.

As done in Section 3, in order to establish that \mathcal{H}_C is complete, we first show its completeness with respect to *core types*, where $\text{CoreTypes}(\mathbf{X}, \alpha)$ is here defined as the set of formulae $\{\varphi \in \text{Conj}(\text{Core}(\mathbf{X}, \alpha)) \mid \forall \psi \in \text{Core}(\mathbf{X}, \alpha), \{\psi \mid \neg\psi\} \subseteq_{\text{Lt}} \varphi, \text{ and } (\psi \wedge \neg\psi) \not\subseteq_{\text{Lt}} \varphi\}$.

► **Lemma 9.** Let $\varphi \in \text{CoreTypes}(\mathbf{X}, \alpha)$. We have $\neg\varphi$ is valid iff $\vdash_{\mathcal{H}_C} \neg\varphi$. If $\vdash_{\mathcal{H}_C} \neg\varphi$ is provable then it has a proof where all derivation steps only have formulae from $\text{Bool}(\text{Core}(\mathbf{X}, \alpha))$.

Then, the proof of completeness of \mathcal{H}_C follows with the same arguments used for Theorem 2.

► **Theorem 10.** A Boolean combination of core formulae φ is valid iff $\vdash_{\mathcal{H}_C} \varphi$.

4.4 Constructive elimination of $\exists: \rightsquigarrow$

We write $\mathcal{H}_C(\exists: \rightsquigarrow)$ to denote the system \mathcal{H}_C augmented by the axioms and the inference rule from System 5. In System 5, given an arbitrary object \mathfrak{O} (this can be a term, a set of terms, a formula etc.), we write $\text{var}(\mathfrak{O})$ to denote the set of program variables occurring in \mathfrak{O} . For instance, $\text{var}(\mathfrak{m}_z(x, y)) = \{x, y, z\}$. Axioms from (\exists_{40}) to (\exists_{42}) and the introduction rule are classical tautologies of first-order quantification, whereas the other axioms characterise the peculiar semantics of $\exists: \rightsquigarrow$. By way of example, let us explain why the axiom (\exists_{45}) , equal to $\text{sees}_{\emptyset}(x, y) \wedge \text{sees}_{\{y\}}(x, \mathfrak{t}_1) \Rightarrow \exists z: \langle x \rightsquigarrow y \rangle z = \mathfrak{t}_1$ ($z \notin \text{var}(\{x, y, \mathfrak{t}_1\})$) is sound. Suppose $(s, h) \models \text{sees}_{\emptyset}(x, y) \wedge \text{sees}_{\{y\}}(x, \mathfrak{t}_1)$. By the semantics of core formulae, we have $\emptyset \neq h[s(x), \llbracket \mathfrak{t}_1 \rrbracket_{s,h} \subseteq h[s(x), s(y)]$ and therefore $\llbracket \mathfrak{t}_1 \rrbracket_{s,h}$ is defined. Given $z \notin \text{var}(\{x, y, \mathfrak{t}_1\})$, we have $(s[z \leftarrow \llbracket \mathfrak{t}_1 \rrbracket_{s,h}], h) \models z = \mathfrak{t}_1$. This holds because $z \notin \text{var}(\mathfrak{t}_1)$ as we want to guarantee $\llbracket \mathfrak{t}_1 \rrbracket_{s,h} = \llbracket \mathfrak{t}_1 \rrbracket_{s[z \leftarrow \llbracket \mathfrak{t}_1 \rrbracket_{s,h}], h}$. From $\emptyset \neq h[s(x), \llbracket \mathfrak{t}_1 \rrbracket_{s,h} \subseteq h[s(x), s(y)]$, we conclude that $h[s(x), s(y)] \neq \emptyset$ and $\llbracket \mathfrak{t}_1 \rrbracket_{s,h} \in h[s(x), s(y)] \cup \{s(y)\}$. Therefore, $(s, h) \models \exists z: \langle x \rightsquigarrow y \rangle z = \mathfrak{t}_1$. As done in Section 3 for $*$ and \ast , given a formula $\exists z: \langle x \rightsquigarrow y \rangle \varphi$, where φ is in $\text{CoreTypes}(\mathbf{X}, \alpha)$, we can show within $\mathcal{H}_C(\exists: \rightsquigarrow)$ that there is a conjunction χ from $\text{Conj}(\text{Core}(\mathbf{X}, 2\alpha))$ equivalent to it. By the axiom (\exists_{42}) , this applies when φ is a Boolean combination of core formulae.

► **Lemma 11.** Let $\varphi \in \text{Bool}(\text{Core}(\mathbf{X} \cup \{z\}, \alpha))$ with $z \notin \mathbf{X} \supseteq \{x, y\}$. There is a Boolean combination of core formulae $\chi \in \text{Bool}(\text{Core}(\mathbf{X}, 2\alpha))$ such that $\vdash_{\mathcal{H}_C(\exists: \rightsquigarrow)} \exists z: \langle x \rightsquigarrow y \rangle \varphi \Leftrightarrow \chi$.

4.5 Eliminating $*$ with a big-step axiom

The proof system $\mathcal{H}_C(*, \exists: \rightsquigarrow)$ for $\text{SL}(*, \exists: \rightsquigarrow)$ is defined as $\mathcal{H}_C(\exists: \rightsquigarrow)$ augmented by the axioms and the rule from System 6. Its main ingredient is given by the axiom (\ast_{48}) which, following the description in Section 3.4, is clearly a big-step axiom. Indeed, as much as we would like to give a set of small-step axioms as we did for $\text{SL}(*, \ast)$, we argue that producing

(System 4) \mathcal{H}_C : Axioms for Boolean combinations of core formulae

$$\begin{array}{ll}
(=^C_1) x = m_x(x, x) & (=^C_6) \text{def}(m_z(x, y)) \wedge \text{def}(m_u(x, y)) \Rightarrow m_z(x, y) = m_u(x, y) \\
(=^C_2) t_1 = t_2 \Rightarrow t_2 = t_1 & (=^C_7) m_z(x, y) = m_w(u, v) \Rightarrow \text{def}(m_w(x, y)) \\
(=^C_3) \varphi \wedge t_1 = t_2 \Rightarrow \varphi[t_1 \leftarrow t_2] & (=^C_8) \text{def}(m_z(x, y)) \wedge \text{def}(m_v(z, z)) \Rightarrow \text{def}(m_v(x, y)) \\
(=^C_4) \text{def}(m_x(x, y)) \Rightarrow x = m_x(x, y) & (=^C_9) \text{def}(m_z(x, y)) \wedge \text{def}(m_v(x, u)) \Rightarrow \text{def}(m_v(z, z)) \vee \text{def}(m_z(v, v)) \\
(=^C_5) \text{def}(m_z(x, y)) \Rightarrow \text{def}(m_z(y, x)) & (=^C_{10}) \text{def}(m_z(x, y)) \wedge \text{def}(m_z(u, v)) \Rightarrow \text{def}(m_z(x, u)) \\
(=^C_{11}) \text{sym}(m_z(x, y)) \wedge \text{def}(m_z(x, u)) \wedge m_z(x, u) \neq m_z(y, u) \Rightarrow (m_z(x, y) = m_z(x, u) \vee m_z(x, y) = m_z(y, u)) \\
(=^C_{12}) m_z(x, y) = m_z(u, v) \Rightarrow \text{sym}(m_z(x, u)) \wedge (m_z(x, y) = m_z(x, u) \vee m_z(x, y) = m_z(x, v)) \\
(=^C_{13}) \text{sym}(m_z(x, y)) \wedge \text{asym}(m_v(x, u)) \Rightarrow m_v(y, u) = m_v(x, u) \wedge m_v(u, y) = m_v(u, x) \\
(=^C_{14}) \text{asym}(m_z(x, y)) \wedge \text{asym}(m_v(x, u)) \Rightarrow m_z(x, y) = m_v(x, u)
\end{array}$$

$$\begin{array}{ll}
(s^C_{15}) t=t' \wedge \text{sees}_{\{t\} \cup \mathfrak{I}}(t_1, t_2) \Rightarrow \text{sees}_{\{t, t'\} \cup \mathfrak{I}}(t_1, t_2) & (s^C_{16}) \text{sees}_{\mathfrak{I} \cup \{t\}}(t_1, t_2) \geq \beta \Rightarrow \text{sees}_{\mathfrak{I}}(t_1, t_2) \geq \beta \\
(s^C_{17}) \text{sees}_{\mathfrak{I}}(t_1, t_2) \geq \beta \Rightarrow \text{sees}_{\mathfrak{I} \cup \{t_1, t_2\}}(t_1, t_2) \geq \beta & (s^C_{18}) \text{sees}_{\mathfrak{I}}(t_1, t_2) \geq \beta + 2 \Rightarrow \text{sees}_{\mathfrak{I}}(t_1, t_2) \geq \beta + 1 \\
(s^C_{19}) \text{sees}_{\{t_3\}}(t_1, t_2) \wedge \text{sees}_{\{t_2\}}(t_1, t_3) \Rightarrow t_2 = t_3 & (s^C_{20}) \text{sees}_{\mathfrak{I}}(t_1, t_2) \Rightarrow \text{def}(t_1) \wedge \text{def}(t_2) \\
(s^C_{21}) \text{sees}_{\emptyset}(t_1, t_1) \wedge \neg \text{sees}_{\{t_2\}}(t_1, t_1) \Leftrightarrow \text{samecycle}(t_1, t_2) & (s^C_{22}) \text{before}(t_1, t_2) \Rightarrow \text{sees}_{\emptyset}(t_1, t_2) \\
(s^C_{23}) \text{sees}_{\mathfrak{I}}(t_1, t_2) \geq \beta \wedge \text{sees}_{\mathfrak{I}'}(t_1, t_2) \geq \beta' \Rightarrow \text{sees}_{\mathfrak{I} \cup \mathfrak{I}'}(t_1, t_2) \geq \max(\beta, \beta') \\
(s^C_{24}) \text{sees}_{\mathfrak{I}}(t_1, t_2) \geq \beta_1 \wedge \text{sees}_{\mathfrak{I}}(t_2, t_3) \geq \beta_2 \wedge t_2 \notin \mathfrak{I} \wedge t_3 \in \mathfrak{I} \Rightarrow \text{sees}_{\mathfrak{I}}(t_1, t_3) \geq \beta_1 + \beta_2 \wedge \neg \text{sees}_{\{t_2\}}(t_1, t_3) \\
(s^C_{25}) \text{sees}_{\mathfrak{I}}(t_1, t_3) \geq \beta \wedge \neg \text{sees}_{\{t_2\}}(t_1, t_3) \Rightarrow \bigvee_{\beta_1 + \beta_2 = \max(2, \beta) - 2} (\text{sees}_{\mathfrak{I}}(t_1, t_2) \geq \beta_1 + 1 \wedge \text{sees}_{\mathfrak{I}}(t_2, t_3) \geq \beta_2 + 1) \\
(s^C_{26}) \text{sees}_{\mathfrak{I}}(m_z(x, y), m_w(u, v)) \Rightarrow \text{def}(m_w(x, y)) \\
(s^C_{27}) \text{sees}_{\mathfrak{I}}(m_z(x, y), m_w(u, v)) \wedge \text{asym}(m_w(x, u)) \Rightarrow m_w(u, v) = m_w(u, x) \\
(s^C_{28}) \text{sees}_{\mathfrak{I}}(m_z(x, y), m_w(u, v)) \wedge \text{sym}(m_w(x, u)) \wedge m_z(x, y) \neq m_w(u, v) \Rightarrow \text{before}(m_z(x, y), m_w(u, v)) \\
(s^C_{29}) \text{before}(t_1, t_2) \wedge \neg \text{sees}_{\{t_3\}}(t_1, t_2) \Rightarrow \neg \text{sees}_{\emptyset}(t_2, t_3) \wedge \text{before}(t_1, t_3) \\
(s^C_{30}) \text{samecycle}(t_1, t_2) \wedge \text{samecycle}(t_2, t_3) \wedge t_1 \neq t_3 \Rightarrow (\text{sees}_{\{t_2\}}(t_1, t_3) \Leftrightarrow \neg \text{sees}_{\{t_2\}}(t_3, t_1)) \\
(r^C_{31}) \text{rem}_{\mathfrak{I}} \geq 0 & (r^C_{34}) t_1 = t_2 \wedge \text{rem}_{\{(t_1, t_3)\} \cup \mathfrak{I}} \geq \beta \Rightarrow \text{rem}_{\{(t_1, t_3), (t_2, t_3)\} \cup \mathfrak{I}} \geq \beta \\
(r^C_{32}) \text{rem}_{\mathfrak{I}} \geq \beta + 1 \Rightarrow \text{rem}_{\mathfrak{I}} \geq \beta & (r^C_{35}) t_1 = t_2 \wedge \text{rem}_{\{(t_3, t_1)\} \cup \mathfrak{I}} \geq \beta \Rightarrow \text{rem}_{\{(t_3, t_1), (t_3, t_2)\} \cup \mathfrak{I}} \geq \beta \\
(r^C_{33}) \text{rem}_{\{(t_1, t_2)\} \cup \mathfrak{I}} \geq \beta \Rightarrow \text{rem}_{\mathfrak{I}} \geq \beta & (r^C_{36}) \neg \text{sees}_{\emptyset}(t_1, t_2) \geq \beta_2 + 1 \wedge \text{rem}_{\mathfrak{I}} \geq \beta_1 \Rightarrow \text{rem}_{\mathfrak{I} \cup \{(t_1, t_2)\}} \geq \beta_1 + \beta_2 \\
(r^C_{37}) \text{sees}_{\mathfrak{I}}(t_1, t_2) \wedge \neg \text{sees}_{\{t_3\}}(t_1, t_2) \wedge \text{rem}_{\{(t_1, t_3), (t_3, t_2)\} \cup \mathfrak{I}} \geq \beta \Rightarrow \text{rem}_{\{(t_1, t_2)\} \cup \mathfrak{I}} \geq \beta \\
(r^C_{38}) \text{sees}_{\mathfrak{I}}(t_1, t_2) \wedge \neg \text{sees}_{\{t_3\}}(t_1, t_2) \wedge \text{rem}_{\{(t_1, t_2)\} \cup \mathfrak{I}} \geq \beta \Rightarrow \text{rem}_{\{(t_1, t_2), (t_1, t_3), (t_3, t_2)\} \cup \mathfrak{I}} \geq \beta \\
(r^C_{39}) (\text{sees}_{\emptyset}(t_1, t_2) \geq \beta_2 \wedge \bigwedge_{(t_3, t_4) \in \mathfrak{I}} (\text{sees}_{\emptyset}(t_3, t_4) \Rightarrow \text{sees}_{\{t_3, t_4\}}(t_1, t_2) \wedge \text{sees}_{\{t_1, t_2\}}(t_3, t_4) \wedge t_3 \neq t_1) \\
\wedge \text{rem}_{\mathfrak{I} \cup \{(t_1, t_2)\}} \geq \beta_1) \Rightarrow \text{rem}_{\mathfrak{I}} \geq \beta_1 + \beta_2
\end{array}$$

(System 5) Axioms and inference rule for the guarded quantification $\exists: \rightsquigarrow$

$$\begin{array}{ll}
(\exists_{40}) \exists z: \langle x \rightsquigarrow y \rangle \varphi \Rightarrow \exists u: \langle x \rightsquigarrow y \rangle (\varphi[z \leftarrow u]) & \llbracket u \notin \text{var}(\varphi) \rrbracket \\
(\exists_{41}) \exists z: \langle x \rightsquigarrow y \rangle (\varphi \wedge \psi) \Leftrightarrow (\exists z: \langle x \rightsquigarrow y \rangle \varphi) \wedge \psi & \llbracket z \notin \text{var}(\psi) \rrbracket \quad \exists\text{-Intro: } \frac{\varphi \Rightarrow \psi}{\exists z: \langle x \rightsquigarrow y \rangle \varphi \Rightarrow \exists z: \langle x \rightsquigarrow y \rangle \psi} \\
(\exists_{42}) \exists z: \langle x \rightsquigarrow y \rangle (\varphi_1 \vee \varphi_2) \Leftrightarrow (\exists z: \langle x \rightsquigarrow y \rangle \varphi_1) \vee (\exists z: \langle x \rightsquigarrow y \rangle \varphi_2) \\
(\exists_{43}) \text{sees}_{\emptyset}(x, y) \Rightarrow \exists z: \langle x \rightsquigarrow y \rangle z = x & \llbracket z \notin \{x, y\} \rrbracket \quad (\exists_{44}) \neg \exists z: \langle x \rightsquigarrow y \rangle \perp \\
(\exists_{45}) \text{sees}_{\emptyset}(x, y) \wedge \text{sees}_{\{y\}}(x, t_1) \Rightarrow \exists z: \langle x \rightsquigarrow y \rangle z = t_1 & \llbracket z \notin \text{var}(\{x, y, t_1\}) \rrbracket \\
(\exists_{46}) (x = t_1 \vee \text{sees}_{\mathfrak{I}'}(x, t_1)) \wedge \text{sees}_{\mathfrak{I}}(t_1, t_2) \geq \beta_1 + \beta_2 \wedge (t_2 = y \vee \text{sees}_{\mathfrak{I}''}(t_2, y)) \wedge (y = t_1 \Rightarrow x = y) \\
\Rightarrow \exists z: \langle x \rightsquigarrow y \rangle (\text{sees}_{\mathfrak{I}}(t_1, z) \sim_1 \beta_1 \wedge \text{sees}_{\mathfrak{I}}(z, t_2) \sim_2 \beta_2 \wedge z \notin \{t_1, t_2\}) \\
\llbracket \{x, y, t_1, t_2\} \subseteq \mathfrak{I}, \mathfrak{I}', \mathfrak{I}'', z \notin \text{var}(\mathfrak{I}), \beta_1, \beta_2 \in \mathbb{N}^+, \geq \in \{\sim_1, \sim_2\} \subseteq \{\geq, =\} \rrbracket \\
(\exists_{47}) \neg \exists z: \langle x \rightsquigarrow y \rangle ((x \neq z \wedge y \neq z \wedge \text{sees}_{\{x, z, y\}}(x, y)) \vee \neg \text{sees}_{\emptyset}(x, y)) & \llbracket z \notin \{x, y\} \rrbracket
\end{array}$$

(System 6) Axioms and inference rule for the separating conjunction

$$\begin{array}{ll}
(*48) \Gamma_{\text{Sms}}(\mathfrak{S}_1) * \Gamma_{\text{Sms}}(\mathfrak{S}_2) \Leftrightarrow \bigvee_{\mathfrak{S} \text{ s.t. } +^s(\mathfrak{S}_1, \mathfrak{S}_2, \mathfrak{S})} \Gamma_{\text{Sms}}(\mathfrak{S}) & \llbracket \mathfrak{S}_1, \mathfrak{S}_2 \text{ resp. over } (X, \alpha_1) \text{ and } (X, \alpha_2) \rrbracket \quad *\text{-Intro: } \frac{\varphi \Rightarrow \chi}{\varphi * \psi \Rightarrow \chi * \psi} \\
(*49) (\varphi \vee \psi) * \chi \Rightarrow (\varphi * \chi) \vee (\psi * \chi) & (*50) (\varphi * \psi) \Leftrightarrow (\psi * \varphi) \quad (*51) (\perp * \psi) \Leftrightarrow \perp
\end{array}$$

such an axiomatisation for $\text{SL}(*, \exists: \rightsquigarrow)$ is unfeasible. In the proof system for $\text{SL}(*, *)$, we found out that given two core types φ and ψ , $\varphi * \psi$ is equivalent to a conjunction of core formulae literals (see the proof sketch of Lemma 3). Similar results hold for the separating implication \multimap (Lemma 5) and the $\exists: \rightsquigarrow$ quantifier. This property of being equivalent to a simple conjunction of core formulae literals facilitates the design of small-step axioms. This is not the case for $*$ within $\text{SL}(*, \exists: \rightsquigarrow)$: given two core types φ and ψ , the formula $\varphi * \psi$ is equivalent to a non-trivial disjunction of possibly exponentially many conjunctions. Because of this, small-step axioms are hard to obtain and some technical developments are needed in order to produce an adequate axiom system. These developments are centered around the notions of symbolic memory states and characteristic formulae. A *symbolic memory state* is an abstraction on the memory state (s, h) that is guided by the definition of core formulae, essentially highlighting the properties of (s, h) that are expressible through these formulae, while removing the ones that are not expressible. Given $\mathbf{X} \subseteq_{\text{fin}} \text{VAR}$ and $\alpha \in \mathbb{N}^+$, a *symbolic memory states* \mathfrak{S} over (\mathbf{X}, α) is defined as a finite structure $(\mathfrak{D}, \mathfrak{f}, \mathfrak{r})$ such that

- \mathfrak{D} is a partition of a subset of $\mathbb{T}(\mathbf{X})$, encoding (dis)equalities. We introduce the partial function $[\cdot]_{\mathfrak{D}} : \mathbb{T}(\mathbf{X}) \rightarrow \mathfrak{D}$ such that given $\mathfrak{t} \in \mathbb{T}(\mathbf{X})$ returns $\mathfrak{T} \in \mathfrak{D}$ and $\mathfrak{t} \in \mathfrak{T}$, if it exists;
- $\mathfrak{f} : \mathfrak{D} \rightarrow \mathfrak{D} \times [1, \alpha]$ is a partial function encoding paths between terms and their length;
- $\mathfrak{r} \in [0, \alpha]$, encoding the number of memory cells (up to α) not in paths between terms.

We denote with $\text{SMS}_{\alpha}^{\mathbf{X}}$ the set of these structures. The *abstraction* $\text{Symb}_{\alpha}^{\mathbf{X}}(s, h)$ of a memory state (s, h) is defined as the symbolic memory state $(\mathfrak{D}, \mathfrak{f}, \mathfrak{r})$ over (\mathbf{X}, α) such that

- $\mathfrak{D} \stackrel{\text{def}}{=} \{ \{ \mathfrak{t}_1 \in \mathbb{T}(\mathbf{X}) \mid (s, h) \models \mathfrak{t}_1 = \mathfrak{t}_2 \} \mid \mathfrak{t}_2 \in \mathbb{T}(\mathbf{X}) \}$;
- $\mathfrak{f}(\mathfrak{T}) = (\mathfrak{T}', \beta) \stackrel{\text{def}}{\iff}$ there are $\mathfrak{t}_1 \in \mathfrak{T}$ and $\mathfrak{t}_2 \in \mathfrak{T}'$ such that $(s, h) \models \text{sees}_{\mathbb{T}(\mathbf{X})}(\mathfrak{t}_1, \mathfrak{t}_2) \geq \beta$ and if $\beta < \alpha$ then $(s, h) \models \neg \text{sees}_{\mathbb{T}(\mathbf{X})}(\mathfrak{t}_1, \mathfrak{t}_2) \geq \beta + 1$;
- $\mathfrak{r} = \beta \stackrel{\text{def}}{\iff} (s, h) \models \text{rem}_{\mathbb{T}(\mathbf{X}) \times \mathbb{T}(\mathbf{X})} \geq \beta$ and if $\beta < \alpha$ then $(s, h) \models \neg \text{rem}_{\mathbb{T}(\mathbf{X}) \times \mathbb{T}(\mathbf{X})} \geq \beta + 1$.

Thus, a symbolic memory state $(\mathfrak{D}, \mathfrak{f}, \mathfrak{r})$ over (\mathbf{X}, α) simply stores the truth values for equalities, sees and rem predicates with respect to a memory state. Its semantics is best given through the *characteristic formula* $\Gamma_{\text{sms}}(\mathfrak{D}, \mathfrak{f}, \mathfrak{r})$ defined below (sets understood as conjunctions):

$$\begin{aligned} & \{ \text{rem}_{\mathbb{T}(\mathbf{X}) \times \mathbb{T}(\mathbf{X})} \sim \mathfrak{r} \mid \text{if } \mathfrak{r} \neq \alpha \text{ then } (\sim \text{ is } =) \text{ else } (\sim \text{ is } \geq) \} \wedge \{ \mathfrak{t}_1 \neq \mathfrak{t}_2 \mid [\mathfrak{t}_1]_{\mathfrak{D}} \text{ or } [\mathfrak{t}_2]_{\mathfrak{D}} \text{ undefined, or } [\mathfrak{t}_1]_{\mathfrak{D}} \neq [\mathfrak{t}_2]_{\mathfrak{D}} \} \\ & \wedge \{ \mathfrak{t}_1 = \mathfrak{t}_2 \mid [\mathfrak{t}_1]_{\mathfrak{D}} = [\mathfrak{t}_2]_{\mathfrak{D}} \text{ defined} \} \wedge \{ \neg \text{sees}_{\mathbb{T}(\mathbf{X})}(\mathfrak{t}_1, \mathfrak{t}_2) \mid [\mathfrak{t}_1]_{\mathfrak{D}} \text{ undefined or } \forall \beta \in [1, \alpha] : \mathfrak{f}([\mathfrak{t}_1]_{\mathfrak{D}}) \neq ([\mathfrak{t}_2]_{\mathfrak{D}}, \beta) \} \\ & \wedge \{ \text{sees}_{\mathbb{T}(\mathbf{X})}(\mathfrak{t}_1, \mathfrak{t}_2) = \beta \mid \mathfrak{f}([\mathfrak{t}_1]_{\mathfrak{D}}) = ([\mathfrak{t}_2]_{\mathfrak{D}}, \beta) \text{ and } \beta < \alpha \} \wedge \{ \text{sees}_{\mathbb{T}(\mathbf{X})}(\mathfrak{t}_1, \mathfrak{t}_2) \geq \beta \mid \mathfrak{f}([\mathfrak{t}_1]_{\mathfrak{D}}) = ([\mathfrak{t}_2]_{\mathfrak{D}}, \beta) \text{ and } \beta = \alpha \} \end{aligned}$$

From the definitions of $\Gamma_{\text{sms}}(\mathfrak{S})$ and $\text{Symb}_{\alpha}^{\mathbf{X}}(s, h)$, we can easily prove the following result.

► **Lemma 12.** *For every (s, h) and every $\mathfrak{S} \in \text{SMS}_{\alpha}^{\mathbf{X}}$, $(s, h) \models \Gamma_{\text{sms}}(\mathfrak{S})$ iff $\mathfrak{S} = \text{Symb}_{\alpha}^{\mathbf{X}}(s, h)$.*

Thanks to this lemma, it is easy to see that every satisfiable characteristic formula $\Gamma_{\text{sms}}(\mathfrak{S})$ of a symbolic memory state \mathfrak{S} over (\mathbf{X}, α) is equivalent to exactly one core type in $\text{CoreTypes}(\mathbf{X}, \alpha)$. Indeed, by definition of core types, the conjunction $\varphi \wedge \psi$ of two core types φ and ψ that are not syntactically equivalent up to associativity and commutativity of \wedge is unsatisfiable. Hence, by Lemma 12, if a core type $\varphi \in \text{CoreTypes}(\mathbf{X}, \alpha)$ is satisfied by a memory state (s, h) , it must be equivalent to $\Gamma_{\text{sms}}(\text{Symb}_{\alpha}^{\mathbf{X}}(s, h))$. By Theorem 10 this equivalence is provable in $\mathcal{H}_{\mathcal{C}}$.

The fundamental reason for taking symbolic memory states over memory states is that, given \mathbf{X} and α , there are finitely many symbolic memory states in $\text{SMS}_{\alpha}^{\mathbf{X}}$. This leads to the definition of the axiom $(*_{48})$, which given two characteristic formulae φ and ψ computes a finite disjunction of characteristic formulae that is equivalent to $\varphi * \psi$. This disjunction is defined over a new composition operator $+^{\text{S}}$ on symbolic memory states that mimicks the disjoint union $+$ on memory states. More precisely, the following property shall be satisfied.

For all (s, h) and all $\mathfrak{S}_1, \mathfrak{S}_2$ resp. over (\mathbf{X}, α_1) and (\mathbf{X}, α_2) , $+^{\text{S}}(\mathfrak{S}_1, \mathfrak{S}_2, \text{Symb}_{\alpha_1 + \alpha_2}^{\mathbf{X}}(s, h))$ iff there are h_1 and h_2 such that $h_1 + h_2 = h$, $\mathfrak{S}_1 = \text{Symb}_{\alpha_1}^{\mathbf{X}}(s, h_1)$ and $\mathfrak{S}_2 = \text{Symb}_{\alpha_2}^{\mathbf{X}}(s, h_2)$,

where $+^S \subseteq \sum_{\mathbf{x}, \alpha_1, \alpha_2} \text{SMS}_{\alpha_1}^{\mathbf{x}} \times \text{SMS}_{\alpha_2}^{\mathbf{x}} \times \text{SMS}_{\alpha_1 + \alpha_2}^{\mathbf{x}}$, and $\mathfrak{S}_1, \mathfrak{S}_2$ have satisfiable characteristic formulae. Defining $+^S$ is clearly challenging. Unlike the disjoint union of memory states, $+^S$ is not functional on its first two components. For instance, let $\mathfrak{S} = (\{\mathbf{x}, \mathbf{m}_{\mathbf{x}}(\mathbf{x}, \mathbf{x})\}, \emptyset, 1)$ and let us determine for which \mathfrak{S}' , we have $+^S(\mathfrak{S}, \mathfrak{S}, \mathfrak{S}')$:

1. As \mathfrak{S} is the abstraction of the memory states $(s, \{\ell_1 \mapsto \ell_2\})$ and $(s, \{\ell_2 \mapsto \ell_1\})$ where $s(\mathbf{x}) = \ell_1 \neq \ell_2$, the abstraction of $(s, \{\ell_1 \mapsto \ell_2, \ell_2 \mapsto \ell_1\})$ must be a solution for \mathfrak{S}' . More precisely, this abstraction is $(\mathbb{T}, \{\mathbb{T} \mapsto (\mathbb{T}, 2)\}, 0)$ where $\mathbb{T} = \{\mathbf{x}, \mathbf{m}_{\mathbf{x}}(\mathbf{x}, \mathbf{x})\}$.
2. \mathfrak{S} is however also the abstraction of $(s, \{\ell_1 \mapsto \ell_2\})$ and $(s, \{\ell_3 \mapsto \ell_4\})$ such that $s(\mathbf{x}) \notin \{\ell_1, \ell_3\}$. Then, the abstraction $(\{\mathbf{x}, \mathbf{m}_{\mathbf{x}}(\mathbf{x}, \mathbf{x})\}, \emptyset, 2)$ must also be a solution for \mathfrak{S}' .

The main challenge for defining $+^S$ is the composition of the two “garbage”: memory cells that are abstracted with τ_1 and τ_2 in $\text{Symb}_{\alpha_1}^{\mathbf{x}_1}(s, h_1)$ and $\text{Symb}_{\alpha_2}^{\mathbf{x}_2}(s, h_2)$ may generate new paths between program variables in $h_1 + h_2$. This possibility was depicted in the first case above. The definition of $+^S$ can be found in [17] and is too long to be presented herein. Roughly speaking, for $((\mathfrak{D}, f_1, \tau_1), (\mathfrak{D}, f_2, \tau_2), (\mathfrak{D}, f, \tau))$ being in $+^S$, one needs to witness two graph homomorphisms from the graphs (\mathfrak{D}_1, f_1) and (\mathfrak{D}_2, f_2) to (\mathfrak{D}, f) , together with the existence of a partition that guarantees that paths that do not belong to the homomorphisms can be generated using the memory cells from the garbage (abstracted by τ_1 and τ_2).

Together with the other axioms in System 6, which essentially allows to rewrite every formula into a disjunction of $\varphi * \psi$ where φ and ψ are characteristic formulae, the axiom (*48) allows us to eliminate $*$, as done in Lemma 3 for $\text{SL}(*, -)$.

► **Lemma 13.** *Let $\varphi \in \text{Bool}(\text{Core}(\mathbf{X}, \alpha_1))$ and $\psi \in \text{Bool}(\text{Core}(\mathbf{X}, \alpha_2))$. There is a Boolean combination of core formulae $\chi \in \text{Bool}(\text{Core}(\mathbf{X}, \alpha_1 + \alpha_2))$ such that $\vdash_{\mathcal{H}_{\mathcal{C}}(*, \exists: \rightsquigarrow)} \varphi * \psi \Leftrightarrow \chi$.*

The adequacy of $\mathcal{H}_{\mathcal{C}}(*, \exists: \rightsquigarrow)$ then stems from Theorem 10 and Lemmata 11 and 13.

► **Theorem 14.** *$\mathcal{H}_{\mathcal{C}}(*, \exists: \rightsquigarrow)$ is sound and complete for $\text{SL}(*, \exists: \rightsquigarrow)$.*

4.6 A PSpace upper bound for checking $\text{SL}(*, \exists: \rightsquigarrow)$ satisfiability

In this short section, we explain why the satisfiability problem for $\text{SL}(*, \exists: \rightsquigarrow)$ is in PSPACE. The *memory size* of a formula φ , written $|\varphi|_{\text{m}}$, is defined inductively as: $|\mathbf{x} = \mathbf{y}|_{\text{m}} \stackrel{\text{def}}{=} |\text{emp}|_{\text{m}} = 1$, $|\mathbf{x} \mapsto \mathbf{y}|_{\text{m}} \stackrel{\text{def}}{=} 2$, $|\exists \mathbf{z}: \langle \mathbf{x} \rightsquigarrow \mathbf{y} \rangle \varphi|_{\text{m}} \stackrel{\text{def}}{=} 2 \times |\varphi|_{\text{m}}$, $|\neg \psi|_{\text{m}} \stackrel{\text{def}}{=} |\psi|_{\text{m}}$, $|\psi_1 * \psi_2|_{\text{m}} \stackrel{\text{def}}{=} |\psi_1|_{\text{m}} + |\psi_2|_{\text{m}}$ and $|\psi_1 \wedge \psi_2|_{\text{m}} \stackrel{\text{def}}{=} \max(|\psi_1|_{\text{m}}, |\psi_2|_{\text{m}})$. Given φ with tree height δ , $|\varphi|_{\text{m}} \leq 2^{\delta+1}$. Intuitively, $|\varphi|_{\text{m}}$ provides an upper bound on the path length between terms and on the size of the garbage on models for φ (above $|\varphi|_{\text{m}}$, φ cannot see the difference). As a consequence of the proofs for the elimination of the connectives $\exists: \rightsquigarrow$ and $*$ in the calculus, for each φ in $\text{SL}(*, \exists: \rightsquigarrow)$, there is a Boolean combination of core formulae from $\text{Core}(\text{var}(\varphi), |\varphi|_{\text{m}})$ logically equivalent to φ .

$\text{SL}(*, \exists: \rightsquigarrow)$ may require small memory states whose heap has an exponential amount of memory cells, as shown in Section 4.1 with the formula $\text{R}^n(\mathbf{x}, \mathbf{y})$. So, to establish a PSPACE bound, we cannot rely on an algorithm that guesses a polynomial-size memory state and performs model-checking on it without further refinements. Nevertheless, polynomial-size symbolic memory states are able to abstract a garbage of exponential size or a path between terms of exponential length by encoding these quantities in binary, which leads to PSPACE.

► **Theorem 15.** *The satisfiability problem for $\text{SL}(*, \exists: \rightsquigarrow)$ is PSPACE-complete.*

PSPACE-hardness is from [10]. To establish PSPACE-easiness, there is a nondeterministic polynomial-space algorithm that guesses a satisfiable $\mathfrak{S} \in \text{SMS}_{|\varphi|_{\text{m}}}^{\text{var}(\varphi)}$ and that performs a symbolic model-checking on \mathfrak{S} against φ . This works fine as $*$ and $\exists: \rightsquigarrow$ have symbolic counterparts that can be decided in polynomial space.

5 Conclusion

We presented a method to axiomatise internally separation logics based on the axiomatisation of Boolean combinations of core formulae. We designed the first proof system for $\text{SL}(*, -*)$ that is completely internal and highlights the essential ingredients of the heaplet semantics. To further illustrate our method, we provided an internal Hilbert-style axiomatisation for the new separation logic $\text{SL}(*, \exists: \rightsquigarrow)$. It contains the “list quantifier” $\exists z: \langle x \rightsquigarrow y \rangle$ that, we believe, is of interest for its own sake as it allows to quantify over elements of a list. The completeness proof, following our general pattern, still reveals to be very complex as not only we had to invent the adequate family of core formulae but their axiomatisation was challenging. As far as we know, this is the first axiomatisation of a separation logic having ls and a guarded form of quantification. Moreover, through a small model property derived from its proof system, we proved that $\text{SL}(*, \exists: \rightsquigarrow)$ has a PSPACE-complete satisfiability problem. Obviously, our proof systems for separation logics are of theoretical interest, for instance to grasp the essential features of the logics. It is open whether it can help for designing decision procedures, e.g. to feed provers with axiom instances to speed-up the proof search.

References

- 1 C. Areces, P. Blackburn, and M. Marx. Hybrid logics: characterization, interpolation and complexity. *The Journal of Symbolic Logic*, 66(3):977–1010, 2001.
- 2 J. Berdine, C. Calcagno, and P.W. O’Hearn. A decidable fragment of separation logic. In *FST&TCS’04*, volume 3328 of *LNCS*, pages 97–109. Springer, 2004.
- 3 M. Bozga, R. Iosif, and S. Perarnau. Quantitative Separation Logic and Programs with Lists. *Journal of Automated Reasoning*, 45(2):131–156, 2010.
- 4 R. Brochenin, S. Demri, and E. Lozes. On the Almighty Wand. *Information and Computation*, 211:106–137, 2012.
- 5 J. Brotherston. Bunched Logics Displayed. *Studia Logica*, 100(6):1223–1254, 2012. doi: 10.1007/s11225-012-9449-0.
- 6 J. Brotherston and M. Kanovich. Undecidability of Propositional Separation Logic and Its Neighbours. *Journal of the Association for Computing Machinery*, 61(2), 2014.
- 7 J. Brotherston and M. Kanovich. On the Complexity of Pointer Arithmetic in Separation Logic. In *APLAS’18*, volume 11275 of *LNCS*, pages 329–349. Springer, 2018.
- 8 J. Brotherston and J. Villard. Parametric completeness for separation theories. In *POPL’14*, pages 453–464. ACM, 2014.
- 9 C. Calcagno, Ph. Gardner, and M. Hague. From separation logic to first-order logic. In *FoSSaCS’05*, volume 3441 of *LNCS*, pages 395–409. Springer, 2005.
- 10 C. Calcagno, P.W. O’Hearn, and H. Yang. Computability and Complexity Results for a Spatial Assertion Language for Data Structures. In *FST&TCS’01*, volume 2245 of *LNCS*, pages 108–119. Springer, 2001.
- 11 B. Cook, C. Haase, J. Ouaknine, M. Parkinson, and J. Worrell. Tractable Reasoning in a Fragment of Separation Logic. In *CONCUR’11*, volume 6901 of *LNCS*, pages 235–249. Springer, 2011.
- 12 S. Demri and M. Deters. Separation Logics and Modalities: A Survey. *Journal of Applied Non-Classical Logics*, 25(1):50–99, 2015.
- 13 S. Demri, R. Fervari, and A. Mansutti. Axiomatising Logics with Separating Conjunction and Modalities. In *JELIA’19*, volume 11468 of *LNAI*, pages 692–708. Springer, 2019.
- 14 S. Demri, D. Galmiche, D. Larchey-Wendling, and D. Mery. Separation Logic with One Quantified Variable. *Theory of Computing Systems*, 61:371–461, 2017.

- 15 S. Demri, É. Lozes, and A. Mansutti. The Effects of Adding Reachability Predicates in Propositional Separation Logic. In *FoSSaCS*, volume 10803 of *LNCS*, pages 476–493. Springer, 2018.
- 16 S. Demri, E. Lozes, and A. Mansutti. The Effects of Adding Reachability Predicates in Propositional Separation Logic. arXiv:1810.05410, October 2018. 44 pages. Long version of [15].
- 17 S. Demri, E. Lozes, and A. Mansutti. Internal Calculi for Separation logics. arXiv:1910.05016, October 2019.
- 18 S. Docherty and D. Pym. Modular Tableaux Calculi for Separation Theories. In *FoSSaCS'18*, volume 10803 of *LNCS*, pages 441–458. Springer, 2018.
- 19 A. Doumane. Constructive completeness for the linear-time μ -calculus. In *LICS'17*, pages 1–12. IEEE Computer Society, 2017.
- 20 M. Echenim, R. Iosif, and N. Peltier. The Bernays-Schönfinkel-Ramsey Class of Separation Logic on Arbitrary Domains. In *FoSSaCS'19*, volume 11425 of *LNCS*, pages 242–259. Springer, 2019.
- 21 D. Galmiche and D. Larchey-Wending. Expressivity properties of Boolean BI through relational models. In *FST&TCS'06*, volume 4337 of *LNCS*, pages 358–369. Springer, 2006.
- 22 D. Galmiche and D. Méry. Tableaux and Resource Graphs for Separation Logic. *Journal of Logic and Computation*, 20(1):189–231, 2010.
- 23 V. Goranko and G. van Drimmelen. Complete axiomatization and decidability of Alternating-time temporal logic. *Theoretical Computer Science*, 353(1-3):93–117, 2006.
- 24 E. Grädel and I. Walukiewicz. Guarded Fixed Point Logic. In *LICS'99*, pages 45–54, 1999.
- 25 Z. Hou, R. Clouston, R. Goré, and A. Tiu. Modular labelled sequent calculi for abstract separation logics. *ACM Transactions on Computational Logic*, 19(2):13:1–13:35, 2018.
- 26 Z. Hou and A. Tiu. Completeness for a First-Order Abstract Separation Logic. In *APLAS'16*, volume 10017 of *LNCS*, pages 444–463. Springer, 2016.
- 27 S. Ishtiaq and P.W. O'Hearn. BI as an assertion language for mutable data structures. In *POPL'01*, pages 14–26. ACM, 2001.
- 28 R. Kaivola. Axiomatizing linear time mu-calculus. In *CONCUR'95*, volume 962 of *LNCS*, pages 423–437. Springer, 1995.
- 29 E. Lozes. *Expressivité des Logiques Spatiales*. PhD thesis, ENS Lyon, 2004.
- 30 E. Lozes. Separation Logic preserves the expressive power of classical logic. In *SPACE'04*, 2004.
- 31 M. Lück. Axiomatizations of team logics. *Annals of Pure and Applied Logic*, 169(9):928–969, 2018.
- 32 A. Mansutti. Extending Propositional Separation Logic for Robustness Properties. In *FST&TCS'18*, volume 122 of *LIPICs*, pages 42:1–42:23. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
- 33 P.W. O'Hearn. A Primer on Separation Logic. In *Software Safety and Security: Tools for Analysis and Verification*, volume 33 of *NATO Science for Peace and Security Series*, pages 286–318, 2012.
- 34 P.W. O'Hearn and D. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.
- 35 R. Piskac, Th. Wies, and D. Zufferey. Automating Separation Logic using SMT. In *CAV'13*, volume 8044 of *LNCS*, pages 773–789. Springer, 2013.
- 36 D. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic*. Kluwer Academic Publishers, 2002.
- 37 D. Pym, J. Spring, and P.W. O'Hearn. Why Separation Logic Works. *Philosophy & Technology*, pages 1–34, 2018.
- 38 J.C. Reynolds. Separation logic: a logic for shared mutable data structures. In *LICS'02*, pages 55–74. IEEE, 2002.


19:18 Internal Calculi for Separation Logics

- 39 I. Walukiewicz. Completeness of Kozen's axiomatisation of the propositional μ -calculus. *Information and Computation*, 157(1-2):142-182, 2000.
- 40 Y. Wang and Q. Cao. On axiomatizations of public announcement logic. *Synthese*, 190(Supplement-1):103-134, 2013.
- 41 H. Yang. *Local Reasoning for Stateful Programs*. PhD thesis, University of Illinois, Urbana-Champaign, 2001.

Monitoring Event Frequencies

Thomas Ferrère 

IST Austria, Klosterneuburg, Austria

Thomas A. Henzinger 

IST Austria, Klosterneuburg, Austria

Bernhard Kragl 

IST Austria, Klosterneuburg, Austria

Abstract

The monitoring of event frequencies can be used to recognize behavioral anomalies, to identify trends, and to deduce or discard hypotheses about the underlying system. For example, the performance of a web server may be monitored based on the ratio of the total count of requests from the least and most active clients. Exact frequency monitoring, however, can be prohibitively expensive; in the above example it would require as many counters as there are clients. In this paper, we propose the efficient probabilistic monitoring of common frequency properties, including the mode (i.e., the most common event) and the median of an event sequence. We define a logic to express composite frequency properties as a combination of atomic frequency properties. Our main contribution is an algorithm that, under suitable probabilistic assumptions, can be used to monitor these important frequency properties with four counters, independent of the number of different events. Our algorithm samples longer and longer subwords of an infinite event sequence. We prove the almost-sure convergence of our algorithm by generalizing ergodic theory from increasing-length prefixes to increasing-length subwords of an infinite sequence. A similar algorithm could be used to learn a connected Markov chain of a given structure from observing its outputs, to arbitrary precision, for a given confidence.

2012 ACM Subject Classification Software and its engineering → Software organization and properties; Theory of computation → Randomness, geometry and discrete structures

Keywords and phrases monitoring, frequency property, Markov chain

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.20

Related Version A full version of the paper is available at <http://arxiv.org/abs/1910.06097>.

Funding This research was supported in part by the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE/SHiNE) and Z211-N23 (Wittgenstein Award).

Acknowledgements We thank Jan Maas for showing us a simple proof of convergence of the mode in the i.i.d. case, and Zbigniew S. Szewczak for pointing out to us the use of Karamata's Tauberian theorem in connection with ergodic theory [23].

1 Introduction

The safety and security of computerized systems are ensured by a chain of methods that use logic and formal semantics to assert and check the correct operation of a system, real or simulated. Runtime monitoring [4] happens at the end of this chain and complements rigorous design and verification practices to catch malfunctions as they occur in a live system. In addition to critical functional aspects, softer performance metrics also need to be monitored to ensure a suitable quality of service. Monitoring system properties takes place in parallel with the execution of the system itself. A dedicated component, called monitor, observes the system behavior as input and generate a verdict about the system behavior as output. Due to reactivity considerations, the monitor is often required to perform its observations in real time, and not being the main computational artifact, should consume limited resources.



© Thomas Ferrère, Thomas A. Henzinger, and Bernhard Kragl;
licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 20; pp. 20:1–20:16

Leibniz International Proceedings in Informatics



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

In this paper we propose a new class of quantitative properties based on event frequencies, called *frequency properties*, and study their monitoring problem. In particular, we define a logic to express composite frequency properties as linear and Boolean combinations of atomic frequency properties. While all such frequency properties are theoretically monitorable using counter registers, there are, in general, no efficient monitoring algorithms in the case of large or infinite input alphabets. As a motivating example we use the *mode* of a sequence over a finite alphabet Σ . By definition, $a \in \Sigma$ is the mode of an ω -word w if there exists a length n such that each prefix of w longer than n contains more occurrences of a 's than occurrences of any other letter $b \in \Sigma$. This frequency property can be monitored using a separate counter for every event in Σ . However, the alphabet Σ is typically too large for this to be practical.¹ We show that there is no shortcut to monitor the mode exactly and in real time: in general $|\Sigma|$ counters are needed for this task.

However, we are not always interested in monitoring exactly and in real time the mode after every new event, and sometimes wish to estimate what the mode is expected to be in the future. Perhaps surprisingly, we can then do much better. Let us assume that the past, finite, observed behavior of an event sequence is representative of the future, infinite, unknown behavior. This is the case for stochastic systems, for instance if the observation sequence is generated by a Markov chain. We move from the *real-time monitoring* problem, asking to compute or approximate, in real time, the value of a frequency property for each observed prefix, to the *limit monitoring* problem, asking to estimate the future limit value of the frequency property, if it exists. In particular, for the mode of a connected Markov chain, the longer we observe a behavior, the higher our confidence in predicting its mode. While every real-time monitor can be used as limit monitor, there can be limit monitors that use dramatically fewer resources.

We present a simple, memory-efficient strategy to limit monitor frequency properties of random ω -words. In particular, our mode monitor uses four counters only. Two of the counters keep track of the number of occurrences of two letters at a time. The first letter is the current mode prediction, say a . The second letter is the mode replacement candidate, say b . We count the number of a 's and b 's over a given subword, until a certain number of events, say 10, has been processed. The most frequent letter out of a and b in this 10-letter subword, say a , wins the round and becomes the new mode prediction. The other letter loses the round and is replaced by a letter sampled at random, say c . In the next round the subword length will be increased, say to 11, and a will compete against c over the next subword. We reuse two counters for the two letters, and the other two counters to keep track of the current subword length and to stop counting when that length is reached. By repeating the process we get increasingly higher confidence that a is indeed the mode. Even if by random perturbation the mode a of the generating Markov chain was no longer the current prediction, it would eventually get sampled again and statistically reappear, and eventually remain, as the prediction.

The algorithm of our mode monitor easily transfers to an efficient monitor for the median. Indeed, we also show that our results generalize to any property expressible as Boolean combination of linear inequalities over frequencies of events. An application of our algorithmic ideas is to learn the transition probabilities of a connected Markov chain of known structure through the observation of subword frequencies.

¹ Consider the IPv4 protocol alphabet with its 4,294,967,296 letters (addresses) and the UTF-8 encoding alphabet with its 1,112,064 letters (code points).

The main result of this paper is that, assuming the monitored system is a connected Markov chain, our monitoring algorithm converges almost surely. The proof of this fact calls for a new ergodic theory based on subwords as opposed to prefixes. This theory uses as its main building block a variant of the law of large numbers over so-called triangular random arrays of the form $X_{1,1}, X_{2,1}, X_{2,2}, X_{3,1}, \dots$ and hinges on deep results from matrix theory. The correctness of the algorithm can also be understood, in a weaker form, by showing convergence in probability of its output. Assuming that the Markov chain starts in a stationary distribution, the probability of a given word u occurring as subword of an ω -word w at position i is independent of i . As a result, when the value of a function over prefixes converges probabilistically, then the same limit is reached probabilistically over arbitrary subwords.

In short, the main conceptual and technical contributions of this paper are the following:

1. We show that precise real-time monitoring is inherently resource-intensive (Section 4).
2. We propose the novel setting of limit monitoring (Section 3).
3. We provide a generic scheme for efficient limit monitoring (Section 5) and instantiate it to specialized monitoring algorithms for the mode (Section 5.2) and median (Section 5.3).
4. We define a logic for composite frequency properties which combines atomic frequency properties such that each formula of the logic can be limit monitored efficiently (Section 6).
5. We develop a new ergodic theory for connected Markov chains (Section 5.1) to prove our monitoring algorithms correct.

1.1 Related Work

In the area of formal verification, probabilistic model checking [15, 16] and quantitative verification [12] are concerned with the white-box static analysis of a probabilistic system. Statistical model checking [1] tries to learn the probabilistic structure of a system by sampling *many* executions, and thus also applies to black-box systems. These are in contrast to our monitoring setting where a *single* execution of a black-box system is dynamically observed during execution. Our work belongs specifically to the field of runtime verification [4], which is concerned with the evaluation of temporal properties over program traces. While much of the research in this domain assumes finite-state monitors, in this work we study an infinite-state problem based on the model of counter monitors. The expressiveness of different register machines and resource trade-offs for monitoring safety properties involving counters and arithmetic registers is studied in [10]. Another infinite-state model for monitoring is that of quantified event automata [3], which combine finite automata specifications with first-order quantification. Other quantitative automata machines are surveyed in [7].

The computation of aggregates over an ongoing system execution in real time was considered in various areas of computer science. Stream expressions [8, 9] and quantitative regular expressions [2] provide frameworks for the specification of transducers over data streams. The work on runtime verification and stream processing can be seen as solving real-time monitoring problems, and very rarely assumes a probabilistic model. A notable exception can be found in [22], who propose to use hypothesis testing to provide an interval of confidence on the monitor outcome when evaluating some probabilistic property. In the vast literature from runtime verification to online algorithms, the problem of limit monitoring as defined, solved, and applied in this paper was, to the best of our knowledge, not studied before.

It is well-known that certain common statistical indicators can be computed in real time. For example, the average can be computed by simply maintaining the sum and sample size. Perhaps more surprisingly, the variance and covariance of a sequence can also be computed

in one pass through classical online algorithms [24]. However, other indicators, like the median, are hard or impossible to compute in real time. Offline algorithms for the median include selection algorithms (e.g., quickselect [13]) with $O(n)$ run time (versus $O(n \log n)$ for sorting), median of medians [5] (which is approximate), and the randomized algorithm of Mitzenmacher & Upfal [18]. The best known online algorithm uses two heaps to store the lower and higher half of values (i.e., all samples have to be stored), with an amortized cost of $O(\log n)$ per input. To the best of our knowledge, no real-time algorithm to compute the median exactly was proposed in the literature.

Statistical properties of subword frequencies in Markov chains are studied in [6]. In Markov chain theory, the existence, uniqueness, and convergence results for stationary distributions are among the most fundamental results [19]. The rate of convergence towards a stationary distribution is called mixing time [17]. In general, the mixing time is controlled by the spectral gap of the transition matrix, with precise results only known for particular random processes, like card shuffling. These results do not lead to bounds on the convergence rate of frequencies of events in labeled Markov chains.

An indirect (and somewhat degenerate) approach to monitoring would be to first learn the monitored system, and then perform offline verification on the learned model. Learning probabilistic generators was studied in the setting of automata learning [20], but requires more powerful oracle queries like membership and equivalence. Rudich showed that the structure and transition probabilities of a Markov chain can, in principle, be learned from a single input sequence [21]. However, the algorithm is impractical as it essentially enumerates all possible structures.

2 Definitions

Let Σ be a finite alphabet of events. Given a finite or infinite word or ω -word $w \in \Sigma^* \cup \Sigma^\omega$ and a position i , $1 \leq i \leq |w|$, we denote by w_i its i 'th value. Given a pair of positions i and j , $i \leq j$, we denote by $w_{i..j}$ the *infix* of w from i to j , such that $|w_{i..j}| = j - i + 1$ and $(w_{i..j})_k = w_{i+k-1}$ for all $1 \leq k \leq j - i + 1$. We denote by $w_{..i} = w_{1..i}$ the *prefix* of w of length i . For any word $w \in \Sigma^*$ and letter $a \in \Sigma$ we write $|w|_a$ for the number of occurrences of a in w .

2.1 Sequential Statistics

We define a statistic to be any function that outputs an indicator for a given input word.

► **Definition 1** (Statistic). *Let Σ be a finite alphabet and Λ be an output domain. A statistic is a function $\mu : \Sigma^* \rightarrow \Lambda$.*

In this paper we focus on statistics that are based on the frequency, or number of occurrence, of events. Two typical examples are the *mode*, i.e. the most frequent event, and the *median*, i.e., the value separating as evenly as possible the upper half from the lower half of a data sample.

► **Example 2** (Mode). We say that $a \in \Sigma$ is the *mode* of w when $|w|_a > |w|_\sigma$ for all $\sigma \in \Sigma \setminus \{a\}$. We denote by $\text{mode} : \Sigma^* \rightarrow \Sigma \uplus \{\perp\}$ the statistic that maps a word to its mode if it exists, or to \perp otherwise.

► **Example 3** (Median). Let Σ be ordered by \prec . We say that $a \in \Sigma$ is the *median* of w when $\sum_{\sigma \succ a} |w|_\sigma < \sum_{\sigma \preceq a} |w|_\sigma$ and $\sum_{\sigma \prec a} |w|_\sigma < \sum_{\sigma \succeq a} |w|_\sigma$. We denote by $\text{median} : \Sigma^* \rightarrow \Sigma \uplus \{\perp\}$ the statistic that maps a word to its median if it exists, or to \perp otherwise.

An example of a statistic that takes into account the order of events in a word is the most frequent event that occurs right after some dedicated event.

2.2 Counter Monitors

The task of a monitor is to compute a statistic in real time. We define a variant of monitor machines that allows us to classify a monitor based on the amount of resources it uses. We adapt the definition of counter monitors set in [10] to our setting of monitoring frequencies.

Let X be a set of integer variables, called *registers* or *counters*. Registers can be read and written according to relations and functions in the signature $S = \langle 0, +1, \leq \rangle$ as follows:

- A *test* is a conjunction of atomic formulas over S and their negation;
- An *update* is a mapping from variables to terms over S .

The set of tests and updates over X are denoted $\Phi(X)$ and $\Gamma(X)$, respectively.

► **Definition 4 (Counter Monitor).** A counter monitor is a tuple $\mathcal{A} = (\Sigma, \Lambda, X, Q, \lambda, s, \Delta)$, where Σ is an input alphabet, Λ is an output alphabet, X is a set of registers, Q is a set of control locations, $\lambda : Q \times \mathbb{N}^X \rightarrow \Lambda$ is an output function, $s \in Q$ is the initial location, and $\Delta \subseteq Q \times \Sigma \times \Phi(X) \times \Gamma(X) \times Q$ is a transition relation such that for every location $q \in Q$, event $\sigma \in \Sigma$, and valuation $v : X \rightarrow \mathbb{N}$ there exists a unique edge $(q, \sigma, \phi, \gamma, q') \in \Delta$ such that $v \models \phi$ is satisfied. The sets Σ, X, Q, Δ are assumed to be finite.

A run of the monitor \mathcal{A} over a word $w \in \Sigma^* \cup \Sigma^\omega$ is a sequence of transitions $(q_1, v_1) \xrightarrow{w_1} (q_2, v_2) \xrightarrow{w_2} \dots$ labeled by w such that $q_1 = s$ and $v_1(x) = 0$ for all $x \in X$. Here we write $(q, v) \xrightarrow{\sigma} (q', v')$ when there exists an edge $(q, \sigma, \phi, \gamma, q') \in \Delta$ such that $v \models \phi$ and $v'(x) = v(\gamma(x))$ for all $x \in X$. There exists exactly one run of a given counter monitor \mathcal{A} over a given word w .

► **Definition 5 (Monitor Semantics).** Every counter monitor \mathcal{A} computes a statistic $\llbracket \mathcal{A} \rrbracket : \Sigma^* \rightarrow \Lambda$, such that $\llbracket \mathcal{A} \rrbracket(w) = \lambda(q, v)$ for (q, v) the final state in the run of \mathcal{A} over $w \in \Sigma^*$.

We remark that the term “counter machine” has various different meanings in the literature and designates machines with varying computational power. In our definition we note the use of the constant 0 which enables resets. Such resets cannot be simulated in real time. On the contrary, arbitrary increments are w.l.o.g., as shown in [11].

2.3 Probabilistic Generators

In this work we model systems as labeled Markov chains, whose executions generate random ω -words.

► **Definition 6 (Markov Chain).** A (finite, connected, labeled) Markov chain is a tuple $\mathcal{M} = (\Sigma, Q, \lambda, \pi, p)$, where Σ is a finite set of events, Q is a finite set of states, $\lambda : Q \rightarrow \Sigma$ is a labeling, π is an initial-state distribution over Q , and $p : Q \times Q \rightarrow [0, 1]$ is a transition distribution with $\sum_{q' \in Q} p(q, q') = 1$ for all $q \in Q$ and whose set of edges (q, q') such that $p(q, q') > 0$ forms a strongly connected graph.

In the rest of this paper, even when not explicitly stated, every Markov chain is assumed to be finite and connected.

Let $\mathcal{M} = (\Sigma, Q, \lambda, \pi, p)$ be a Markov chain. A random infinite sequence $(X_i)_{i \geq 1}$ of states is an execution of \mathcal{M} , *Markov*(\mathcal{M}) for short, if (i) X_1 has distribution π and (ii) conditional on $X_i = q$, X_{i+1} has distribution $q' \mapsto p(q, q')$ and is independent of X_1, \dots, X_{i-1} . By extension, a random ω -word w is *Markov*(\mathcal{M}) if $w_i = \lambda(X_i)$ for all $i \geq 1$.

We denote by $V_q(k) = \sum_{i=1}^k 1_{\{X_i=q\}}$ the *number of visits* to state q within k steps, and by $T_q = \inf\{i > 1 \mid X_i = q\}$ the *first time of visiting* state q (after the initial state). Then $m_q = \mathbb{E}(T_q \mid X_1 = q)$ is the *expected return time* to state q . The ergodic theorem for Markov chains states that the long-run proportion of time spent in each state q is the inverse of m_q . Thus we call $f_q = \frac{1}{m_q}$ the (long-run) *frequency* of q .

► **Theorem 7** (Ergodic Theorem [19]). *Let \mathcal{M} be a finite connected Markov chain. If $(X_i)_{i \geq 1}$ is Markov(\mathcal{M}) then $V_q(n)/n \xrightarrow{a.s.} f_q$ as $n \rightarrow \infty$ for every state q .*

Now summing the frequencies of all states mapped to a letter σ gives the expected frequency of σ , $f_\sigma = \sum_{\substack{q \in Q \\ \lambda(q)=\sigma}} f_q$, as characterized by the following corollary.

► **Corollary 8.** *Let \mathcal{M} be a finite connected Markov chain. If w is Markov(\mathcal{M}) then $|w_{..n}|_\sigma/n \xrightarrow{a.s.} f_\sigma$ as $n \rightarrow \infty$ for every letter σ .*

3 The Limit-Monitoring Problem

We want to monitor the value of a given statistic $\mu : \Sigma^* \rightarrow \Lambda$ over the execution of some (probabilistic) process \mathcal{P} . This execution is potentially infinite, forming an ω -word $w \in \Sigma^\omega$. In practice, the statistic μ is often used as an estimator of some parameter $v \in \Lambda$ of process \mathcal{P} . Such a parameter is always well-defined in the case where μ converges to v as follows.

► **Definition 9** (Convergence). *A statistic $\mu : \Sigma^* \rightarrow \Lambda$ (almost surely) converges to a value $v \in \Lambda$ over a random process \mathcal{P} , written $\mu(\mathcal{P}) = v$, if $\mathbb{P}_{w \sim \mathcal{P}}(\lim_{n \rightarrow \infty} \mu(w_{..n}) = v) = 1$.*

Computing the value of the statistic μ over every finite prefix of w can be an objective in itself. It gives us the most precise estimate of the parameter v when defined. A monitor fulfilling this requirement is called *real-time*. Such a monitor is past-oriented, and is concerned with computing accurately the value $\mu(w_{..n})$ of the statistic at step n , for all n .

► **Definition 10** (Real-Time Monitoring). *A monitor \mathcal{A} is a real-time monitor of statistic μ , if $[[\mathcal{A}]] = \mu$.*

However, if the aim of the monitor is to serve as an estimator of the parameter v , then it may not be strictly required to output the exact value of μ at every step, as long as its output almost surely converges to v . A monitor that almost surely converges to v is qualified as *limit*. Such a monitor is future-oriented, and is concerned with the asymptotic value of the statistic μ as time tends to infinity, not necessarily computing its precise value over each prefix of the computation.

► **Definition 11** (Limit Monitoring). *A monitor \mathcal{A} is a limit monitor of statistic $\mu : \Sigma^* \rightarrow \Lambda$ on process \mathcal{P} , when $[[\mathcal{A}]](\mathcal{P}) = v$ if and only if $\mu(\mathcal{P}) = v$ for all $v \in \Lambda$.*

In other words, if the statistic converges then the limit monitor converges to the same value, and if the statistic does not converge then neither does the monitor. To the best of our knowledge, the notion of limit monitoring was not previously considered. By definition, every real-time monitor is trivially also a limit monitor for the corresponding statistic. However, in this paper we show that dedicated limit monitors can be much more efficient.

► **Proposition 12.** *Every real-time monitor of some statistic μ is also a limit monitor of μ , on arbitrary generating processes.*

This is in clear contrast to a common trend in runtime verification, where past-oriented monitoring (inherently deterministic) often turns out to be computationally easier than future-oriented monitoring (requiring nondeterministic simulation).

4 Precise Real-Time Monitoring

In this section we study the real-time monitoring of statistics by counter monitors. Real-time monitors can be seen as monitoring the past in a precise manner. We show that for some common statistics such as the *mode* and *median* statistics this problem is inherently resource-intensive. More precisely, we identify a class of statistical quantities that require at least as many counters as there are events in the input alphabet.

To illustrate the difficulty of monitoring certain statistics in real time, recall the *mode* as defined in Example 2. A straightforward real-time monitor for the mode counts the number of occurrences of each letter σ in a separate counter x_σ . Then σ is the mode if and only if $x_\sigma > x_\rho$ for all $\rho \in \Sigma \setminus \{\sigma\}$. Hence $|\Sigma|$ counters suffice to monitor the mode. But can we do better? Intuitively it seems necessary to keep track of the exact number of occurrences for each individual letter. Indeed, we show in this section that for real-time monitors this number is tight: any real-time counter monitor of the mode must use at least $|\Sigma|$ counters. In many applications where the alphabet Σ is large this may be beyond the amount of resources available for a monitor. While Proposition 12 implies that the mode can also be limit monitored using $|\Sigma|$ counters, we show in the next section that limit monitoring can be much more resource-sparing.

To capture the hardness of real-time monitoring for a whole class of statistics, we start by defining an equivalence relation over words relative to a statistic. Two words are μ -equivalent if it is impossible for μ to distinguish them, even with an arbitrary suffix appended to both words.

► **Definition 13** (μ -Equivalence). *Let μ be a statistic over Σ . Two words $w_1, w_2 \in \Sigma^*$ are μ -equivalent, denoted $w_1 \equiv_\mu w_2$, if $\mu(w_1u) = \mu(w_2u)$ for all words $u \in \Sigma^*$.*

Now we define the notion of a Σ -counting statistic, which states that two equivalent words must have exactly the same number of occurrences per letter, modulo a constant shift across all letters. Intuitively a Σ -counting statistic induces many equivalence classes, too many to be possibly tracked by a counter monitor with less than $|\Sigma|$ counters.

► **Definition 14** (Σ -Counting). *A statistic μ is Σ -counting if $w \equiv_\mu w'$ implies that there exists $n \in \mathbb{Z}$ such that $|w|_\sigma = |w'|_\sigma + n$ for all $\sigma \in \Sigma$.*

► **Proposition 15.** *For any Σ such that $|\Sigma| > 1$ both the mode and the median statistics are Σ -counting.*

To illustrate the definition of Σ -counting, consider the *mode*-equivalent words abc and a over the alphabet $\Sigma = \{a, b, c\}$. The distance for all letter counts is one. Over the alphabet with an additional letter d the two words are not *mode*-equivalent (for example, consider the extensions $aabcd$ and ad), since the distance for the count of d is zero.

We prove that Σ -counting statistics are expensive to monitor by showing that for large n the number of μ -inequivalent words of length at most n is strictly greater than the number of possible configurations reachable by a counter monitor with less than $|\Sigma| - 1$ counters over words of length at most n .

► **Theorem 16.** *Real-time counter monitors of a Σ -counting statistic require $\Omega(|\Sigma|)$ counters.*

As a corollary of Proposition 15 and Theorem 16, we have that precisely monitoring the mode and the median in real time requires roughly as many counters as the size of the alphabet, which is prohibitive in many practical applications.

5 Efficient Limit Monitoring

In this section we develop a new algorithmic framework for efficient limit monitoring of frequency-based statistics. We first present a general monitoring scheme and then instantiate it to derive efficient monitoring algorithms for both mode (Section 5.2) and median (Section 5.3). In Section 6 we present a monitoring algorithm for a general class of frequency properties. While corresponding real-time monitors require a number of counters proportional to the size of the input alphabet, our limit monitors only use a constant number of counters (e.g., four for the mode), independent of the alphabet size. The algorithmic ideas in our monitoring scheme are simple and intuitive, which makes our algorithms easy to understand, implement, and deploy. However, the correctness proofs are surprisingly hard and required us to develop a new ergodic theory for Markov chains that takes limits over arbitrary subwords (Section 5.1).

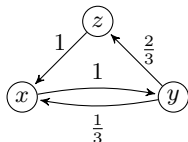
Our high-level monitoring strategy comprises the following points:

1. Split the input sequence into subwords of increasing length.
2. In every subword, acquire partial information about the statistic.
3. Assemble global information about the statistic across different subwords.

The idea behind splitting the input sequence into subwords is that when the monitored property involves frequencies of many events, then different events can be counted separately over different subwords, which enables us to reuse registers. Because of the probabilistic nature of the generator we can still ensure that, in the long run, the monitor value converges to the limit of the statistic. As we will see, there is great flexibility in how exactly the sequence is partitioned. In principle, the subwords can overlap or leave gaps arbitrarily, as long as the length of the considered subwords grows “fast enough”.

5.1 An Ergodic Theorem over Infixes

Consider the following Markov chain on the left-hand side, and a random ω -word generated by this Markov chain in the table on the right-hand side.



ω -word	x	y	z	x	y	z	x	y	z	x	y	z	x	y	\dots	
Prefixes	0	.5	.33	.25	.4	.33	.29	.38	.33	.4	.36	.33	.38	.36	.33	.38 $\xrightarrow{\text{a.s.}} \frac{3}{8}$
Infixes	0		.5			.33			.5				.2			$\xrightarrow{\text{a.s.}} \frac{3}{8}$

The second row of the table shows the frequency of state y in prefixes of increasing length. For example, after $xyzx$ we have frequency $\frac{1}{4}$. The classic ergodic theorem (Theorem 7) tells us that this frequency almost surely converges to $f_y = \frac{3}{8}$, the inverse of the expected return time to y . However, this theorem does not apply to take a limit over arbitrary subwords, for example, the infixes of increasing length (indicated by vertical lines) in the third row of the table. We prove a result that shows that also in this much more general case the limit frequency of y is $\frac{3}{8}$.

The strong law of large numbers states that the empirical average of i.i.d. random variables converges to their expected value, i.e., $(X_1 + \dots + X_n)/n \xrightarrow{\text{a.s.}} \mathbb{E}(X_1)$ as $n \rightarrow \infty$. The fact that random variables are “reused” from the n 'th to the $(n+1)$ 'st sample does matter in this statement. Otherwise the mere existence of a mean value is not sufficient to guarantee convergence. However, when the variance (or higher-order moment) is bounded, then this “reuse” is no longer required. We now prove such a variant of the law of large numbers.²

² Such a setting is sometimes called array of rowwise independent random variables in the literature, see [14] in particular.

► **Theorem 17.** *Let $\{X_{n,i} : n, i \geq 1\}$ be a family of identically distributed random variables with $\mathbb{E}(X_{1,1}) = \mu$ and $\mathbb{E}(X_{1,1}^4) < \infty$, such that $\{X_{n,i} : i \geq 1\}$ are mutually independent for every $n \geq 1$. Let $(s_n)_{n \geq 1}$ be a sequence of indices with $s_n \geq an$ for every $n \geq 1$ and fixed $a > 0$. Set $S_n = \sum_{i=1}^{s_n} X_{n,i}$. Then $S_n/s_n \xrightarrow{a.s.} \mu$ as $n \rightarrow \infty$.*

In our proof the combination of the fourth-moment bound and the linear increase of s_n leads to a converging geometric series. We believe that these assumptions could be slightly relaxed to a second-moment bound or to sublinearly increasing sequences. Theorem 17 already gives a basis to reason about infix-convergence for i.i.d. processes. We now use it to derive a corresponding result for Markov chains.

Let \mathcal{M} be a Markov chain and $(X_i)_{i \geq 1}$ be *Markov*(\mathcal{M}). Given an *offset function* $s : \mathbb{N} \rightarrow \mathbb{N}$, we refer to $X_{s(n)+1}X_{s(n)+2} \cdots$ as the *n*'th *suffix* of X . We denote by $V_q^n(k) = \sum_{i=1}^k \mathbb{1}_{\{X_{s(n)+i}=q\}}$ the *number of visits* to state q within k steps in the *n*'th suffix. We generalize the classic ergodic theorem for Markov chains (Theorem 7) to take the limit over arbitrary subwords.

► **Theorem 18.** *Let \mathcal{M} be a finite connected Markov chain and s an offset function. If $(X_i)_{i \geq 1}$ is *Markov*(\mathcal{M}) then $V_q^n(n)/n \xrightarrow{a.s.} f_q$ as $n \rightarrow \infty$ for every state q .*

Our proof applies Theorem 17 to the i.i.d. excursion times between visiting state q within the *n*'th suffix. This requires bounding the moments of excursion times and showing that the time until visiting q for the first time in every subword becomes almost surely negligible for increasing size subwords. As a corollary of Theorem 18 we get the following characterization for the long-run frequencies of letters over infixes.

► **Corollary 19.** *Let \mathcal{M} be a finite connected Markov chain and s an offset function. If w is *Markov*(\mathcal{M}) then $|w_{s(n)+1..s(n)+n}|_\sigma/n \xrightarrow{a.s.} f_\sigma$ as $n \rightarrow \infty$ for every letter σ .*

5.2 Monitoring the Mode

As we saw in Section 4, precisely monitoring the mode in real time requires at least $|\Sigma|$ counters. By contrast, we now show that the mode can be limit monitored using only four counter registers. For convenience we also use two registers to store event letters; since we assume Σ to be finite they can be emulated in the finite state component of the monitor.

The core idea of our monitoring algorithm is to split w into chunks, and for each chunk only count the number of occurrences of two letters x and y . Letter x is considered the current candidate for the mode and y is a randomly selected contender. If x does not occur more frequently than y in the current chunk, y becomes the mode candidate for the next chunk. The success of the monitor relies on two points: (i) it must be repeatably possible for the true mode to end up in x , and (ii) it must be likely for the true mode to eventually remain in x . The first point is achieved by taking y randomly, and the second point is achieved by gradually increasing the chunk size. It is sufficient to increase the chunk size by one and decompose w as follows:

$$\underbrace{\sigma_1}_{\text{chunk 1}} \underbrace{\sigma_2\sigma_3}_{\text{chunk 2}} \underbrace{\sigma_4\sigma_5\sigma_6}_{\text{chunk 3}} \underbrace{\sigma_7\sigma_8\sigma_9\sigma_{10}}_{\text{chunk 4}} \underbrace{\sigma_{11} \cdots}_{\text{chunk 5}}$$

Formally, the decomposition of w into chunks is given by an offset function $s : \mathbb{N} \rightarrow \mathbb{N}$ with $s(n) = \frac{n(n-1)}{2}$, such that the *n*'th chunk starts at $s(n) + 1$ and ends at $s(n) + n$. For convenience, we introduce a double indexing of w by $n \geq 1$ and $1 \leq i \leq n$, such that $w_{n,i} = w_{s(n)+i}$ is the *i*'th letter in the *n*'th chunk.

■ **Algorithm 1** Mode monitor.

```

1 Function Init( $\sigma$ ):
2    $x, y := \sigma, \sigma$ 
3    $c_x, c_y := 0, 0$ 
4    $n, i := 2, 1$ 
5   return  $x$ 
6 Function Next( $\sigma$ ):
7   if  $i = 1$  then
8     if  $c_x \leq c_y$  then  $x := y$ 
9      $y := \sigma$ 
10     $c_x, c_y := 0, 0$ 
11
12   if  $x = \sigma$  then  $c_x := c_x + 1$ 
13   if  $y = \sigma$  then  $c_y := c_y + 1$ 
14
15   if  $i = n$  then  $n, i := n + 1, 1$ 
16     else  $i := i + 1$ 
17   return  $x$ 

```

■ **Algorithm 2** Median monitor.

```

1 Function Init( $\sigma$ ):
2    $x := \sigma$ 
3    $c_1, c_2, c_3, c_4 := 0, 0, 0, 0$ 
4    $n, i := 2, 1$ 
5   return  $x$ 
6 Function Next( $\sigma$ ):
7   if  $i = 1$  then
8     if  $c_1 \geq c_2$  then  $x := pre_{\prec}(x)$ 
9     if  $c_3 \geq c_4$  then  $x := succ_{\prec}(x)$ 
10     $c_1, c_2, c_3, c_4 := 0, 0, 0, 0$ 
11
12   if  $\sigma < x$  then  $c_1 := c_1 + 1$ 
13   if  $\sigma \geq x$  then  $c_2 := c_2 + 1$ 
14   if  $\sigma > x$  then  $c_3 := c_3 + 1$ 
15   if  $\sigma \leq x$  then  $c_4 := c_4 + 1$ 
16   if  $i = n$  then  $n, i := n + 1, 1$ 
17     else  $i := i + 1$ 
18   return  $x$ 

```

A formal description of our mode monitor is given in Algorithm 1. The counters n and i keep track of the decomposition of w . For the very first letter σ , `Init` initializes both registers x and y to σ (line 2). Then, for every subsequent letter, `Next` counts an occurrence of x and y using counters c_x and c_y , respectively (line 12-13). At the beginning of every chunk, x is replaced by y if it did not occur more frequently in the previous chunk (line 8), and y is set to the first letter of the chunk (line 9). At every step, x is the current estimate of the mode.

► **Example 20.** For alphabet $\Sigma = \{a, b, c\}$ and probability distribution p with $p(a) = 0.5$, $p(b) = 0.3$, and $p(c) = 0.2$, the following table shows a word w where every letter was independently sampled from p , and the corresponding mode at every position in w .

w	c b b a b a c a a b c a c a a a ...
mode	c - b b b b b - a - - a a a a a ...

In this example, `mode` first switches between the different letters and undefined, but then eventually seems to settle on a . We show that this is not an accident, but happens precisely because a is the unique letter that p assigns the highest probability.

Now the following table shows the execution of Algorithm 1 on the same random word.

n	1	2	3	4	5	6 ...
i	1	1 2	1 2 3	1 2 3 4	1 2 3 4 5	1 ...
σ	c	b b	a b a	c a a b	c a c a a	a ...
x	c	c	b	a	a	a ...
y	c	b	a	c	c	a ...
c_x	1	0 0	0 1 1	0 1 2 2	0 1 1 2 3	1 ...
c_y	1	1 2	1 1 2	1 1 1 1	1 1 2 2 2	1 ...

Initially c is considered the mode and compared to b in the second chunk, where b occurs more frequently. Thus b is considered the mode and compared to a in the third chunk, where a occurs more frequently. In the fourth and fifth chunk a is compared to c , where a occurs more frequently in both chunks. Again, the algorithm seems to settle on a , the true mode.

To prove the correctness of our algorithm according to Definition 11 requires us to first characterize when a Markov chain has a mode, i.e., under which conditions the mode statistic almost surely converges. For this it is illustrative to instantiate Definition 9 for the mode, which states that a is the mode of an ω -word w if there exists a length n , such that for every length $n' \geq n$, $|w_{..n'}|_a > |w_{..n'}|_b$ for every $b \neq a$. In a Markov chain the ergodic theorem characterizes the long-run frequencies of states, and thus the long-run frequencies of letters (see Corollary 8). Hence a Markov chain has a mode if and only if its random ω -word almost surely has a unique letter that occurs most frequently.

► **Theorem 21.** *Over Markov chains, the mode statistic converges to a if and only if $f_a > f_b$ for all $b \neq a$.*

Proof. Let \mathcal{M} be a Markov chain and w be $\text{Markov}(\mathcal{M})$. According to Corollary 8, $|w_{..n}|_\sigma/n \xrightarrow{\text{a.s.}} f_\sigma$ as $n \rightarrow \infty$ for every $\sigma \in \Sigma$

Now assuming $f_a > f_b$ for all $b \neq a$, we have for sufficiently large n that $|w_{..n}|_a > |w_{..n}|_b$ for all $b \neq a$, and thus a is the mode of w almost surely.

Conversely, if there are two distinct letters a, a' with equal maximal frequencies $f_a, f_{a'}$, then almost surely the mode switches infinitely often between a and a' , thus neither a nor a' is the mode of w , and thus w does not have a mode. ◀

Now we can prove that Algorithm 1 is a limit monitor for the mode. The core of the argument is that the probability of the true mode eventually staying in register x is lower-bounded by the probability of a eventually being the most frequent letter in *every* subword and a being eventually selected into y , which happens almost surely.

► **Theorem 22.** *Algorithm 1 limit-monitors the mode over Markov chains.*

Proof. Let w be $\text{Markov}(\mathcal{M})$ and let a be the mode of w (the other case where w does not have a mode is obvious). Let γ_n be the function that maps every letter to the number of its occurrences in the n 'th subword, i.e., $\gamma_n(\sigma) = |w_{s(n)+1..s(n)+n}|_\sigma$. To capture Algorithm 1 mathematically, we define the random variables

$$Y_n = w_{n,1}; \quad X_1 = w_{1,1}; \quad X_{n+1} = \begin{cases} X_n, & \text{if } \gamma_n(X_n) > \gamma_n(Y_n); \\ Y_n, & \text{if } \gamma_n(X_n) \leq \gamma_n(Y_n). \end{cases}$$

That is, X_n and Y_n are the values of x and y throughout the n 'th subword. We need to show that almost surely, eventually $X_n = a$ forever, i.e., $\mathbb{P}(\diamond \square X_n = a) = 1$.³

It is more likely that a eventually stays in x forever as that a eventually is the most frequent letter in *every* subword and that a is also eventually sampled into y :

$$\begin{aligned} & \mathbb{P}(\diamond \square X_n = a) \\ & \geq \mathbb{P}(\diamond (\square \forall b \neq a : \gamma_n(b) < \gamma_n(a)) \wedge (\diamond Y_n = a)) \\ & \geq \mathbb{P}((\square_{\geq n_0} \forall b \neq a : \gamma_n(b) < \gamma_n(a)) \wedge (\diamond_{\geq n_0} Y_n = a)) \end{aligned}$$

The last lower bound holds for any fixed n_0 and we show that it converges to 1 as $n_0 \rightarrow \infty$.

$$\begin{aligned} & \mathbb{P}((\square_{\geq n_0} \forall b \neq a : \gamma_n(b) < \gamma_n(a)) \wedge (\diamond_{\geq n_0} Y_n = a)) \\ & \geq \mathbb{P}(\square_{\geq n_0} \forall b \neq a : \gamma_n(b) < \gamma_n(a)) \cdot \mathbb{P}(\diamond_{\geq n_0} Y_n = a) \\ & = \mathbb{P}(\square_{\geq n_0} \forall b \neq a : \gamma_n(b) < \gamma_n(a)) \end{aligned}$$

³ In the interest of readability we use temporal (modal) logic notation \diamond and \square meaning *eventually* and *forever*, respectively.

Since $\gamma_n(\sigma)/n \xrightarrow{\text{a.s.}} f_\sigma$ (by Corollary 19) and a is the unique letter with highest frequency f_a (by Theorem 21), we have $\mathbb{P}(\square_{\geq n_0} \forall b \neq a : \gamma_n(b) < \gamma_n(a)) = 1$ for sufficiently large n_0 . Thus, $\mathbb{P}(\diamond \square X_n = a) = 1$. \blacktriangleleft

Note that our policy of always selecting the mode contender y from the input is an optimization, since we expect to see the mode often in the input. Our proof requires that the true mode is selected into y infinitely often, which is the case because we update y at irregular positions. Two other policies to update y would be (i) to always uniformly sample from Σ , or (ii) to cycle deterministically through all elements of Σ .

5.3 Monitoring the Median

Recall from Example 3 that a is the *median* of a word w over a \prec -ordered alphabet Σ when

$$\sum_{\sigma \succ a} |w|_\sigma < \sum_{\sigma \preceq a} |w|_\sigma \quad (1)$$

on the one hand, and

$$\sum_{\sigma \prec a} |w|_\sigma < \sum_{\sigma \succeq a} |w|_\sigma \quad (2)$$

on the other hand. These equations readily lead to our median limit-monitoring algorithm shown in Algorithm 2, which we display next to our mode monitor to highlight their common structure. The idea of the algorithm is to maintain a median candidate x and then use four counters c_1, c_2, c_3, c_4 to compute the sums in inequality (1) and (2), for $a = x$, in every subword (line 11-14). Whenever any of the two inequalities is not satisfied at the end of a subword, a new median candidate is selected into x for the next subword. In particular, if inequality (1) is violated then the next lower value in the ordering \prec is selected (line 8), and if inequality (2) is violated then the next higher value is selected (line 9). Notice that we could eliminate the counters c_3, c_4 , by alternating the computation of inequality (1) and (2) over different subwords, and thus reusing c_1, c_2 to compute inequality (2).

► **Theorem 23.** *Algorithm 2 limit-monitors the median over Markov chains.*

6 Monitoring General Frequency Properties

In the previous section we presented high-level principles for efficient limit monitoring and designed specialized monitoring algorithms for the mode and median statistic, which are both derived from event frequencies. We postulate that our algorithmic ideas are straightforward to adapt to obtain monitors for many other frequency-based statistics. However, we did not yet precisely define what we mean by *frequency property*, nor demonstrated how efficiently these can be limit monitored in general. In this section we provide a first step in this direction by defining a simple language to specify frequency-based Boolean statistics, and showing that all statistics definable in this language can be limit monitored over Markov chains with four counters only.

From the defining equations of the mode and median we observe that a characteristic construction is the formation of linear inequalities over the frequencies (or equivalently, occurrence counts) of specific events. The key part of the argument for the correctness of our monitoring algorithms is that since event frequencies almost surely converge, both over prefixes and infixes, also these inequalities almost surely “stabilize”. We use the same construction at the core of a language to define general frequency-based statistics. For simplicity we focus on statistics that output a Boolean value.

► **Definition 24.** A frequency formula over alphabet Σ is a Boolean combination of atomic formulas of the form

$$\sum_{\sigma \in \Sigma} \alpha_{\sigma} \cdot f_{\sigma} > \alpha \quad (3)$$

where all α 's are integer coefficients.

A frequency formula ϕ is built from linear inequalities over frequencies of events. The evaluation of a frequency formula is as expected (we write $w \models \phi$ if ϕ evaluates to true over w). Hence we see ϕ as defining the Boolean statistic $\llbracket \phi \rrbracket : \Sigma^* \rightarrow \mathbb{B}$, where

$$\llbracket \phi \rrbracket(w) = \begin{cases} 1, & \text{if } w \models \phi; \\ 0, & \text{if } w \not\models \phi. \end{cases}$$

► **Example 25.** The existence of a mode is expressed as the frequency formula

$$\bigvee_{a \in \Sigma} \bigwedge_{\substack{\sigma \in \Sigma \\ \sigma \neq a}} f_a > f_{\sigma}.$$

► **Example 26.** Consider a web server that favors certain client requests over others. Such a malfunction could be observed by detecting that certain events are disproportionately more frequent than others. The following frequency formula specifies that no event can occur 100-times more frequent than any other event:

$$\bigwedge_{\substack{a, b \in \Sigma \\ a \neq b}} f_a < 100 \cdot f_b.$$

A frequency formula ϕ can be limit monitored by simply evaluating ϕ repeatedly over longer and longer subwords. However, the key to save resources is to evaluate different atomic subformulas of ϕ over different subwords, and thus only evaluating one subformula at a time.

► **Theorem 27.** Over Markov chains, every frequency formula can be limit monitored using 4 counters.

Proof. Let ϕ be a frequency formula with k atomic subformulas ϕ_1, \dots, ϕ_k of the form (3). The monitor partitions the input word w into infixes $w_{n,i}$ with $|w_{n,i}| = n$, for $n \geq 1$ and $1 \leq i \leq k$, as follows:

$$\dots \underbrace{w_{n,1} w_{n,2} \dots w_{n,k}}_{\phi} \dots$$

$\underbrace{\quad \quad \quad}_{\phi_1} \quad \underbrace{\quad \quad \quad}_{\phi_2} \quad \underbrace{\quad \quad \quad}_{\phi_k}$

Keeping track of the increasing infix length n and the current position within an infix requires two counters. Then over every infix $w_{n,i}$ the monitor uses two counters to compute ϕ_i , one for positive and one for negative increments. At the end of $w_{n,i}$ we have a truth value for ϕ_i that is used to partially evaluate ϕ . This evaluation is implemented in the final-state component of the monitor, and the two counters are reused across all infixes. Then after every k 'th infix we have a new "estimate" of ϕ that in the long run converges the same way as $\llbracket \phi \rrbracket$. Hence the resulting automaton is a limit monitor of ϕ : by Corollary 19, the frequency of each event over infixes of increasing length tends to its respective asymptotic frequency, so that strict inequalities holding over empirical frequencies almost surely hold over infixes of increasing length. ◀

7 Conclusion

In this paper we have studied the monitoring of frequency properties of event sequences. We observed that real-time monitoring can be surprisingly hard (i.e., resource-intensive) for such properties, and introduced the alternative notion of limit monitoring. In this limit-monitoring setting we showed that a simple algorithmic idea leads to resource-efficient monitoring algorithms for frequency properties. To prove the correctness of our algorithms we generalized the ergodic theory of Markov chains.

The results in this paper are a first indicator of the relevance and potential of limit monitoring. We hope that future research broadens the understanding of this problem and we close with a number of interesting directions.

First, we are interested in a tighter characterization of properties that can be efficiently limit monitored. Let us remark that the results in this paper immediately generalize from counting individual events to counting the occurrences of regular event patterns. This is the case because regular expression matching can be performed in real time by the finite state component of a counter monitor. We extended our frequency formulas with free variables to support non-Boolean statistics, and quantification to reason about unknown alphabet symbols. However, the shape and efficiency of a generic monitoring algorithm is not yet clear. For examples, we saw that there are different policies to partition the input sequence and different policies to obtain candidate values for the monitor output. Certain forms of existential quantification can be translated to random sampling, but this does not seem to hold in general since not all events in the alphabet may occur in the execution under consideration. Going even further, it would be interesting to consider limit monitoring of properties with temporal aspects (such as *always* and *eventually* modalities).

Second, it is well known (see e.g. [6]) that the asymptotic frequencies of k -long subwords fully characterize a k -state connected Markov chain. Hence the transition probabilities of a Markov chain (of known structure) can be inferred from the conditional probabilities of events. Thus, assuming the structure of a Markov chain is known, frequency queries and the algorithmic ideas in this paper can be used to learn its transition probabilities to an arbitrary precision. It would be interesting to study more broadly “how much” of a system can be learned from frequency properties (and similar observations).

Third, throughout this paper we used the term efficient to mean resource-efficient in the amount of memory used by a monitor. However, there is the orthogonal question of time-efficiency. For a limit monitor this means how quickly a monitor converges in relation to the monitored statistic. We hope that future research can provide numerical guarantees or estimates for convergence rates. For the simple setting of an i.i.d. word over a two-letter alphabet, we proved that the mode statistic converges exponentially fast. More precisely, if w is a random ω -word where every letter is i.i.d. according to a probability distribution p over $\{a, b\}$ with $p(a) > p(b)$, then $\mathbb{P}(\text{mode}(w_{..n}) = a) \geq 1 - (4p(a)p(b))^{\lfloor \frac{n}{2} \rfloor}$. Since this depends on the exact probabilities, the analytical expressions of the confidence value seem to become intractable for three letters or more. In probability theory, there exist several different notions of convergence of random variables. The results in this paper use the notion of almost-sure convergence of a statistic μ (Definition 9), that is, $\mathbb{P}_{w \sim \mathcal{P}}(\lim_{n \rightarrow \infty} \mu(w_{n..}) = v) = 1$. It would be interesting to study also other notions, for example convergence in probability, that is, $\lim_{n \rightarrow \infty} \mathbb{P}_{w \sim \mathcal{P}}(\mu(w_{n..}) = v) = 1$.

Fourth, the correctness results we derived for our monitoring algorithms hold for systems modeled as connected Markov chains. However, we believe that the algorithmic ideas of this paper are more widely applicable. Thus it would be interesting to study limit monitoring

for other types of systems, for example, Markov decision processes which are challenging for our monitoring scheme because nondeterminism allows certain events to *always* occur deliberately when the monitor is not watching for them. In the security context a monitored system is usually assumed to be adversarial, not probabilistic. It could be interesting to turn our deterministic monitors of probabilistic systems into probabilistic monitors for nondeterministic systems.

References

- 1 Gul Agha and Karl Palmskog. A Survey of Statistical Model Checking. *ACM Trans. Model. Comput. Simul.*, 28(1):6:1–6:39, 2018. doi:10.1145/3158668.
- 2 Rajeev Alur, Dana Fisman, and Mukund Raghothaman. Regular Programming for Quantitative Properties of Data Streams. In *ESOP*, volume 9632 of *Lecture Notes in Computer Science*, pages 15–40. Springer, 2016. doi:10.1007/978-3-662-49498-1_2.
- 3 Howard Barringer, Yliès Falcone, Klaus Havelund, Giles Reger, and David E. Rydeheard. Quantified Event Automata: Towards Expressive and Efficient Runtime Monitors. In *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 68–84. Springer, 2012. doi:10.1007/978-3-642-32759-9_9.
- 4 Ezio Bartocci and Yliès Falcone, editors. *Lectures on Runtime Verification - Introductory and Advanced Topics*, volume 10457 of *Lecture Notes in Computer Science*. Springer, 2018. doi:10.1007/978-3-319-75632-5.
- 5 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time Bounds for Selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.
- 6 Taylor L. Booth. Statistical Properties of Random Digital Sequences. *IEEE Trans. Computers*, 17(5):452–461, 1968. doi:10.1109/TC.1968.226909.
- 7 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Quantitative Monitor Automata. In *SAS*, volume 9837 of *Lecture Notes in Computer Science*, pages 23–38. Springer, 2016. doi:10.1007/978-3-662-53413-7_2.
- 8 Ben D’Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra, and Zohar Manna. LOLA: runtime monitoring of synchronous systems. In *TIME*, pages 166–174. IEEE Computer Society, 2005. doi:10.1109/TIME.2005.26.
- 9 Peter Faymonville, Bernd Finkbeiner, Maximilian Schwenger, and Hazem Torfah. Real-time Stream-based Monitoring, 2019. arXiv:1711.03829v4.
- 10 Thomas Ferrère, Thomas A. Henzinger, and N. Ege Saraç. A Theory of Register Monitors. In *LICS*, pages 394–403. ACM, 2018. doi:10.1145/3209108.3209194.
- 11 Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. Counter Machines and Counter Languages. *Mathematical Systems Theory*, 2(3):265–283, 1968. doi:10.1007/BF01694011.
- 12 Thomas A. Henzinger. Quantitative reactive modeling and verification. *Computer Science - R&D*, 28(4):331–344, 2013. doi:10.1007/s00450-013-0251-7.
- 13 C. A. R. Hoare. Algorithm 65: find. *Commun. ACM*, 4(7):321–322, 1961. doi:10.1145/366622.366647.
- 14 Tien-Chung Hu, F Moricz, and R Taylor. Strong laws of large numbers for arrays of rowwise independent random variables. *Acta Mathematica Hungarica*, 54(1-2):153–162, 1989. doi:10.1007/BF01950716.
- 15 Marta Kwiatkowska. Quantitative verification: models techniques and tools. In *ESEC/SIG-SOFT FSE*, pages 449–458. ACM, 2007. doi:10.1145/1287624.1287688.
- 16 Marta Kwiatkowska, Gethin Norman, and David Parker. Probabilistic Model Checking: Advances and Applications. In Rolf Drechsler, editor, *Formal System Verification: State-of-the-Art and Future Trends*, pages 73–121. Springer, 2018. doi:10.1007/978-3-319-57685-5_3.

- 17 David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. Markov Chains and Mixing Times, second edition, 2017. URL: <https://pages.uoregon.edu/dlevin/MARKOV/mcmt2e.pdf>.
- 18 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. doi:10.1017/CB09780511813603.
- 19 James R. Norris. *Markov Chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1998.
- 20 Dana Ron. *Automata Learning and its Applications*. PhD thesis, Hebrew University, 1995. URL: http://www.cs.huji.ac.il/labs/learning/Theses/Dana_Ron_PhD.pdf.
- 21 Steven Rudich. Inferring the Structure of a Markov Chain from its Output. In *FOCS*, pages 321–326. IEEE Computer Society, 1985. doi:10.1109/SFCS.1985.34.
- 22 Usa Sammapun, Insup Lee, and Oleg Sokolsky. RT-MaC: Runtime monitoring and checking of quantitative and probabilistic properties. In *RTCSA*, pages 147–153. IEEE Computer Society, 2005. doi:10.1109/RTCSA.2005.84.
- 23 Zbigniew S. Szewczak. On moments of recurrence times for positive recurrent renewal sequences. *Statistics & Probability Letters*, 78(17):3086–3090, 2008. doi:10.1016/j.spl.2008.05.013.
- 24 B. P. Welford. Note on a Method for Calculating Corrected Sums of Squares and Products. *Technometrics*, 4(3):419–420, 1962. doi:10.1080/00401706.1962.10490022.

Automatic Equivalence Structures of Polynomial Growth

Moses Ganardi

University of Siegen, Siegen, Germany
ganardi@eti.uni-siegen.de

Bakhadyr Khoussainov

University of Auckland, Auckland, New Zealand
bmk@cs.auckland.ac.nz

Abstract

In this paper we study the class EqP of automatic equivalence structures of the form $\mathfrak{E} = (D, E)$ where the domain D is a regular language of polynomial growth and E is an equivalence relation on D . Our goal is to investigate the following two foundational problems (in the theory of automatic structures) aimed for the class EqP . The first is to find algebraic characterizations of structures from EqP , and the second is to investigate the isomorphism problem for the class EqP . We provide full solutions to these two problems. First, we produce a characterization of structures from EqP through multivariate polynomials. Second, we present two contrasting results. On the one hand, we prove that the isomorphism problem for structures from the class EqP is undecidable. On the other hand, we prove that the isomorphism problem is decidable for structures from EqP with domains of quadratic growth.

2012 ACM Subject Classification Theory of computation \rightarrow Formal languages and automata theory

Keywords and phrases automatic structures, polynomial growth, isomorphism problem

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.21

Funding *Moses Ganardi*: This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 731143.

1 Introduction

Automatic structures are relational structures $\mathfrak{A} = (D, R_1, \dots, R_k)$ where the domain D is a regular language and every relation $R_i \subseteq D^{r_i}$ is recognized by a finite automaton with r_i many synchronous heads [3, 8]. They constitute a robust class of finitely presented structures with good algorithmic and often algebraic properties; in particular, the model checking problem for first-order logic (and some of its extensions such as $(FO + \exists^\infty)$ -logic) is decidable over automatic structures [3, 7, 11]. However, going beyond first-order logic, problems quickly become undecidable over automatic structures, e.g. the reachability problem is undecidable for automatic structures.

An important problem in the theory of automatic structures is the isomorphism problem. The problem asks to design an algorithm that given two automatic structures decides if the structures are isomorphic. Blumensath and Grädel proved that the isomorphism problem is undecidable [3]. Furthermore, it turns out that the isomorphism problem for automatic structures is complete for the first level of the analytical hierarchy Σ_1^1 [9]. In addition, Nies [15] proved that the problem remains Σ_1^1 -complete for the class of undirected graphs and partial orders, and Kuske, Liu and Lohrey [4] showed that the problem is Σ_1^1 -complete for even automatic linear orders. In contrast, the isomorphism problem is decidable for automatic ordinals [10] and Boolean algebras [9]. These decidability results follow from full characterization results for automatic ordinals and Boolean algebras [10, 9]. Interestingly,



© Moses Ganardi and Bakhadyr Khoussainov;
licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 21; pp. 21:1–21:16

Leibniz International Proceedings in Informatics



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

21:2 Automatic Equivalence Structures of Polynomial Growth

full characterizations of isomorphism types do not immediately imply decidability of the isomorphism problem for automatic structures. For instance, Thomas and Oliver [16] proved that automata presented finitely generated groups are virtually abelian. However, it is still unknown if the isomorphism problem for this class of automatic groups is decidable.

The class of equivalence structures, these are structures of the form (D, E) where E is an equivalence relation on D , are among the simplest algebraic structures (in terms of descriptions of their isomorphism types). The isomorphism type of each such structure (D, E) is fully characterized by the function $f: \mathbb{N}_+ \cup \{\infty\} \rightarrow \mathbb{N} \cup \{\infty\}$ defined as follows:

$$f(n) = \text{the number of equivalence classes of size } n \tag{1}$$

This description immediately implies that the isomorphism problem for automatic equivalence structures is a Π_1^0 -predicate. It had been a long-standing open question if the isomorphism problem for automatic equivalence structures is decidable. Kuske, Liu and Lohrey [4] proved that the isomorphism problem over automatic equivalence structures is Π_1^0 -complete, and hence undecidable. It is worth to mention the following simple observation from [4]. There is an algorithm that, given two automatic isomorphic equivalence structures, builds a computable isomorphism between them. This is in spite the fact that the isomorphism problem for automatic equivalence relations is undecidable.

In light of the (undecidability) results above, the following question arises. Find classes of automatic structures for which the isomorphism problem is decidable. One approach to address the question is to put algebraic restrictions on the class of automatic structures. For instance, one can consider the classes of automatic torsion free abelian groups and ask if the isomorphism problem for this class of structures is decidable. The second approach is to consider classes of automatic structures whose domains belong to some robust class of regular languages. For instance, in [2, 18, 13] automatic structures with unary domains are studied; it is proved the isomorphism problem is decidable for unary automatic linear orders, equivalence structures, and trees. Although, we still do not know if the isomorphism problem for unary automatic structures is decidable. Bárány [1] initiated the study of automatic structures with domains of polynomial growth. He provided examples of universal structures in this class and proved that the isomorphism problem in this class of structures is undecidable. The third way to address the problem is to combine the above two approaches by restricting both the class of structures and the class of regular domains. This is exactly what we do in this paper. We focus on automatic equivalence structures of the form (D, E) where D is a regular language of polynomial growth and E is an equivalence relation on D . We denote this class of automatic structures by EqP . The choice of this class is partly motivated by the facts mentioned above: (1) The isomorphism types of equivalence structures have full descriptions, and (2) the isomorphism problem for automatic equivalence structures is undecidable.

In this paper we thus address two foundational problems for the class EqP . The first is to find algebraic characterizations of structures from EqP , and the second is to investigate the isomorphism problem for the class EqP . We fully solve these two problems. First, we produce a characterization of automatic equivalence structures from EqP in the language of multivariate polynomials. Second, we present two contrasting results. On the one hand, we prove that the isomorphism problem for automatic structures from the class EqP is undecidable. On the other hand, we prove that the isomorphism problem is decidable for structures from EqP with domains bounded by a quadratic growth.

2 Summary of results

Let $\mathbb{N} = \{0, 1, 2, \dots\}$ and $\mathbb{N}_+ = \{1, 2, \dots\}$ be the sets of nonnegative and positive integers. Polynomials $f \in \mathbb{N}[x_1, \dots, x_k]$ are viewed as functions $f: \mathbb{N}^k \rightarrow \mathbb{N}$. The number k is also denoted by $\text{var}(f)$; the degree of g is denoted by $\text{deg}(g)$.

An *equivalence structure* $\mathfrak{E} = (D, E)$ consists of a domain D and an equivalence relation E on D . We denote by $[x] = \{y \in D \mid (x, y) \in E\}$ the equivalence class of $x \in D$. As described in (1), the isomorphism type of \mathfrak{E} can be described by specifying the number of equivalence classes of every finite or infinite size. Since we will only deal with countable domains, there is only one infinite cardinality. We defer the formal definition of automatic structures to Section 3.

Let D be a regular language. Its growth is the function gr_D that for each n computes the number of strings of length n that belong to D . We say that the language D has a polynomial growth if its growth function gr_D is bounded by a polynomial in n . We denote by EqP the class of all automatic equivalence structures (D, E) such that D is a regular language of polynomial growth. Here is a simple yet an important example of a structure from EqP :

► **Example 1.** Consider the equivalence structure $\mathfrak{E} = (D, E)$ defined as follows: The domain is $D = 0^*1^*2^*3^*$ and the equivalence relation E consists of pairs (u, v) from the domain such that $(u, v) \in E$ if and only if $|u|_0 + |u|_1 = |v|_0 + |v|_1$ and $|u|_2 + |u|_3 = |v|_2 + |v|_3$. Here, $|w|_\sigma$ denotes the number of times σ appears in w . It is easy to see that the equivalence structure \mathfrak{E} is automatic. A *set of representatives* is a subset $R \subseteq D$ containing exactly one element from each equivalence class. An example of a regular set of representatives of E is the language 0^*2^* . Note that the class $[0^{t_0}2^{t_2}]$ has size $(t_0 + 1)(t_2 + 1)$. One could say that the polynomial $g(t_0, t_2) = (t_0 + 1)(t_2 + 1)$ defines \mathfrak{E} up to isomorphism: for each tuple $(t_0, t_2) \in \mathbb{N}^2$ it contains a class of size $g(t_0, t_2)$.

This example suggests us to give the following definition (construction):

► **Definition 2.** For a function $g: \mathbb{N}^k \rightarrow \mathbb{N}$, the equivalence structure $\mathfrak{E}(g)$ is defined (up to isomorphism) as follows. The number of classes of size $s \in \mathbb{N}_+$ in $\mathfrak{E}(g)$ is given by the cardinality $|\{\bar{t} \in \mathbb{N}^k \mid g(\bar{t}) = s\}|$. Furthermore $\mathfrak{E}(g)$ has no infinite classes.

Note that $\mathfrak{E}(g)$ can have infinitely many classes of a certain size s . For instance, if $g(t_0, t_1) = t_0$ is the polynomial in two variables t_0, t_1 , then for all $s \in \mathbb{N}_+$ there are infinitely many classes of size s in $\mathfrak{E}(g)$. However, all classes in $\mathfrak{E}(g)$ are finite. We remark that tuples which are mapped to 0 are irrelevant for the definition of $\mathfrak{E}(g)$. Our characterization theorem for equivalence structures from the class EqP is the following:

► **Theorem 3.** Let \mathfrak{E} be an equivalence structure and $k \in \mathbb{N}$. Then the following statements are equivalent:

1. \mathfrak{E} is isomorphic to an automatic equivalence structure (D, E) where D has growth $O(n^k)$.
2. \mathfrak{E} is a finite disjoint union of equivalence structures $\mathfrak{E}(g_1), \dots, \mathfrak{E}(g_m)$ where each g_i is a polynomial with natural coefficients and $\text{var}(g_i) + \text{deg}(g_i) \leq k + 1$ and a number of infinite classes (which must be finitely many if $k = 0$).

Furthermore, this correspondence is effective.

The decomposition into equivalence structures $\mathfrak{E}(g_i)$ defined by polynomials g_i is obtained by applying a result from Woods [22] who characterized *counting functions* of Presburger definable relations. The bound on the degree and the number of variables is obtained by a growth argument.

21:4 Automatic Equivalence Structures of Polynomial Growth

The characterization theorem provides us with two contrasting results. The first result is undecidability of the isomorphism problem for the class EqP .

► **Theorem 4.** *There exists a number $k \geq 0$ such that it is Π_1^0 -complete to decide whether two automatic equivalence structures of growth $O(n^k)$ are isomorphic.*

The proof of Theorem 4 follows the ideas of the undecidability proof of [4], which uses the MRDP-theorem [14]. The second result is decidability of the isomorphism problem for structures from EqP of quadratic growth:

► **Theorem 5.** *It is decidable whether two given automatic equivalence structures of growth $O(n^2)$ are isomorphic.*

The proof idea of Theorem 5 is to reduce it to equality of multisets defined by quadratic polynomials, which can be decided with the help of the theory of quadratic Diophantine equations. The outline of the paper is as follows. After giving the necessary definitions in Section 3 we prove the characterization theorem (Theorem 3) in Section 4. In Section 5 we prove the undecidability result (Theorem 4) and in Section 6 we prove Theorem 5.

3 Preliminaries

We presuppose basic definitions in regular languages and first-order logic. Let us recall the definition of automatic structures. The *convolution* of k words v_1, \dots, v_k where $v_i = a_{i,1} \cdots a_{i,n_i}$ is the word $(a_{1,1}, \dots, a_{k,1}) \cdots (a_{1,m}, \dots, a_{k,m})$ of length $m = \max\{n_1, \dots, n_k\}$ over the alphabet $\Sigma_\diamond^k = (\Sigma \cup \{\diamond\})^k$ where $a_{i,j} = \diamond$ for all $n_j < i \leq m$ and $1 \leq j \leq k$. It is denoted by $v_1 \otimes v_2 \otimes \cdots \otimes v_k$. A relation $R \subseteq D^k$ over a language D is *automatic* if $\otimes R_i = \{v_1 \otimes \cdots \otimes v_k \mid (v_1, \dots, v_k) \in R\}$ is regular. A relational structure $\mathfrak{A} = (D, R_1, \dots, R_m)$ is *automatic* if the domain D is a regular language and each relation R_i automatic. Given an automatic structure $\mathfrak{A} = (D, R_1, \dots, R_m)$ and a first-order formula $\varphi(\bar{x})$ with infinity quantifiers \exists^∞ , one can compute an automaton recognizing $\otimes \{\bar{v} \mid \mathfrak{A} \models \varphi(\bar{v})\}$, see [8, 3]. Here a formula of the form $\exists^\infty x \varphi(x, \bar{y})$ states that there are infinitely many elements x satisfying $\varphi(x, \bar{y})$. In particular, if $\varphi(x)$ is such a formula then the restriction of \mathfrak{A} to $\{v \in D \mid \mathfrak{A} \models \varphi(v)\}$ is also automatic.

The *growth function* of a language L is the function $n \mapsto |\{w \in L \mid |w| = n\}|$. It is known that a regular language has growth $O(n^k)$ if and only if it can be written as a finite union of languages defined by regular expressions of the form $x_0 y_0^* \cdots x_\ell y_\ell^* x_{\ell+1}$ where $0 \leq \ell \leq k$, see [21]. Furthermore, we can compute such regular expressions such that the union is disjoint and that each expression is *unambiguous*, i.e. the function $(i_0, \dots, i_\ell) \mapsto x_0 y_0^{i_0} \cdots x_\ell y_\ell^{i_\ell} x_{\ell+1}$ is injective, cf. [21, Proof of Lemma 3].

Semilinear sets and Presburger arithmetic. A set $S \subseteq \mathbb{N}^k$ is *semilinear* if it is a finite union of *linear sets*

$$L = \bar{v}_0 + \langle \bar{v}_1, \dots, \bar{v}_n \rangle = \left\{ \bar{v}_0 + \sum_{i=1}^n \lambda_i \bar{v}_i \mid \lambda_1, \dots, \lambda_n \in \mathbb{N} \right\}.$$

A linear set is *fundamental* if the period vectors $\bar{v}_1, \dots, \bar{v}_n$ are linearly independent in \mathbb{R}^k . It is known that every semilinear set is a finite disjoint union of fundamental linear sets [6] and that such a representation can be computed effectively. In the one-dimensional case this means that every semilinear set $S \subseteq \mathbb{N}$ is a finite disjoint union of singleton sets and arithmetic progressions $\{a + bn \mid n \in \mathbb{N}\}$ with $a, b \in \mathbb{N}$, $b \neq 0$.

An important theorem which connects context-free languages and semilinear sets is Parikh's theorem. For an ordered alphabet $\Sigma = \{a_1, \dots, a_k\}$ the *Parikh mapping* $\Phi: \Sigma^* \rightarrow \mathbb{N}^k$ is defined by $\Phi(w) = (|w|_{a_1}, \dots, |w|_{a_k})$. Parikh's theorem states that for every context-free language $L \subseteq \Sigma^*$ the Parikh image $\Phi(L) = \{\Phi(w) \mid w \in L\}$ is effectively semilinear [17]. Recall that *Presburger arithmetic* is the first-order logic over the structure $(\mathbb{N}, +, 0, \leq)$. It is known that a relation $R \subseteq \mathbb{N}^k$ is definable by a Presburger formula $\varphi(x_1, \dots, x_k)$ if and only if R is semilinear, and this correspondence is effective [5].

Counting functions. Given a formula $\varphi(\bar{s}, \bar{t})$ of Presburger arithmetic, we will in our arguments employ the counting function $c(\bar{t}) = |\{\bar{s} \mid \varphi(\bar{s}, \bar{t})\}|$ where we assume that this quantity is finite. For example, given the formula $\varphi(s, t_1, t_2) = \exists x(s = x + x \wedge t_1 \leq s \wedge s \leq t_2)$ the function $c(t_1, t_2) = |\{s \mid \varphi(s, t_1, t_2)\}|$ counts the number of even numbers s between t_1 and t_2 .

We will also use quasi-polynomials. A *quasi-polynomial* is a function $g: \mathbb{N}^k \rightarrow \mathbb{Q}$ such that there is a k -dimensional lattice $\Lambda \subseteq \mathbb{Z}^k$ (that is, Λ is a finite index subgroup of \mathbb{Z}^k) and polynomials $q_{\lambda+\Lambda}(\bar{t})$ such that $g(\bar{t}) = q_{\lambda+\Lambda}(\bar{t})$ for all $\bar{t} \in \lambda + \Lambda$, where $\lambda + \Lambda$ belongs to the quotient set \mathbb{Z}^k/Λ . Notice that each coset $\lambda + \Lambda \in \mathbb{Z}^k/\Lambda$ is semilinear. A *piecewise quasi-polynomial* is a function $g: \mathbb{N}^k \rightarrow \mathbb{Q}$ such that there exist a finite partition $\bigcup_i (P_i \cap \mathbb{N}^k) = \mathbb{N}^k$ with rational polyhedra P_i and quasi-polynomials g_i such that $g(\bar{t}) = g_i(\bar{t})$ for all $\bar{t} \in P_i \cap \mathbb{N}^k$. Recall that a rational polyhedron is the finite intersection of half-spaces $\{(x_1, \dots, x_k) \in \mathbb{R}^k \mid \sum_{i=1}^k a_i x_i \leq b\}$ where the coefficients a_1, \dots, a_k and the right hand side b are integers. If $P \subseteq \mathbb{R}^k$ is a rational polyhedron then $P \cap \mathbb{N}^k$ is clearly effectively Presburger-definable and hence effectively semilinear.

We will need the following two theorems:

► **Theorem 6** ([22]). *For every Presburger formula $\varphi(\bar{s}, \bar{t})$ the function $c(\bar{t}) = |\{\bar{s} \mid \varphi(\bar{s}, \bar{t})\}|$ is piecewise quasi-polynomial. Furthermore, the representation of c can be effectively computed.*

For example the counting function $c(t_1, t_2)$ from above which counts the number of even numbers between t_1 and t_2 can be seen to be piecewise quasi-polynomial: Choose the polyhedron $P = \{(x_1, x_2) \in \mathbb{R}^2 \mid x_1 \leq x_2\}$ and the lattice $\Lambda = 2\mathbb{Z} \times 2\mathbb{Z}$. Then $c(t_1, t_2) = 0$ for all $(t_1, t_2) \in \mathbb{N}^2 \setminus P$ and

$$c(t_1, t_2) = \begin{cases} \frac{t_2 - t_1}{2} + 1, & \text{for } (t_1, t_2) \in P \cap \Lambda, \\ \frac{t_2 - t_1 + 1}{2}, & \text{for } (t_1, t_2) \in P \cap (((1, 0) + \Lambda) \cup ((0, 1) + \Lambda)), \\ \frac{t_2 - t_1}{2}, & \text{for } (t_1, t_2) \in P \cap ((1, 1) + \Lambda). \end{cases}$$

Since every semilinear set is a disjoint union of fundamental linear sets for every counting function c of a Presburger formula there exists a finite partition $\mathbb{N}^k = \bigcup_i L_i$ and polynomials g_i such that each L_i is a fundamental linear set and the counting function c coincides with g_i on L_i . Furthermore, this representation is effectively computable. Theorem 6 can be strengthened for the special case where the tuple \bar{s} is a single variable:

► **Theorem 7** ([20]). *For every Presburger formula $\varphi(y, \bar{z})$ there exists a formula $\psi(x, \bar{z})$ which states that x is the number of elements y such that $\varphi(y, \bar{z})$ holds.*

Multisets. A *multiset* over a set A is a function $M: A \rightarrow \mathbb{N}_\infty$ where $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$. The number $M(a)$ is the *multiplicity* of a in M . The support of M is the set $\text{supp}(M) = \{a \in A \mid M(a) > 0\}$. We call M *finite* if its support is finite and every multiplicity is finite. If $f: A \rightarrow B$ is a function and $X \subseteq A$, then we define $f(X)$ to be the multiset

over B with $f(X)(b) = |f^{-1}(\{b\}) \cap X|$. Instead of $f(A)$ we also write $\text{Rg}(f)$, which is the *range* of f . The union and difference of two multisets $M_1, M_2: A \rightarrow \mathbb{N}_\infty$ is defined by $(M_1 \uplus M_2)(a) = M_1(a) + M_2(a)$ and $(M_1 \setminus M_2)(a) = \max(M_1(a) - M_2(a), 0)$ where $n - \infty = 0$ for all $n \in \mathbb{N}_\infty$. We define $M_1 \subseteq M_2$ iff $M_1(a) \leq M_2(a)$ for all $a \in A$. Given a multiset M over A and a subset $S \subseteq A$, we define $M \upharpoonright S$ as $(M \upharpoonright S)(a) = M(a)$ if $a \in S$ and $(M \upharpoonright S)(a) = 0$ otherwise.

4 Characterization: Proof of Theorem 3

Our proof consists of several lemmas. To prove the implication (2) \rightarrow (1), we first observe that the class of automatic equivalence structures with polynomially bounded growth is closed under disjoint union.

► **Lemma 8.** *Let $\mathfrak{E}_1 = (D_1, E_1)$ and $\mathfrak{E}_2 = (D_2, E_2)$ be two automatic equivalence structures. Then there exists an automatic equivalence structure $\mathfrak{E} = (D, E)$ isomorphic to the disjoint union of \mathfrak{E}_1 and \mathfrak{E}_2 . If D_1 and D_2 have growth $O(n^k)$ then also D has growth $O(n^k)$.*

Proof. Say $D_1, D_2 \subseteq \Sigma^*$ and let $\#_1, \#_2 \notin \Sigma$ be fresh symbols. The disjoint union $\mathfrak{E}_1 \cup \mathfrak{E}_2$ is isomorphic to (D, E) where $D = \#_1 D_1 \cup \#_2 D_2$ and $E = \bigcup_{i \in \{1, 2\}} \{(\#_i u, \#_i v) \mid (u, v) \in E_i\}$. For $n \geq 1$ we have $|D \cap \Sigma^n| = |D_1 \cap \Sigma^{n-1}| + |D_2 \cap \Sigma^{n-1}| \leq O(n^k)$. ◀

To complete the proof of the implication (2) \rightarrow (1), it suffices to consider equivalence structures of the form $\mathfrak{E}(g)$ and equivalence structures where all classes are infinite. If \mathfrak{E} consists of $n \in \mathbb{N}$ infinite classes, then \mathfrak{E} is isomorphic to $(0^*, E)$ where two words 0^i and 0^j are equivalent iff i and j are congruent mod n . If \mathfrak{E} consists of infinitely many infinite classes then \mathfrak{E} is isomorphic to $(0^* 1^*, E)$ where two words are equivalent iff the number of 0's is equal.

► **Lemma 9.** *Given a non-zero polynomial $g \in \mathbb{N}[t_1, \dots, t_k]$ with degree d one can compute an automatic equivalence structure (D, E) isomorphic to $\mathfrak{E}(g)$ where the growth of D is $O(n^{k+d-1})$.*

Proof. Kuske, Lohrey, Liu [4] construct a finite automaton $\mathcal{A} = (Q, \Sigma, I, \Delta, F)$ over the alphabet $\Sigma = \{1, \dots, k\}$ such that the number of accepting runs of \mathcal{A} on $1^{t_1} \dots k^{t_k}$ is $g(t_1, \dots, t_k)$ for all $t_1, \dots, t_k \in \mathbb{N}$. A run in \mathcal{A} can be described as a sequence of transitions

$$(q_0, a_1, q_1)(q_1, a_2, q_2) \cdots (q_{n-1}, a_n, q_n) \in \Delta^*.$$

Let $D \subseteq \Delta^*$ be the set of all accepting runs of \mathcal{A} , which is a regular language, and let two runs be E -equivalent iff they are runs on the same word. Notice that E is automatic and (D, E) is isomorphic to $\mathfrak{E}(g)$. The number of accepting runs of \mathcal{A} on words of length $n \in \mathbb{N}$ is bounded by

$$\sum_{t_1 + \dots + t_k = n} g(t_1, \dots, t_k) \leq O(n^{k-1}) \cdot g(n, \dots, n) \leq O(n^{k+d-1}),$$

which concludes the proof. ◀

With respect to Lemma 8, note that the class EqP is closed under the product operation (although this fact is not used in our arguments). Namely, let $\mathfrak{E}_1 = (D_1, E_1)$ and $\mathfrak{E}_2 = (D_2, E_2)$ be two structures from EqP . Then the equivalence structure $\mathfrak{E}_1 \cdot \mathfrak{E}_2 = (D_1 \times D_2; E_1 \cdot E_2)$, where $((x, y), (x', y')) \in E_1 \cdot E_2$ iff $(x, x') \in E_1$ and $(y, y') \in E_2$, belong to EqP .

In the rest of the section, we prove the implication (1) \rightarrow (2). We consider an automatic equivalence structure $\mathfrak{E} = (D, E)$ and show that it can be decomposed as stated in Theorem 3. We will start with some preprocessing. First one can define the set of elements in finite E -classes by the formula $\varphi_{fin}(x) = \neg\exists^\infty y Exy$. Hence we can assume that all classes are finite.

► **Lemma 10.** *If $D \subseteq 0^*$ then \mathfrak{E} contains only finitely many infinite classes.*

Proof. We call a set $C \subseteq 0^*$ *eventually d -periodic* ($d \in \mathbb{N}$) if there exists a number $t \in \mathbb{N}$ such that for all $i \geq t$ we have $0^i \in C$ iff $0^{i+d} \in C$. Let \mathcal{A} be a deterministic finite automaton (DFA) for $\otimes E$ with transition function δ . We claim that there are numbers $t \geq 0$ and $d \geq 1$ such that $\delta(q, (0, \diamond)^t) = \delta(q, (0, \diamond)^{t+d})$ for all states q in \mathcal{A} . Clearly, for every state q there are numbers $t_q \geq 0$ and $d_q \geq 1$ such that $\delta(q, (0, \diamond)^{t_q}) = \delta(q, (0, \diamond)^{t_q+d_q})$. Then it suffices to take the maximum over all t_q and the product of all d_q over all states q .

Let C be an equivalence class and let $0^i \in C$ be the shortest word. By the property above we have $(0^i, 0^j) \in E$ iff $(0^i, 0^{j+d}) \in E$ for all $j \geq i + t$, i.e. C is eventually d -periodic. Any $d + 1$ infinite eventually d -periodic sets cannot be pairwise disjoint, which proves the claim. ◀

If D has growth $O(n^k)$, then we can assume that $D \subseteq 0^* \cdots k^*$ as stated in the next lemma.

► **Lemma 11** ([1]). *If $\mathfrak{A} = (D, R_1, \dots, R_m)$ is an automatic structure where D has growth $O(n^k)$ then there exists an automatic structure $\mathfrak{A}' = (D', R'_1, \dots, R'_m)$ which is isomorphic to \mathfrak{A} and $D' \subseteq 0^* 1^* \cdots k^*$.*

In the following assume that $D \subseteq 0^* \cdots k^*$ and that every E -class is finite. Let $R \subseteq D$ be the set of minimal elements from the equivalence classes with respect to the length-lexicographical order. A standard pumping argument shows that there exists a constant $b \in \mathbb{N}$ such that the length difference between any two equivalent elements is bounded by b .

► **Lemma 12.** *There exists $b \in \mathbb{N}$ such that $(u, v) \in E$ implies $\|u\| - \|v\| \leq b$.*

Proof. Let b the number of states in an automaton \mathcal{A} for $\otimes E$. Assume that $\|v\| > \|u\| + b$ (the other case is symmetric). The word $u \otimes v$ is accepted by \mathcal{A} and has a suffix of the form $\diamond^{b+1} \otimes w$ for some suffix w of v . In this suffix \mathcal{A} visits some state twice, and hence a nonempty infix of $\diamond^{b+1} \otimes w$ can be pumped, yielding infinitely many equivalent elements to u . This contradicts the assumption that all classes are finite. ◀

► **Lemma 13.** *There is a Presburger formula $\varphi(t_0, \dots, t_k, s_0, \dots, s_k)$ stating that $r = 0^{t_0} \cdots k^{t_k} \in R$, $v = 0^{s_0} \cdots k^{s_k} \in D$ and $(r, v) \in E$.*

Proof. Since $E \cap R \times D$ is an automatic relation the set $L = \otimes(E \cap R \times D)$ is by definition a regular language over the alphabet $\Gamma = \{0, \dots, k, \diamond\}^2$. Notice that the restriction of the Parikh mapping Φ to L is injective since the letters in words of L are naturally ordered. By Parikh's theorem $\Phi(L)$ is effectively semilinear and hence effectively definable by Presburger formula. This allows to construct a formula φ stating that there exists a vector $x \in \Phi(L)$ indexed by pairs in Γ such that

- $\sum_{j \in \{0, \dots, k, \diamond\}} x_{(i, j)} = t_i$ for all $0 \leq i \leq k$
- $\sum_{i \in \{0, \dots, k, \diamond\}} x_{(i, j)} = s_j$ for all $0 \leq j \leq k$.

This concludes the proof. ◀

21:8 Automatic Equivalence Structures of Polynomial Growth

Now we are ready to finish the proof of Theorem 3. Let φ be the formula from Lemma 13. By Theorem 6 the counting function $c(\bar{t}) = |\{\bar{s} \mid \varphi(\bar{t}, \bar{s})\}|$ is a piecewise quasi-polynomial function, and one can compute a representation of c . If $r = 0^{t_0} \dots k^{t_k} \in R$ then $c(t_0, \dots, t_k)$ is the size of the equivalence class of r ; otherwise $c(t_0, \dots, t_k) = 0$. By definition of $\mathfrak{E}(c)$ we have $\mathfrak{E}(c) \cong \mathfrak{E}$. It remains to decompose $\mathfrak{E}(c)$ into equivalence structures $\mathfrak{E}(h_i)$ defined by polynomials h_i and prove that $\deg(h_i) + \text{var}(h_i) \leq k + 1$.

We can assume that c is presented by a finite partition $\mathbb{N}^{k+1} = \bigcup_i L_i$ and polynomials g_i such that each L_i is a fundamental linear set and c coincides with g_i on L_i [6]. Let h_i be the function obtained from g_i by substituting the linear representation of vectors in L_i into g_i . More formally, let $L_i = \bar{v}_0 + \langle \bar{v}_1, \dots, \bar{v}_\ell \rangle$ where the period vectors are linearly independent and let $\alpha_i: \mathbb{N}^\ell \rightarrow \mathbb{N}^{k+1}$ be defined by $\alpha_i(\lambda_1, \dots, \lambda_\ell) = \bar{v}_0 + \sum_{j=1}^\ell \lambda_j \bar{v}_j$. Then $g_i \circ \alpha_i$ is a polynomial and \mathfrak{E} is isomorphic to the disjoint union $\bigcup_i \mathfrak{E}(g_i \circ \alpha_i)$.

Now fix i and let $h_i = g_i \circ \alpha_i$, which is a polynomial in the variables $\lambda_1, \dots, \lambda_\ell$. It remains to show that $\deg(h_i) + \ell \leq k + 1$. Let $R_i = R \cap \{0^{t_0} \dots k^{t_k} \mid \bar{t} \in L_i\}$ and $D_i = \{v \in D \mid \exists r \in R_i : (v, r) \in E\}$. Then $\mathfrak{E}(h_i)$ is isomorphic to the restriction of \mathfrak{E} to D_i . The representatives in R_i of length n are

$$R_{i,n} = \{0^{t_0} \dots k^{t_k} \mid \exists \bar{\lambda} \in \mathbb{N}^\ell : \alpha_i(\bar{\lambda}) = \bar{t}, \sum_j t_j = n\}.$$

Each $r \in R_{i,n}$ is only equivalent to words of length at least n , since r is length-lexicographically minimal in its class, and at most $n + b$, by Lemma 12. Since b is a constant we know that $|\{v \in D_i \mid n \leq |v| \leq n + b\}| = O(n^k)$ and hence

$$\sum_{r \in R_{i,n}} |[r]| = \left| \bigcup_{r \in R_{i,n}} [r] \right| = O(n^k). \quad (2)$$

For a tuple $\bar{\lambda} = (\lambda_1, \dots, \lambda_\ell)$ let $\text{len}(\bar{\lambda})$ be the sum of all entries in $\alpha_i(\bar{\lambda})$, which is an affine function in $\bar{\lambda}$, namely $\text{len}(\lambda_1, \dots, \lambda_\ell) = a_0 + \sum_{j=1}^\ell a_j \lambda_j$ where $a_j \in \mathbb{N}$ is the sum of all entries in \bar{v}_j . Since α_i is injective, none of the vectors \bar{v}_j can be the zero vector and therefore we must have $a_1, \dots, a_\ell \geq 1$. We obtain

$$\begin{aligned} \sum_{\text{len}(\lambda_1, \dots, \lambda_\ell) = n} h_i(\lambda_1, \dots, \lambda_\ell) &= \sum_{\text{len}(\lambda_1, \dots, \lambda_\ell) = n} g_i(\alpha_i(\lambda_1, \dots, \lambda_\ell)) \\ &= \sum_{0^{t_0} \dots k^{t_k} \in R_{i,n}} g_i(t_0, \dots, t_k) = \sum_{r \in R_{i,n}} c(r) \stackrel{(2)}{=} O(n^k). \end{aligned}$$

Let a be the least common multiple of a_1, \dots, a_ℓ and assume that $n = a_0 + a \cdot m$ for some $m \in \mathbb{N}$. We restrict the left handside to those tuples $(\lambda_1, \dots, \lambda_\ell)$ where each $a_j \lambda_j$ is divisible by a , i.e. $a_j \cdot \lambda_j = a \cdot \mu_j$ for some μ_j . We get

$$\sum_{\mu_1 + \dots + \mu_\ell = m} h_i(a\mu_1, \dots, a\mu_\ell) = O(n^k).$$

The number of tuples $(\mu_1, \dots, \mu_\ell) \in \mathbb{N}^\ell$ with $m/(\ell - 1) \leq \mu_j$ for all j and $\mu_1 + \dots + \mu_\ell = m$ is $\Omega(m^{\ell-1}) = \Omega(n^{\ell-1})$ because in the coordinates 1 to $\ell - 1$ we can pick any integer in the interval $[m/(\ell - 1), m/\ell]$ and pick $\mu_\ell \geq m/\ell$ such that the sum equals n . This implies $\Omega(n^{\ell-1}) \cdot h_i(am', \dots, am') \leq O(n^k)$ where $m' = m/(\ell - 1)$. Since $m' = \Theta(n)$ the degree of h_i must satisfy $\ell - 1 + \deg(h_i) \leq k$.

5 Undecidability: Proof of Theorem 4

Using Theorem 3 we can state an equivalent formulation of the isomorphism problem for automatic equivalence structures with growth $O(n^k)$. For this we define two sets. The first set is the set of polynomials f such that the number of variables in f plus the degree of f is not greater than $k + 1$:

$$\mathcal{P}_k = \{f \in \mathbb{N}[x_1, \dots, x_\ell] \mid 0 \leq \ell \leq k + 1, \text{var}(f) + \deg(f) \leq k + 1\}.$$

The second set defines a collection of multi-sets determined by tuples of polynomials from \mathcal{P}_k . Formally:

$$\mathcal{M}_k = \left\{ \biguplus_{i=1}^m \text{Rg}(f_i) \mid f_1, \dots, f_m \in \mathcal{P}_k, m \in \mathbb{N} \right\}.$$

► **Definition 14.** Let \mathcal{P} be a set of polynomials. A \mathcal{P} -representation for a multiset M over \mathbb{N} is a list of polynomials $(f_1, \dots, f_m) \in \mathcal{P}^m$ such that $M = \biguplus_{i=1}^m \text{Rg}(f_i)$.

For example, the list (x, x^2) is a representation of the multiset $\{0, 0, 1, 1, 2, 3, 4, 4, 5, 6, \dots\}$. The decision problem \mathcal{P} -MULTISET-EQ asks whether two given \mathcal{P} -representations define the same multiset.

► **Lemma 15.** For each constant $k \geq 0$, the isomorphism problem for automatic equivalence structures of growth $O(n^k)$ is equivalent to \mathcal{P}_k -MULTISET-EQ.

Proof. The equivalence follows basically from Theorem 3. However, we need to pay attention to infinite equivalence classes and multisets containing 0.

First we observe that the isomorphism problem for automatic equivalence structures of growth $O(n^k)$ is equivalent to the question whether $F \upharpoonright \mathbb{N}_+ = G \upharpoonright \mathbb{N}_+$ for two given multisets $F, G \in \mathcal{M}_k$, i.e. we exclude 0 from the multisets. Let us call this decision problem \mathcal{P}_k -POS-MULTISET-EQ. To solve the isomorphism problem we first compute for the given equivalence structures representative sets for the set of infinite equivalence classes and compare their cardinality. If they are unequal, we reject. Otherwise we restrict the equivalence structures to those elements which are contained in finite classes. By Theorem 3 we can compute representations $\bigcup_i \mathfrak{E}(f_i)$ and $\bigcup_i \mathfrak{E}(g_i)$ for the restricted equivalence structures where $f_i, g_i \in \mathcal{P}_k$. Then the equivalence structures are isomorphic if and only if $(\biguplus_i \text{Rg}(f_i)) \upharpoonright \mathbb{N}_+ = (\biguplus_i \text{Rg}(g_i)) \upharpoonright \mathbb{N}_+$. Conversely, given two \mathcal{M}_k -multisets $F = \biguplus_i \text{Rg}(f_i)$ and $G = \biguplus_i \text{Rg}(g_i)$, we have $F \upharpoonright \mathbb{N}_+ = G \upharpoonright \mathbb{N}_+$ if and only if $\bigcup_i \mathfrak{E}(f_i)$ and $\bigcup_i \mathfrak{E}(g_i)$ are isomorphic. By Theorem 3 we can compute two automatic structures equivalent to $\bigcup_i \mathfrak{E}(f_i)$ and $\bigcup_i \mathfrak{E}(g_i)$, respectively.

It remains to prove the equivalence of \mathcal{P}_k -POS-MULTISET-EQ and \mathcal{P}_k -MULTISET-EQ. Since $\text{Rg}(x_1)$ is the multiset containing 0 infinitely often, we have $(\biguplus_i \text{Rg}(f_i)) \upharpoonright \mathbb{N}_+ = (\biguplus_i \text{Rg}(g_i)) \upharpoonright \mathbb{N}_+$ if and only if $\biguplus_i \text{Rg}(f_i) \cup \text{Rg}(x_1) = \biguplus_i \text{Rg}(g_i) \cup \text{Rg}(x_1)$. This yields a reduction from \mathcal{P}_k -POS-MULTISET-EQ to \mathcal{P}_k -MULTISET-EQ. For the other direction, suppose we are given two multisets $F = \biguplus_i \text{Rg}(f_i)$ and $G = \biguplus_i \text{Rg}(g_i)$ from \mathcal{M}_k . Then $F = G$ if and only if $F(0) = G(0)$ and $F \upharpoonright \mathbb{N}_+ = G \upharpoonright \mathbb{N}_+$. The latter is equivalent to the \mathcal{P}_k -MULTISET-EQ-instance $\bigcup_i \mathfrak{E}(f_i) = \bigcup_i \mathfrak{E}(g_i)$. To test $F(0) = G(0)$ it suffices to show how to compute $\text{Rg}(g)(0)$ for a given polynomial $g \in \mathbb{N}[x_1, \dots, x_\ell]$. First notice that $\text{Rg}(g)(0)$ is the number of solutions $\bar{u} \in \mathbb{N}^\ell$ for $g(\bar{u}) = 0$. If $\bar{u}, \bar{v} \in \mathbb{N}^\ell$ are tuples with the same non-zero coordinates then $g(\bar{u}) = 0$ if and only if $g(\bar{v}) = 0$. Hence $g(\bar{u}) = 0$ either has zero, one, or infinitely many solutions, and it suffices to search for solutions in $\bar{u} \in \{0, 1\}^\ell$. ◀

21:10 Automatic Equivalence Structures of Polynomial Growth

We also consider the related problem over sets. Let us write $\text{Img}(f)$ for the image of a polynomial $f \in \mathbb{N}[x_1, \dots, x_k]$, i.e. the set $\{f(\bar{x}) \mid \bar{x} \in \mathbb{N}^k\}$. If \mathcal{P} is a set of polynomials, a \mathcal{P} -representation for a set $M \subseteq \mathbb{N}$ is a list of polynomials $(f_1, \dots, f_m) \in \mathcal{P}^m$ such that $M = \bigcup_{i=1}^m \text{Img}(f_i)$. The decision problem \mathcal{P} -SET-EQ asks whether two given \mathcal{P} -representations define the same set.

► **Lemma 16.** *If $k \in \mathbb{N}$, then \mathcal{P}_k -SET-EQ is reducible to \mathcal{P}_{k+1} -MULTISET-EQ.*

Proof. Let $(f_1, \dots, f_m, g_1, \dots, g_n)$ be an instance for \mathcal{P}_k -SET-EQ. If $f_i: \mathbb{N}^k \rightarrow \mathbb{N}$ then let $f'_i: \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ be the polynomial defined by $f'_i(\bar{x}, y) = f_i(\bar{x})$ for all $\bar{x} \in \mathbb{N}^k, y \in \mathbb{N}$, and similarly g'_i . Since every element has either multiplicity 0 or ∞ in $\text{Rg}(f'_i)$ and $\text{Rg}(g'_i)$ we have

$$\bigcup_{i=1}^m \text{Img}(f_i) = \bigcup_{i=1}^n \text{Img}(g_i) \iff \biguplus_{i=1}^m \text{Rg}(f'_i) = \biguplus_{i=1}^n \text{Rg}(g'_i).$$

The polynomials f'_i, g'_i have one more variable and hence belong to \mathcal{P}_{k+1} . ◀

Proof of Theorem 4. We use the MRDP-theorem [14] stating that a set of natural numbers $X \subseteq \mathbb{N}$ is recursively enumerable if and only if it is *Diophantine*, i.e. there exists a polynomial $p(x, y_1, \dots, y_k) \in \mathbb{Z}[x, y_1, \dots, y_k]$ such that

$$X = \{a \in \mathbb{N} \mid \exists y_1, \dots, y_k \in \mathbb{N} : p(a, y_1, \dots, y_k) = 0\}.$$

Let $X \subseteq \mathbb{N}$ be a Σ_1^0 -complete set and $p \in \mathbb{Z}[x, x_1, \dots, x_k]$ be a polynomial as above defining X .¹ By splitting p into its monomials with positive and negative coefficients we obtain polynomials $p_1, p_2 \in \mathbb{N}[x, x_1, \dots, x_k]$ such that

$$a \in X \iff \exists y_1, \dots, y_k \in \mathbb{N} : p_1(a, y_1, \dots, y_k) = p_2(a, y_1, \dots, y_k). \quad (3)$$

If we define $N = \{(x, y) \mid x \neq y \in \mathbb{N}\}$, then $a \in X$ is also equivalent to

$$\{(p_1(a, \bar{y}), p_2(a, \bar{y})) \mid \bar{y} \in \mathbb{N}^k\} \not\subseteq N. \quad (4)$$

Using the injective pairing function $C(x, y) = (x + y)^2 + 3x + y$ we can alternatively state this by

$$\text{Img}(C(p_1(a, \bar{y}), p_2(a, \bar{y}))) \not\subseteq \text{Img}(C(y, x + y + 1)) \cup \text{Img}(C(x + y + 1, y)).$$

Since $A \not\subseteq B$ iff $A \neq A \cup B$ we obtain a reduction from X to the complement of \mathcal{P}_m -SET-EQ where m is bounded in a function of $\text{var}(p)$ and $\text{deg}(p)$. Hence \mathcal{P}_m -SET-EQ is Π_1^0 -hard. Therefore also \mathcal{P}_{m+1} -MULTISET-EQ and the isomorphism problem over automatic equivalence structures of growth $O(n^{m+1})$ is Π_1^0 -hard. ◀

6 Decidability: Proof of Theorem 5

Now we prove Theorem 5 by proving:

► **Theorem 17.** *The problem \mathcal{P}_2 -MULTISET-EQ is decidable.*

To prove Theorem 17 we proceed in three steps. First we reduce it to the case that the multisets have only finite multiplicities. In the second step we test equality of the multisets on their “unbounded linear part” and reduce the problem to testing equality of unions of degree-two polynomial ranges. In the third step we provide a decision procedure for the latter problem.

¹ It is known that p can be chosen to have degree at most four [14, Section 1.2].

6.1 Closure properties

► **Lemma 18.** *If $f \in \mathbb{N}[x_1, \dots, x_k]$ has degree d and $T \subseteq \mathbb{N}^k$ is semilinear, then $f(T)$ is a finite union of ranges $\text{Rg}(g_i)$ where $\text{var}(g_i) \leq k$ and $\deg(g_i) = d$. The polynomials g_i can be computed effectively. In particular, $f(T)$ belongs effectively to \mathcal{M}_{d+k-1} .*

Proof. Let $T = \bigcup_i T_i$ be a representation of T as a disjoint union of fundamental linear sets T_i . Since $f(T) = \biguplus_i f(T_i)$ we can assume that T is a fundamental linear set, say $T = \bar{v}_0 + \langle \bar{v}_1, \dots, \bar{v}_m \rangle$ where the period vectors are linearly independent; in particular, we have $m \leq k$. Consider the polynomial $g \in \mathbb{N}[\lambda_1, \dots, \lambda_m]$ defined by

$$g(\lambda_1, \dots, \lambda_m) = f(\bar{v}_0 + \sum_{j=1}^m \lambda_j \bar{v}_j),$$

which satisfies $f(T) = \text{Rg}(g)$ and $\deg(g) = \deg(f) = d$. ◀

► **Lemma 19.** *If $F \in \mathcal{M}_2$ and $S \subseteq \mathbb{N}$ is semilinear, then $F \upharpoonright S$ belongs effectively to \mathcal{M}_2 .*

Proof. Let $F = \biguplus_{i=1}^m \text{Rg}(f_i)$ with $f_1, \dots, f_m \in \mathcal{P}_2$. Since $F \upharpoonright S = \biguplus_{i=1}^m (\text{Rg}(f_i) \upharpoonright S)$ we can assume that $F = \text{Rg}(f)$ for some $f \in \mathcal{P}_2$. First assume that $\deg(f) \leq 1$. Since S is semilinear and f is an affine function, the set $L = \{\bar{t} \mid f(\bar{t}) \in S\}$ is effectively semilinear. By Lemma 18 we know that $\text{Rg}(f) \upharpoonright S = f(L)$ belongs to \mathcal{M}_2 . Now assume that $\deg(f) = 2$, i.e. $f(t) = at^2 + bt + c$ for some $a \neq 0, b, c \in \mathbb{N}$. Since f is injective, the multiset $F = \text{Rg}(f)$ is a set, and therefore $F \upharpoonright S = F \cap S$. Consider a representation of S as a finite disjoint union $S = \bigcup_i S_i$ of singleton sets and arithmetic progressions. Since $F \cap S = \biguplus_i (F \cap S_i)$ we can assume that S itself is either a singleton or an arithmetic progression. If $S = \{s\}$ then $\text{Rg}(f) \cap S$ is either empty or $\{s\}$, which can be decided. Assume $S = \{e + dn \mid n \in \mathbb{N}\}$ for some $e \in \mathbb{N}$ and $d \geq 1$. It is enough to prove that $T = \{t \in \mathbb{N} \mid \exists n \in \mathbb{N} : at^2 + bt + c = e + dn\}$ is effectively semilinear, since then, $\text{Rg}(f) \cap S = f(T)$ belongs to \mathcal{M}_2 by Lemma 18.

Notice that $t \in T$ if and only if $at^2 + bt + c$ is congruent to $e \pmod{d}$ and $at^2 + bt + c \geq e$. Define the function $h: \mathbb{Z}_d \rightarrow \mathbb{Z}_d$ with $h(t) = at^2 + bt + c$. We obtain a semilinear representation for $\{t \in \mathbb{N} \mid f(t) \equiv e \pmod{d}\}$ from $h^{-1}(e + \mathbb{Z}_d)$. Finally, we intersect this set with the interval $[t_0, \infty)$ where t_0 is the smallest number with $at_0^2 + bt_0 + c \geq e$ to obtain T . ◀

6.2 Reduction to multisets with finite multiplicities

Let $\mathcal{P}_{2,\text{fin}} \subseteq \mathcal{P}_2$ be the set of all polynomials of the form:

- $f = a$
- $f(t) = at^2 + bt + c$ where $a \neq 0$ or $b \neq 0$,
- $f(s, t) = as + bt + c$ where $a, b \neq 0$

Notice that $\text{Rg}(g)$ of a polynomial $g \in \mathcal{P}_2$ has finite multiplicities, i.e. $\text{Rg}(g)(a) < \infty$ for all $a \in \mathbb{N}$, if and only if $g \in \mathcal{P}_{2,\text{fin}}$. Let $\mathcal{M}_{2,\text{fin}}$ be the set of all multisets $\biguplus_{i=1}^m \text{Rg}(f_i)$ where $f_1, \dots, f_m \in \mathcal{P}_{2,\text{fin}}$. We will show that \mathcal{P}_2 -MULTISET-EQ is reducible to $\mathcal{P}_{2,\text{fin}}$ -MULTISET-EQ and start with a useful lemma.

► **Lemma 20.** *If $F = \text{Rg}(f)$ with $f \in \mathcal{P}_2$, then one can construct a Presburger formula $\varphi(x, y)$ stating that $F(x) = y < \infty$.*

Proof. Suppose f has two variables, say $f(s, t) = as + bt + c$. If $a = 0$, then F contains every number of the form $bt + c$ infinitely often, and does not contain any other number. The case $b = 0$ is similar. If both $a \neq 0$ and $b \neq 0$, then F has only finite multiplicities. Using Theorem 7 we can count for a given number x the number $|\{s \in \mathbb{N} \mid \exists t \in \mathbb{N} : as + bt + c = x\}|$.

21:12 Automatic Equivalence Structures of Polynomial Growth

Suppose f has one variable, say $f(t) = at^2 + bt + c$. If $a = b = 0$, then F contains only c infinitely often. Otherwise F contains each number of the form $at^2 + bt + c$ exactly once. ◀

► **Lemma 21.** \mathcal{P}_2 -MULTISET-EQ is reducible to $\mathcal{P}_{2,\text{fin}}$ -MULTISET-EQ.

Proof. Given two multisets $F, G \in \mathcal{M}_2$ and let $F_\infty = \{n \in \mathbb{N} \mid F(n) = \infty\}$ and $G_\infty = \{n \in \mathbb{N} \mid G(n) = \infty\}$. We have

$$F = G \iff F_\infty = G_\infty \text{ and } (F \upharpoonright \overline{F_\infty} = G \upharpoonright \overline{G_\infty})$$

where the complements are taken with respect to \mathbb{N} . Using Lemma 20 we can compute the semilinear sets F_∞ and G_∞ and test whether $F_\infty = G_\infty$. Using Lemma 19 we can compute \mathcal{P}_2 -representations for $F \upharpoonright \overline{F_\infty}$ and $G \upharpoonright \overline{G_\infty}$. ◀

6.3 Elimination of linear polynomials

Let $\mathcal{P}_{2,0} \subseteq \mathcal{P}_2$ be the set of all polynomials $f(t) = at^2 + bt + c$ where $a \neq 0$ and $b, c \in \mathbb{N}$ and polynomials $f = a$, and let $\mathcal{M}_{2,0}$ be the corresponding set of multisets.

► **Lemma 22.** $\mathcal{P}_{2,\text{fin}}$ -MULTISET-EQ is reducible to $\mathcal{P}_{2,0}$ -MULTISET-EQ.

Proof. Given two multisets $F, G \in \mathcal{M}_{2,\text{fin}}$ where $F = \uplus_i \text{Rg}(f_i)$ and $G = \uplus_i \text{Rg}(g_i)$. Let F_1 be the restriction of the union $\uplus_i \text{Rg}(f_i)$ to those polynomials f_i with $\deg(f_i) \leq 1$ and F_2 be the restriction to those polynomials of degree 2, and similarly G_1, G_2 for $\uplus_i \text{Rg}(g_i)$.

Since polynomials of degree 2 are injective, the maximum multiplicity in F_2 and G_2 is bounded by the total number, say k , of polynomials f_i and g_i , respectively. Hence, if $F = G$ then $|F_1(a) - G_1(a)| \leq k$ for all $a \in \mathbb{N}$. We can verify the latter property using the Presburger formulas $\varphi_{F_1}(x, y)$ and $\varphi_{G_1}(x, y)$ from Lemma 20, and return a negative instance if either one of the properties is violated (since $F \neq G$).

Now assume that the maximum multiplicity in $F_1 \setminus G_1$ and in $G_1 \setminus F_1$ is bounded by k . One can verify that $F = G$ if and only if

$$(F_1 \setminus G_1) \uplus F_2 = (G_1 \setminus F_1) \uplus G_2, \tag{5}$$

using the definition of difference between two multisets. If both $\text{supp}(F_1 \setminus G_1)$ and $\text{supp}(G_1 \setminus F_1)$ are finite, then also $F_1 \setminus G_1$ and $G_1 \setminus F_1$ are finite and we can return the instance (5). Otherwise we claim that $F \neq G$, and hence we return a negative instance. Towards a contradiction assume $F = G$ and that $\text{supp}(F_1 \setminus G_1)$ is infinite. The set $\text{supp}(F_1 \setminus G_1)$ is in fact effectively semilinear by Lemma 20 since

$$\text{supp}(F_1 \setminus G_1) = \{x \in \mathbb{N} \mid F_1(x) > G_1(x)\}.$$

Therefore the growth of $\text{supp}(F_1 \setminus G_1)$ is $\Omega(n)$ whereas the growth of $\text{supp}(G_2)$ is $O(\sqrt{n})$ because it is a finite union of ranges of quadratic polynomials and singletons. This contradicts the fact that $\text{supp}(F_1 \setminus G_1) \subseteq \text{supp}(G_2)$. ◀

6.4 Decision procedure for degree-two polynomials

In preparation for the decidability proof of $\mathcal{P}_{2,0}$ -MULTISET-EQ we show the following lemma concerning the solutions of quadratic Diophantine equations. The *growth function* of a subset $M \subseteq \mathbb{N}$ is the function $n \mapsto |M \cap [1, n]|$.

► **Lemma 23.** *Let $f, g \in \mathbb{N}[x]$ with $\deg(f) = \deg(g) = 2$. Let $S = \{(x, y) \in \mathbb{N}^2 \mid f(x) = g(y)\}$ and S_x be the projection to the first component. Then exactly one of the following cases holds:*

1. *the growth of S_x is $\Omega(n)$ and S is infinite and semilinear.*
2. *the growth of S_x is $o(n)$.*

It is decidable whether (1) or (2) holds. Moreover, if (1) holds then S can be effectively computed.

Proof. We follow the analysis of quadratic bivariate Diophantine equations from [19]. Consider the equation

$$ax^2 + cy^2 + dx + ey + f = 0 \quad (6)$$

where $a, c \neq 0$, $d \geq 0$, $e \leq 0$ and $f \in \mathbb{N}$. Define $D = -4ac \neq 0$, $E = -2ae$, $F = d^2 - 4af$ and $Y = 2ax + d$. Then (6) implies $DY^2 = (Dy + E)^2 + DF - E^2$. If $N = E^2 - DF$ and $X = Dy + E$ then we obtain the generalized Pell equation

$$X^2 - DY^2 = N. \quad (7)$$

Let L be the set of solutions $(X, Y) \in \mathbb{N}^2$ of (7) and let L_Y be the projection to the second component. Notice that the transformation $(x, y) \mapsto (X, Y) = (Dy + E, 2ax + d)$ is an injective function from S to L , and that if the growth of S_x is $\Omega(n)$ then also the growth of L_Y is $\Omega(n)$. Also notice that if $X^2 = DY^2 + N$ and $Y \geq 1$ then $X^2 \leq \max(D, |N|) \cdot Y^2$, hence X is linearly bounded in Y for all solutions $(X, Y) \in L$.

We will do a case distinction:

1. If $D < 0$ then any solution (X, Y) of (7) satisfies $X^2 + Y^2 \leq N$. Then L is finite, and hence also S is finite.
2. If $D > 0$ is a square number then L is finite, hence also S is finite.
3. If $D > 0$ and $N = 0$, then (7) is solvable if and only if D is a square number. In this case the solutions of (7) are $L = \{(\sqrt{DY}, Y) \mid Y \in \mathbb{N}\}$. Hence the solutions of (6) are of the form

$$S = \{(x, y) \in \mathbb{N}^2 \mid Dy + E = \sqrt{D}(2ax + d)\}.$$

From the equation we can compute a semilinear representation of S .

4. Now suppose that $D > 0$ is not a square number and $N \neq 0$. In this case we will show that L_Y , and therefore also S_x , has growth $o(n)$. Let $t, u \in \mathbb{N}$ be the smallest solution of the Pell equation $t^2 - Du^2 = 1$, the so called *fundamental solution*. We define an equivalence relation on \mathbb{Z}^2 where two pairs (X, Y) and (X', Y') are equivalent if $X + Y\sqrt{D} = (X' + Y'\sqrt{D})(t + u\sqrt{D})^m$ for some $m \in \mathbb{Z}$. It is known that the set of solutions of (7) over \mathbb{Z} is a finite union of equivalence classes, see [12, Theorem 8-8, 8-9]. Hence the number of solutions $(X, Y) \in L$ with $X + \sqrt{D}Y \leq n$ is bounded by $O(\log n)$. Since X is linearly bounded in Y for all solutions $(X, Y) \in L$, this implies that L_Y has growth $O(\log n)$, which is contained in $o(n)$.

This concludes the proof. ◀

► **Theorem 24.** $\mathcal{P}_{2,0}$ -MULTISET-EQ is decidable.

Proof. We will prove how to solve the following inclusion problem: Given polynomials $f, g_1, \dots, g_m \in \mathcal{P}_{2,0}$, test whether

$$\text{Rg}(f) \subseteq \bigoplus_{i=1}^m \text{Rg}(g_i) \quad (8)$$

21:14 Automatic Equivalence Structures of Polynomial Growth

holds and, if so, compute a $\mathcal{P}_{2,0}$ -representation for $[\biguplus_{i=1}^m \text{Rg}(g_i)] \setminus \text{Rg}(f)$. Then, given an instance $(f_1, \dots, f_m, g_1, \dots, g_n)$ of $\mathcal{P}_{2,0}$ -MULTISET-EQ, we can test $\biguplus_i \text{Rg}(f_i) \subseteq \biguplus_i \text{Rg}(g_i)$ as follows (the other inclusion is symmetric):

1. Initialize $G_0 = \biguplus_{i=1}^m \text{Rg}(g_i)$.
2. For all $1 \leq k \leq m$:
 - a. Test whether $\text{Rg}(f_k) \subseteq G_{k-1}$,
 - b. If so, compute $G_k = G_{k-1} \setminus \text{Rg}(f_k)$ otherwise return “no”.
3. Return “yes”.

It remains to show how to solve the defined inclusion problem. We assume that the polynomials g_i are sorted by $\text{var}(g_i)$, i.e. there exists some $0 \leq \ell \leq m$ such that $\text{var}(g_i) = 1$ for all $1 \leq i \leq \ell$ and $\text{var}(g_i) = 0$ for all $\ell + 1 \leq i \leq m$, i.e. $\biguplus_{i=\ell+1}^m \text{Rg}(g_i)$ is a finite multiset.

Case 1. If $f = a$, then we can test whether there exists some i such that $a \in \text{Rg}(g_i)$. If there is no such index, we reject. Otherwise pick such an index i . If $1 \leq i \leq \ell$ then we decompose $\text{Rg}(g_i) \setminus \{a\}$ into the finite set $\{g_i(0), \dots, g_i(x_0 - 1)\}$ and $\text{Rg}(g_i(x + x_0 + 1))$. If $\ell + 1 \leq i \leq m$ we can remove g_i from the list.

Case 2. If $f(x) = ax^2 + bx + c$ with $a \neq 0$ we test for each $1 \leq i \leq \ell$ whether the solution set

$$S_i = \{(x, y) \in \mathbb{N}^2 \mid f(x) = g_i(y)\}$$

is infinite and semilinear, and, if so, compute a semilinear representation for it using Lemma 23. Let $D_i = \{x \in \mathbb{N} \mid f(x) \in \text{Rg}(g_i)\}$ for all $1 \leq i \leq m$. Notice that (8) is equivalent to $\bigcup_{i=1}^m D_i = \mathbb{N}$. We rearrange the indices such that exactly the sets S_1, \dots, S_k are infinite and semilinear and hence by Lemma 23 the sets D_{k+1}, \dots, D_ℓ have growth $o(n)$. The sets $D_{\ell+1}, \dots, D_m$ have at most size 1. Define $X = \bigcup_{i=1}^k D_i$, which is effectively semilinear since each set D_i is the projection of S_i to the first component. We also define subsets $X_i \subseteq D_i$ for all $1 \leq i \leq k$ by

$$X_i = D_i \setminus \bigcup_{j=1}^{i-1} X_j,$$

which form a disjoint union $X_1 \cup \dots \cup X_k$ of X . Compute the semilinear sets $Y_i = \{y \in \mathbb{N} \mid \exists x \in X_i : (x, y) \in S_i\}$ for $1 \leq i \leq k$. Then we have $f(X_i) = g(Y_i)$ for all $1 \leq i \leq k$. We can rewrite (8) as

$$f(X) \uplus f(\mathbb{N} \setminus X) \subseteq \biguplus_{i=1}^k (g_i(Y_i) \uplus g_i(\mathbb{N} \setminus Y_i)) \uplus \biguplus_{i=k+1}^m \text{Rg}(g_i).$$

Since $f(X_i) = g(Y_i)$ and $f(\mathbb{N} \setminus X)$ is disjoint from all sets $g_i(\mathbb{N} \setminus Y_i)$, this is equivalent to

$$f(\mathbb{N} \setminus X) \subseteq \biguplus_{i=k+1}^m \text{Rg}(g_i) =: G.$$

We will do a case distinction.

Case 2a. If $\mathbb{N} \setminus X$ is finite, then we can test for each $x \in \mathbb{N} \setminus X$ whether $f(x)$ belongs to G and compute a representation for $G \setminus \{f(x)\}$, as above in case 1.

Case 2b. If $\mathbb{N} \setminus X$ is infinite we claim that $X \cup D_{k+1} \cup \dots \cup D_m \neq \mathbb{N}$ and hence (8) does not hold. Assume that $X \cup D_{k+1} \cup \dots \cup D_m = \mathbb{N}$ and therefore $\mathbb{N} \setminus X \subseteq D_{k+1} \cup \dots \cup D_m$. Since $\mathbb{N} \setminus X$ is infinite and semilinear, its growth must be $\Omega(n)$. However, all sets D_{k+1}, \dots, D_m have growth $o(n)$, contradiction.

Notice that we can distinguish cases 2a and 2b since X is effectively semilinear. \blacktriangleleft

7 Conclusion

We have characterized automatic equivalence structures over polynomially growing domains, and have investigated the decidability of the isomorphism problem. Since equivalence structures can be viewed as trees of height 2, as a next step one could study automatic trees over polynomially growing domains. Also it is still open whether the isomorphism problem over *unary* automatic structures is decidable (automatic structures whose domains are unary regular languages).

References

- 1 Vince Bárány. *Automatic presentations of infinite structures*. PhD thesis, RWTH Aachen University, Germany, 2007. URL: <http://darwin.bth.rwth-aachen.de/opus3/volltexte/2007/2019/>.
- 2 Achim Blumensath. *Automatic Structures*. Master's thesis, RWTH Aachen University, 1999.
- 3 Achim Blumensath and Erich Grädel. Finite Presentations of Infinite Structures: Automata and Interpretations. *Theory Comput. Syst.*, 37(6):641–674, 2004. doi:10.1007/s00224-004-1133-y.
- 4 Jiamou Liu Dietrich Kuske and Markus Lohrey. The isomorphism problem on classes of automatic structures with transitive relations. *Trans. Amer. Math. Soc.*, 365:5103–5151, 2013.
- 5 Ginsburg, Seymour and Spanier, Edwin. Semigroups, Presburger formulas, and languages. *Pacific journal of Mathematics*, 16(2):285–296, 1966.
- 6 Ryuichi Ito. Every Semilinear Set is a Finite Union of Disjoint Linear Sets. *J. Comput. Syst. Sci.*, 3(2):221–231, 1969. doi:10.1016/S0022-0000(69)80014-0.
- 7 Lukasz Kaiser, Sasha Rubin, and Vince Bárány. Cardinality and counting quantifiers on omega-automatic structures. In *STACS 2008, 25th Annual Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings*, pages 385–396, 2008. doi:10.4230/LIPIcs.STACS.2008.1360.
- 8 Bakhadyr Khoussainov and Anil Nerode. Automatic Presentations of Structures. In *Logical and Computational Complexity. Selected Papers. Logic and Computational Complexity, International Workshop LCC '94, Indianapolis, Indiana, USA, 13-16 October 1994*, pages 367–392, 1994. doi:10.1007/3-540-60178-3_93.
- 9 Bakhadyr Khoussainov, André Nies, Sasha Rubin, and Frank Stephan. Automatic Structures: Richness and Limitations. *Logical Methods in Computer Science*, 3(2), 2007. doi:10.2168/LMCS-3(2:2)2007.
- 10 Bakhadyr Khoussainov, Sasha Rubin, and Frank Stephan. Automatic linear orders and trees. *ACM Trans. Comput. Log.*, 6(4):675–700, 2005. doi:10.1145/1094622.1094625.
- 11 Dietrich Kuske and Markus Lohrey. First-order and counting theories of omega-automatic structures. *J. Symb. Log.*, 73(1):129–150, 2008. doi:10.2178/jsl/1208358745.
- 12 W.J. LeVeque. *Topics in Number Theory*. Number Bd. 1 in Dover Books on Mathematics. Dover Publications, 2002. URL: <https://books.google.de/books?id=ocAySqvLEEC>.
- 13 Jiamou Liu and Mia Minnes. Deciding the isomorphism problem in classes of unary automatic structures. *Theor. Comput. Sci.*, 412(18):1705–1717, 2011. doi:10.1016/j.tcs.2010.12.045.
- 14 Yuri V Matiyasevich. *Hilbert's tenth problem*, volume 105. MIT press Cambridge, 1993.
- 15 André Nies. Describing Groups. *Bulletin of Symbolic Logic*, 13(3):305–339, 2007. URL: <http://www.math.ucla.edu/~Eas1/bs1/1303/1303-001.ps>, doi:10.2178/bs1/1186666149.
- 16 G. Oliver and R. Thomas. Automatic presentations of finitely generated groups. In *STACS '05: Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science*. Springer-Verlag, 2005.
- 17 Rohit Parikh. On Context-Free Languages. *Journal of the ACM*, 13(4):570–581, 1966.
- 18 Sasha Rubin. *Automatic Structures*. PhD thesis, University of Auckland, 2004.

- 19 Reginald E. Sawilla, Alan K. Silvester, and Hugh C. Williams. A New Look at an Old Equation. In Alfred J. van der Poorten and Andreas Stein, editors, *Algorithmic Number Theory, 8th International Symposium, ANTS-VIII, Banff, Canada, May 17-22, 2008, Proceedings*, volume 5011 of *Lecture Notes in Computer Science*, pages 37–59. Springer, 2008. doi:10.1007/978-3-540-79456-1_2.
- 20 Nicole Schweikardt. Arithmetic, first-order logic, and counting quantifiers. *ACM Trans. Comput. Log.*, 6(3):634–671, 2005. doi:10.1145/1071596.1071602.
- 21 Andrew Szilard, Sheng Yu, Kaizhong Zhang, and Jeffrey Shallit. Characterizing Regular Languages with Polynomial Densities. In Ivan M. Havel and Václav Koubek, editors, *Mathematical Foundations of Computer Science 1992, 17th International Symposium, MFCS'92, Prague, Czechoslovakia, August 24-28, 1992, Proceedings*, volume 629 of *Lecture Notes in Computer Science*, pages 494–503. Springer, 1992. doi:10.1007/3-540-55808-X_48.
- 22 Kevin Woods. Presburger Arithmetic, Rational Generating Functions, and quasi-polynomials. *J. Symb. Log.*, 80(2):433–449, 2015. doi:10.1017/jsl.2015.4.

Guarded Teams: The Horizontally Guarded Case

Erich Grädel

RWTH Aachen University, Germany
graedel@logic.rwth-aachen.de

Martin Otto

Technische Universität Darmstadt, Germany
otto@mathematik.tu-darmstadt.de

Abstract

Team semantics admits reasoning about large sets of data, modelled by sets of assignments (called teams), with first-order syntax. This leads to high expressive power and complexity, particularly in the presence of atomic dependency properties for such data sets. It is therefore interesting to explore fragments and variants of logic with team semantics that permit model-theoretic tools and algorithmic methods to control this explosion in expressive power and complexity.

We combine here the study of team semantics with the notion of *guarded logics*, which are well-understood in the case of classical Tarski semantics, and known to strike a good balance between expressive power and algorithmic manageability. In fact there are two strains of guardedness for teams. *Horizontal guardedness* requires the individual assignments of the team to be guarded in the usual sense of guarded logics. *Vertical guardedness*, on the other hand, posits an additional (or definable) hypergraph structure on relational structures in order to interpret a constraint on the component-wise variability of assignments within teams.

In this paper we investigate the horizontally guarded case. We study horizontally guarded logics for teams and appropriate notions of *guarded team bisimulation*. In particular, we establish *characterisation theorems* that relate invariance under guarded team bisimulation with guarded team logics, but also with logics under classical Tarski semantics.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases Team semantics, guarded logics, bisimulation, characterisation theorems

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.22

Acknowledgements We gratefully acknowledge participation in the programme on *Logical Structure in Computation* at the Simons Institute in Berkeley in 2016, which provided interesting stimulation towards this work.

1 Introduction

Team semantics, which originates in the work of the model theorist Wilfrid Hodges [18], is based on the idea to evaluate logical formulae $\varphi(x_1, \dots, x_n)$ not for single assignments $s : \{x_1, \dots, x_n\} \rightarrow A$ from the free variables to elements of a structure \mathfrak{A} , but for *sets of such assignments*. These sets, which may have arbitrary size, are now called *teams*. The original motivation for team semantics has been to provide a compositional, model-theoretic semantics of the independence-friendly logic (IF-logic) [22], for which one previously only knew semantics based on either Skolem functions or on games of imperfect information. Team semantics has then become important as the mathematical basis of the modern logics of dependence and independence, which go back to the fundamental idea of Väänänen [25] to treat dependencies not as annotations of quantifiers (as in IF-logic), but as atomic properties of teams. Logics with team semantics for reasoning about dependence, independence, and imperfect information have meanwhile been established as a lively interdisciplinary research area, involving not just first-order logics, but also logics on the propositional and modal level, see e.g. [1].



© Erich Grädel and Martin Otto;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 22; pp. 22:1–22:17

Leibniz International Proceedings in Informatics



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

Team semantics admits reasoning about large sets of data, modelled by second-order objects such as sets of assignments, with a first-order syntax that does not explicitly refer to higher-order variables. In the presence of appropriate atomic team properties, such as dependence, inclusion and exclusion, or independence properties, team semantics can boost the expressiveness of first-order formalisms to the full power of existential second-order logic (denoted Σ_1^1) or, in the presence of further propositional operators such as different variants of implication or negation, even to full second-order logic (SO). Beyond logics for dependence and independence, team semantics may have broader applications. The idea of second-order reasoning with first-order syntax appears also in certain logics used for program verification (to reason about sets of variable assignments such as heaps, as in separation logic) or in quantum information theory (to reason about superpositions of basic states). Currently there are emerging new research directions that relate team semantics to such areas.

However, the ability of logics with team semantics to reason about second-order objects increases not only the expressive power, but also the complexity, and makes it much more difficult to understand the model theory of such formalisms and to handle them algorithmically. It is therefore relevant to explore fragments or variants of logics with team semantics that permit model-theoretic tools and algorithmic methods to control this explosion in expressive power and complexity. In this paper we explore a promising idea in this direction, namely the use of *guarded teams* and *guarded logics*. Guarded logics have been thoroughly investigated in the context of classical logical formalisms, and guarded fragments of first-order logic, fixed-point logics, and second-order logic have turned out to have very interesting and convenient model-theoretic and algorithmic properties, see e.g. [3, 4, 5, 12, 13, 14, 17, 19, 24] and our survey [15].

The basic guarded logic is the guarded fragment (GF) of first-order logic, introduced by Andr eka, van Benthem and N emeti [2]. It is defined by restricting existential and universal quantification in such a way that formulae only refer to *guarded tuples*, i.e., tuples of elements that occur together in some atomic fact. Syntactically, this means that quantifiers are used only in the form $\exists \bar{y}(\alpha \wedge \varphi)$ or $\forall \bar{y}(\alpha \rightarrow \varphi)$ where α is an atomic formula that must contain *all* free variables of φ (and possibly more). An important motivation for introducing the guarded fragment has been to explain and generalise the good algorithmic and model-theoretic properties of *modal logics* (see [6, 11] for background on modal logic). Recall that modal logic can be viewed as a fragment of first-order logic, via a standard translation that uses only two variables and a restricted kind of guarded quantification. The guarded fragment generalises the modal fragment enormously, dropping all restrictions (such as to use only two variables and only monadic and binary predicates), except the restriction that quantification must be guarded. It has turned out that almost all important algorithmic and model-theoretic properties of modal logic do indeed extend to the guarded fragment. In particular, the satisfiability problem for GF is decidable [2], GF has the finite model property, i.e., every satisfiable formula in the guarded fragment has a finite model [12], and moreover, GF has a generalised variant of the tree model property to the effect that every satisfiable formula has a model that admits a tree decomposition into guarded substructures, which in particular implies a bound on its tree width [12]. The tree model property paves the way to automata based algorithmic procedures for guarded logics. Further, GF admits efficient evaluation algorithms via model checking games of moderate size. There are similar results that hold for more powerful guarded logics, which are obtained either by a more liberal interpretation of guardedness (as in loosely guarded or clique guarded logics), by guarding negation instead of quantifiers, and/or by moving to guarded variants of stronger logics such as fixed-point or second-order logic.

The crucial model-theoretic tool to investigate guarded logics is the notion of a *guarded bisimulation* between two structures \mathfrak{A} and \mathfrak{B} , which we view here as a set Z of pairs of guarded assignments (s, t) such that $s \mapsto t$ induces a local isomorphism, and Z satisfies appropriate back-and-forth properties (cf. Section 4). Results of fundamental importance for guarded logics are *characterisation theorems* such as the one due to Andr eka, van Benthem and N emeti [2], saying that *a first-order formula is invariant under guarded bisimulation if, and only if, it is equivalent to a formula of the guarded fragment*, in short $\text{FO}/\sim_g \equiv \text{GF}$. There are several variants and extensions of this result, among them the finite model theory variant [24] (in which both the invariance statements, and the equivalence to a guarded formula are restricted to hold only over finite structures), as well as characterisation results for stronger guarded logics such as the one for fixed-point logic from [14]. See again [15] for a detailed discussion of guarded bisimulations in various contexts.

In this paper we aim at the development of the theory of *guarded teams* and *guarded logics with team semantics*. There are in fact two completely different variants of guarded teams. *Horizontal guardedness* requires all assignments in a team to be guarded in the usual sense. On this basis, we define horizontally guarded team semantics and horizontally guarded logics and relate them to the established classical framework of guarded logics. In particular, the good algorithmic properties of guarded fragments of first-order logic such as the decidability of the satisfiability problem, are easily seen to carry over to corresponding problems for horizontally guarded team semantics, such as the question whether a given guarded first-order formula is satisfiable by some nonempty team. However, corresponding questions for stronger guarded logics, involving atomic dependencies, are open.

To investigate the power of guarded team semantics we introduce and study two different notions of *guarded team bisimulation*, a weaker and a stronger one, and prove *characterisation theorems*, which relate formulae that are invariant under guarded team bisimulation to guarded team logics, but also to appropriate variants of logics with classical Tarski semantics. These are the core results of this paper. We remark that a loosely related characterisation theorem has been established in [20] for the much weaker context of *modal team semantics*.

Besides horizontal guardedness, there is also the rather different notion of *vertical guardedness* of a team, based on an additional (or definable) hypergraph structure on relational structures in order to interpret a constraint on the component-wise variability of the assignments in teams. When a team X is viewed as a table whose rows are the assignments of the team, then vertical guardedness imposes restraints to the effect that the columns, i.e. the value sets $X(x)$ are guarded. It turns out that this adds new and interesting second-order features to the team semantics of the resulting logics. However, due to space limitations we defer the development of this aspect to a future paper.

2 Team semantics

For a tuple $\bar{a} = (a_1, \dots, a_n) \in A^n$ we denote by $[\bar{a}]$ the set $\{a_1, \dots, a_n\}$ of its components. We write $\mathcal{P}(A)$ for the power set of A and set $\mathcal{P}^+(A) := \mathcal{P}(A) \setminus \{\emptyset\}$. An *assignment* to variables $x \in D$ into a set $A \neq \emptyset$ is a map $s : D \rightarrow A$. We write $s[x \mapsto a]$ for the assignment that extends, or updates, s by mapping x to a . We extend s in the obvious manner to tuples over D , and write $s(\bar{x})$ for $(s(x_1), \dots, s(x_k)) \in A^k$ if $\bar{x} = (x_1, \dots, x_k) \in D^k$, and similarly $s(d) := \{s(x) : x \in d\} \subseteq A$ for subsets $d \subseteq D$.

► **Definition 1.** A *team* is set X of assignments $s : D \rightarrow A$ with a common finite domain $D = \text{dom}(X)$ of variables into a set A . Besides the empty team \emptyset we also admit a unique team $\{\emptyset\}$ with empty domain and empty assignment. For every k -tuple \bar{x} of variables

22:4 Guarded Teams: The Horizontally Guarded Case

from the domain of X , let $X(\bar{x}) := \{s(\bar{x}) : s \in X\} \subseteq A^k$ denote the set of values assumed by \bar{x} in the team X . Thinking of an arbitrary but fixed enumeration of the finite domain of a team X as $\text{dom}(X) = \{x_1, \dots, x_k\}$ we often identify X with its *relational encoding* $\llbracket X \rrbracket := X(\bar{x}) = \{s(\bar{x}) : s \in X\} \subseteq A^k$.

Basic operations that possibly extend the domain of a given team to new variables are the unrestricted *generalisation* over A , $X[x \mapsto A] := \{s[x \mapsto a] : s \in X, a \in A\}$ and the *Skolem-extensions* $X[x \mapsto F] := \{s[x \mapsto a] : s \in X, a \in F(s)\}$ for any function $F : X \rightarrow \mathcal{P}^+(A)$. Note that $X[x \mapsto A] = X[x \mapsto F]$ for the constant function $F : s \mapsto A$, and that x may or may not be in the domain D of the original team X , but in any case the new team has domain $D \cup \{x\}$. Given teams X, Y with $D \subseteq \text{dom}(X) \cap \text{dom}(Y)$ we write $X \equiv_D Y$ if $(X \upharpoonright D) = (Y \upharpoonright D)$ where $(X \upharpoonright D) := \{s \upharpoonright D : s \in X\}$.

The traditional semantics (to which we refer as Tarski semantics) for first-order formulae $\varphi(\bar{x})$ is based on single assignments s whose domain must comprise the variables in $\text{free}(\varphi)$; we write $\mathfrak{A} \models \varphi[s]$ for saying that \mathfrak{A} satisfies φ with the assignment s .

The *team semantics* for $\text{FO}(\tau)$ over τ -structures \mathfrak{A} instead is defined by inductive clauses for the satisfaction relation $\mathfrak{A} \models_X \varphi$ saying that team X satisfies φ in \mathfrak{A} . Here X stands for a team in A with domain D for which we tacitly always assume that $\text{dom}(X) \supseteq \text{free}(\varphi)$.

- if φ is a literal then $\mathfrak{A} \models_X \varphi$ if $\mathfrak{A} \models \varphi[s]$ for all $s \in X$;
- $\mathfrak{A} \models_X \varphi_1 \wedge \varphi_2$ if $\mathfrak{A} \models_X \varphi_i$ for $i = 1, 2$;
- $\mathfrak{A} \models_X \varphi_1 \vee \varphi_2$ if $X = X_1 \cup X_2$ for two teams X_i such that $\mathfrak{A} \models_{X_i} \varphi_i$;
- $\mathfrak{A} \models_X \forall x \varphi$ if $\mathfrak{A} \models_Y \varphi$ for the team $Y = X[x \mapsto A]$;
- $\mathfrak{A} \models_X \exists x \varphi$ if $\mathfrak{A} \models_Y \varphi$ for some team Y of the form $Y = X[x \mapsto F]$, i.e., for some suitable Skolem extension $F : X \rightarrow \mathcal{P}(A) \setminus \{\emptyset\}$.

Note that there is no clause for negation (other than negation of atoms); we assume formulae to be written in negation normal form unless explicitly noted otherwise. It is not hard to see, through standard inductive arguments, that team semantics for FO satisfies the following principles:

Locality: whether $\mathfrak{A} \models_X \varphi$ is determined by $X \upharpoonright \text{free}(\varphi)$.

Downward closure: if $Y \subseteq X$, then $\mathfrak{A} \models_X \varphi$ implies $\mathfrak{A} \models_Y \varphi$.

Flatness: $\mathfrak{A} \models_X \varphi$ if, and only if, $\mathfrak{A} \models_{\{s\}} \varphi$ for all $s \in X$.

Empty team property: $\mathfrak{A} \models_{\emptyset} \varphi$ for all φ .

Union closure: if $\mathfrak{A} \models_{X_i} \varphi$ for all $i \in I$, then $\mathfrak{A} \models_X \varphi$ for $X := \bigcup_{i \in I} X_i$.

Other propositional connectives. For some purposes it is useful to consider a team semantic interpretation of other natural propositional connectives.

Implication: $\mathfrak{A} \models_X \psi \rightarrow \varphi$ if, for all teams $Y \subseteq X$ with $\mathfrak{A} \models_Y \psi$, also $\mathfrak{A} \models_Y \varphi$.

Intuitionistic disjunction: $\mathfrak{A} \models_X \psi \otimes \varphi$ if $\mathfrak{A} \models_X \psi$ or $\mathfrak{A} \models_X \varphi$.

Classical negation: $\mathfrak{A} \models_X \text{non } \varphi$ if it is not the case that $\mathfrak{A} \models_X \varphi$.

Nonemptiness: this is a nullary connective or logical constant without a classical counterpart, which we denote as **NE**, with $\mathfrak{A} \models_X \text{NE}$ if $X \neq \emptyset$.

Falsum: this is a nullary connective as in the classical setting. Keeping in mind the empty team property, $\mathfrak{A} \models_X \perp$ for just $X = \emptyset$.

Regarding ordinary negation (\neg) in team semantics, which we here only allow at the atomic level, it is important to note that, in contrast to classical negation (**non**), it does *not* support the classical principle of the excluded middle (*tertium non datur*). Clearly there are teams that do not uniformly make up their mind between α and $\neg\alpha$ even for equalities or

relational atoms α . The following logical equivalences between some of the above are easily proved, for any atomic α and arbitrary φ_i

$$\text{NE} \equiv \text{non } \perp, \quad \perp \equiv \alpha \wedge \neg\alpha, \quad \varphi_1 \otimes \varphi_2 \equiv \text{non}(\text{non } \varphi_1 \wedge \text{non } \varphi_2).$$

These mark out classical negation as an augmentation of FO in the team semantic setting, which no longer satisfies any of the semantic criteria highlighted above, apart from locality. We denote as FO(non) the extension of FO with strong classical negation; corresponding notation for other choices of additional connectives, like FO(NE, \otimes), is self-explanatory.

Logics of dependence and independence. Team semantics is particularly important as the basis for logics that extend first-order logic by atomic team properties such as dependence, inclusion, exclusion, independence, and others. The best studied such logic is *dependence logic* [25], which extends first-order logic by dependency atoms of form $\text{dep}(\bar{x}, \bar{y})$, saying that the values for \bar{y} are functionally dependent on (i.e. completely determined by) the values for \bar{x} . Other important such logics include different variants of *independence logics* [16], further *inclusion logic* FO(\subseteq), which is based on inclusion dependencies ($\bar{x} \subseteq \bar{y}$) saying that every value for \bar{x} in the team also occurs as a value for \bar{y} , and dually, *exclusion logic*, based on exclusion statements ($\bar{x} \mid \bar{y}$), saying that \bar{x} and \bar{y} have disjoint sets of values in the given team.

One way to describe the expressive power of a logic with team semantics is to relate it to some well-understood logic with classical Tarski semantics. One translates formulae $\varphi(\bar{x})$ from a logic $L(\tau)$ with team semantics into *sentences* φ^T , with Tarski semantics, of vocabulary $\tau \dot{\cup} \{T\}$ where T is an additional relation symbol for the team, such that for every structure \mathfrak{A} and every team X we have that

$$\mathfrak{A} \models_X \varphi(\bar{x}) \iff (\mathfrak{A}, [X]) \models \varphi^T,$$

where $[X]$ is the relational encoding of the team X (see Definition 1). In all logics with team semantics that extend first-order formulae by atomic dependencies that are themselves first-order definable, and which do not make use of additional connectives beyond \wedge, \vee and atomic negation, such a translation will always produce sentences in (a fragment of) existential second-order logic Σ_1^1 . Understanding the expressive power of a logic L with team semantics thus means to identify the fragment of Σ_1^1 to which L is equivalent in the sense just described.

- (1) Dependence logic and exclusion logic are equivalent to the fragment of Σ_1^1 -sentences $\psi(T)$ in which the predicate T describing the team appears only negatively [21].
- (2) Independence logic and inclusion-exclusion logic are equivalent with full Σ_1^1 (and thus can describe all NP-properties of teams) [9].
- (3) Any fragment $L \subseteq \text{FO}$, without any dependence properties, corresponds by flatness to the class $[L]^T$ of sentences of form $\forall \bar{x}(T\bar{x} \rightarrow \varphi(\bar{x}))$ where $\varphi(\bar{x}) \in L$ does not contain T .
- (4) Inclusion logic is equivalent to the set of sentences of form $\forall \bar{x}(X\bar{x} \rightarrow \psi(X, \bar{x}))$, where $\psi(X, \bar{x})$ is a formula in the posGFP-fragment of least fixed-point logic, in which X occurs only positively [10].

In the presence of additional propositional connectives such as implication or strong negation, such translations may produce sentences in full second-order logic (SO), rather than its existential fragment Σ_1^1 .

3 Horizontally guarded first-order logic

We deal with purely relational, finite vocabularies τ . Every interpretation of the relations in a τ -structure $\mathfrak{A} = (A, (R^{\mathfrak{A}})_{R \in \tau})$ induces a notion of *guarded subsets* and *guarded tuples*.

► **Definition 2.** The set $\mathbb{G}(\mathfrak{A})$ of *guarded subsets* of \mathfrak{A} is the downward closure of the collection of all sets $[\bar{a}]$ for any atomic fact $R\bar{a}$ that holds in \mathfrak{A} , together with all singleton subsets $\{a\} \subseteq A$. Formally $\mathbb{G}(\mathfrak{A}) = \{B \subseteq [\bar{a}] : \bar{a} \in R^{\mathfrak{A}}, R \in \tau\} \cup \{\{a\} : a \in A\}$. A tuple $\bar{a} = (a_1, \dots, a_k) \in A^k$ is guarded in \mathfrak{A} if the set of its components $[\bar{a}]$ is. An assignment $s : D \rightarrow A$ is guarded if $s(D) \in \mathbb{G}(\mathfrak{A})$ which is the case if, and only if, $s(\bar{x})$ is a guarded tuple for any tuple of variables from D . A relation $T \subseteq A^k$ is guarded in \mathfrak{A} if it only consists of guarded tuples, and this is the case if, and only if, $\mathbb{G}((\mathfrak{A}, T)) = \mathbb{G}(\mathfrak{A})$, i.e. the expansion of \mathfrak{A} by T does not introduce any new guarded subsets. A team X is *horizontally guarded* if it only consists of guarded assignments. We write $\mathbb{H}(\mathfrak{A})$ for the collection of all horizontally guarded teams over \mathfrak{A} .

Note that $\mathbb{H}(\mathfrak{A})$ is closed under subsets and restrictions of teams. The Skolem extensions $X[x \mapsto F]$ of a horizontally guarded team X will not in general be horizontally guarded, but we have the following.

► **Lemma 3.** *Every horizontally guarded team $X \in \mathbb{H}(\mathfrak{A})$ possesses, for every variable x , a unique maximal Skolem extension $Y = X[x \mapsto F] \in \mathbb{H}(\mathfrak{A})$.*

Proof. For $D = \text{dom}(X) \setminus \{x\}$, put $Y := \{s[x \mapsto a] : s \in X, a \in A, s(D) \cup \{a\} \in \mathbb{G}(\mathfrak{A})\}$. This team is easily checked to be maximal among all teams $Y \in \mathbb{H}(\mathfrak{A})$ with $\text{dom}(Y) = \text{dom}(X) \cup \{x\}$ and $Y \equiv_D X$. ◀

We are now ready to introduce the *horizontally guarded team semantics* $\mathfrak{A} \models_X^{\text{hg}} \varphi$ of first-order formulae for τ -structures \mathfrak{A} and teams $X \in \mathbb{H}(\mathfrak{A})$. For its definition we modify the clauses in the standard definition of team semantics so as to restrict all relevant teams to $\mathbb{H}(\mathfrak{A})$. This modification is trivial for literals and conjunctions and obvious for disjunction since $\mathbb{H}(\mathfrak{A})$ is downward closed. For universal and existential quantification, the restriction is more interesting.

- $\mathfrak{A} \models_X^{\text{hg}} \varphi_1 \vee \varphi_2$ if $X = X_1 \cup X_2$ for two teams $X_i \in \mathbb{H}(\mathfrak{A})$ such that $\mathfrak{A} \models_{X_i}^{\text{hg}} \varphi_i$;
- $\mathfrak{A} \models_X^{\text{hg}} \forall x \varphi$ if $\mathfrak{A} \models_Y \varphi$ for the maximal horizontally guarded Skolem extension of $X \upharpoonright (\text{free}(\varphi) \setminus \{x\})$;
- $\mathfrak{A} \models_X^{\text{hg}} \exists x \varphi$ if $\mathfrak{A} \models_Y \varphi$ for some horizontally guarded Skolem extension Y of $X \upharpoonright (\text{free}(\varphi) \setminus \{x\})$.

Horizontally guarded first-order logic is the logic FO^{hg} with usual syntax in negation normal form and semantics for horizontally guarded teams as defined above. FO^{hg} satisfies the familiar properties of first-order team semantics, locality, downward closure, and flatness.

► **Lemma 4.** *For every $\varphi \in \text{FO}^{\text{hg}}$, every structure \mathfrak{A} , and every team $X \in \mathbb{H}(\mathfrak{A})$ with $\text{free}(\varphi) \subseteq \text{dom}(X)$, we have*

Locality: $\mathfrak{A} \models_X^{\text{hg}} \varphi \Leftrightarrow \mathfrak{A} \models_Y^{\text{hg}} \varphi$ whenever $X \equiv_{\text{free}(\varphi)} Y$.

Downward closure: If $Y \subseteq X$ and $\mathfrak{A} \models_X^{\text{hg}} \varphi$, then also $\mathfrak{A} \models_Y^{\text{hg}} \varphi$.

Flatness: $\mathfrak{A} \models_X^{\text{hg}} \varphi$ if, and only if, $\mathfrak{A} \models_{\{s\}}^{\text{hg}} \varphi$ for all $s \in X$.

The difference between $\mathfrak{A} \models_{\{s\}}^{\text{hg}} \varphi$ (in the sense of FO^{hg}) and $\mathfrak{A} \models \varphi[s]$ (in the sense of ordinary first-order logic) has nothing to do with team semantics but just with the implicit relativisation to guarded assignments in \models^{hg} . In the classical first-order setting for single

assignments, this corresponds to the (explicit) relativisation to guarded assignments in the guarded fragment $\text{GF}(\tau) \subseteq \text{FO}(\tau)$. And indeed, there is a straightforward translation also in the case of team semantics.

► **Definition 5.** The guarded fragment $\text{GF} \subseteq \text{FO}$ is the syntactic fragment of FO generated from atomic formulae by the boolean connectives and quantifications of the form $\exists \bar{y}(\alpha(\bar{x}\bar{y}) \wedge \varphi(\bar{x}\bar{y}))$, and, dually, $\forall \bar{y}(\alpha(\bar{x}\bar{y}) \rightarrow \varphi(\bar{x}\bar{y}))$, where $\varphi(\bar{x}\bar{y}) \in \text{GF}$ has free variables among those listed in $\bar{x}\bar{y}$ and $\alpha(\bar{x}\bar{y})$ is an atomic formula in which all the listed variables occur. The formula α is called the *guard* of this quantification.¹ The semantics of GF is that of FO .

It is obvious that $\text{GF}(\tau) \subseteq \text{FO}(\tau)$ inherits from $\text{FO}(\tau)$ the locality, downward closure and flatness properties for its team semantics. Note that $\text{GF}(\tau)$ involves, in the first-order correspondent for universal guarded quantification, the use of implication as a propositional connective and recall its team semantics, which is downward closed by definition. As the classical equivalence $\alpha \rightarrow \varphi \equiv \neg\alpha \vee \varphi$ only involves the negation of an atomic guard α , it persists as an equivalence in team semantics. This remains true more generally in any context where the team semantics of $(\neg)\alpha$ is flat and that of φ is downward closed.

► **Lemma 6.** *Assume that $\varphi[\mathfrak{A}] = \{X: \mathfrak{A} \models_X \varphi\}$ is downward closed and that the team semantics of α and $\neg\alpha$ is flat. Then, for all teams X , $\mathfrak{A} \models_X \alpha \rightarrow \varphi \Leftrightarrow \mathfrak{A} \models_X \neg\alpha \vee \varphi$. The same logical equivalence holds in terms of horizontally guarded team semantics (keeping in mind that the class of horizontally guarded teams is downward closed).*

Recall that the set of guarded tuples $\bar{a} = (a_1, \dots, a_n) \in A^n$ is uniformly first-order definable in τ -structures \mathfrak{A} , for any fixed length $n \geq 1$ and fixed finite relational τ , by a formula

$$\text{gd}(\bar{x}) := \bigwedge_{i \leq n} x_i = x_1 \vee \bigvee_{R \in \tau} \exists \bar{y} \left(R\bar{y} \wedge \bigwedge_{i \leq n} \bigvee_{j \leq \text{ar}(R)} x_i = y_j \right).$$

In the following we regard, for every finite τ and every \bar{x} , the formula $\text{gd}(\bar{x})$ as a new atomic formula and in particular also allow its negation $\neg\text{gd}(\bar{x})$ which is correspondingly interpreted in the flat sense of $\llbracket X \rrbracket \cap \mathbb{G}(\mathfrak{A}) = \emptyset$:

$$\mathfrak{A} \models_X \neg\text{gd}(\bar{x}) \text{ if } \mathfrak{A} \models \neg\text{gd}[s(\bar{x})] \text{ for all } s \in X.$$

Note that, with this stipulation, $\text{gd}(\bar{x})$ becomes a formula satisfying the requirements for α in Lemma 6, so that, for any $\varphi(\bar{x})$ whose team semantics is downward closed, we have the usual (classical) equivalence $\text{gd}(\bar{x}) \rightarrow \varphi \equiv \neg\text{gd}(\bar{x}) \vee \varphi$.

► **Proposition 7.** *For finite relational τ , there is a translation $\varphi(\bar{x}) \mapsto \varphi^{\text{hg}}(\bar{x})$ from $\text{FO}(\tau)$ to $\text{GF}(\tau)$ such that for all guarded assignments s , and all teams $X \in \mathbb{H}(\mathfrak{A})$,*

$$\mathfrak{A} \models_{\{s\}}^{\text{hg}} \varphi \Leftrightarrow \mathfrak{A} \models \varphi^{\text{hg}}[s] \quad \text{and} \quad \mathfrak{A} \models_X^{\text{hg}} \varphi \Leftrightarrow \mathfrak{A} \models_X \varphi^{\text{hg}}.$$

An analogous translation works for extensions of FO and GF by arbitrary Σ_1^1 -definable atomic dependence relations.

Proof. Define $\varphi \mapsto \varphi^{\text{hg}}$ by induction on $\varphi \in \text{FO}(\tau)$. The only non-trivial steps are those for the quantifiers. For $\varphi(\bar{x}) = \exists y\psi(\bar{x}y)$, put $\varphi^{\text{hg}}(\bar{x}) := \exists y(\text{gd}(\bar{x}y) \wedge \psi^{\text{hg}}(\bar{x}y))$, and for $\varphi(\bar{x}) = \forall y\psi(\bar{x}y)$, set

$$\varphi^{\text{hg}}(\bar{x}) := \forall y(\text{gd}(\bar{x}y) \rightarrow \psi^{\text{hg}}(\bar{x}y)) \equiv \forall y(\neg\text{gd}(\bar{x}y) \vee \psi^{\text{hg}}(\bar{x}y)).$$

¹ If $\bar{x}\bar{y}$ consists of a single variable symbol z , α can be the equality $z = z$.

It is straightforward to verify that these stipulations support the equivalence claim for singleton teams. The equivalence claim for arbitrary teams follows by the flatness properties for FO^{hg} and FO . ◀

One also checks that $\mathfrak{A} \models_X \varphi^{\text{hg}} \Leftrightarrow \mathfrak{A} \models_X^{\text{hg}} \varphi^{\text{hg}}$, whence also $\mathfrak{A} \models_X^{\text{hg}} \varphi \Leftrightarrow \mathfrak{A} \models_X^{\text{hg}} \varphi^{\text{hg}}$, so that the map \cdot^{hg} provides a normal form for FO w.r.t. its horizontally guarded team semantics.

Since most of the standard logics with team semantics have the empty team property, the natural variants of satisfiability problems in this context ask whether a given a formula φ admits a structure \mathfrak{A} and a *nonempty* team X such that $\mathfrak{A} \models_X \varphi$. Notice that, by flatness, the question whether a guarded first-order formula $\varphi(\bar{x}) \in \text{GF}$ is satisfiable in this sense is equivalent to asking whether $\varphi(\bar{x})$ is satisfiable in the usual sense of Tarski semantics, which is well-known to be decidable [2, 12].

4 Two notions of guarded team bisimulation

The natural notion of back and forth equivalence for guarded logics is *guarded bisimulation equivalence*. Just as the model theory of modal logics is governed by (modal) bisimulation equivalence, the nice model-theoretic properties of guarded logics are closely related to its invariance under guarded bisimulation equivalence $\sim_{\mathfrak{g}}$ and its finite approximations $\sim_{\mathfrak{g}}^{\ell}$. For a detailed discussion of guarded bisimulation and beyond, we refer to [15]. In the context investigated here it is convenient to view a guarded bisimulation between two structures \mathfrak{A} and \mathfrak{B} as a set Z of pairs (s, t) of guarded assignments that induce local isomorphisms between the two structures, and satisfy appropriate back and forth properties.

► **Definition 8.** A guarded bisimulation between τ -structures \mathfrak{A} and \mathfrak{B} is a set Z of pairs of guarded assignments $s : [\bar{x}] \rightarrow A$ and $t : [\bar{x}] \rightarrow B$, with $\text{dom}(s) = \text{dom}(t)$, such that, for all $(s, t) \in Z$:

- (i) $s \mapsto t$ induces a local isomorphism from \mathfrak{A} to \mathfrak{B} . This means that for every atomic formulae α with $\text{free}(\alpha) \subseteq \text{dom}(s) = \text{dom}(t)$ we have that $\mathfrak{A} \models \alpha[s] \iff \mathfrak{B} \models \alpha[t]$.
- (ii) (*back*): for every guarded assignment t' into \mathfrak{B} that coincides with t on $\text{dom}(t') \cap \text{dom}(t)$ there is a guarded assignment s' into \mathfrak{A} that coincides with s on $\text{dom}(s') \cap \text{dom}(s)$ such that (s', t') is also in Z .
- (iii) (*forth*): for every guarded assignment s' into \mathfrak{A} that coincides with s on $\text{dom}(s') \cap \text{dom}(s)$ there is a guarded assignment t' into \mathfrak{B} that coincides with t on $\text{dom}(t') \cap \text{dom}(t)$ such that (s', t') is also in Z .

We write $\mathfrak{A}, s \sim_{\mathfrak{g}} \mathfrak{B}, t$ if there is a guarded bisimulation Z between \mathfrak{A} and \mathfrak{B} such that $(s, t) \in Z$. Further we write $\mathfrak{A} \sim_{\mathfrak{g}} \mathfrak{B}$ if $\mathfrak{A}, \emptyset \sim_{\mathfrak{g}} \mathfrak{B}, \emptyset$.

There is an obvious game-theoretic presentation of this in terms of *guarded bisimulation games* on $(\mathfrak{A}, \mathfrak{B})$ whose positions are the pairs (s, t) of guarded assignments that induce a local isomorphism as described above. Then the available moves for the first player, e.g. on the \mathfrak{A} -side, are to guarded assignments s' such that $s'(x) = s(x)$ for all variables $x \in \text{dom}(s') \cap \text{dom}(s)$. The second player then has to respond with a guarded assignment t' with $\text{dom}(t') = \text{dom}(s')$, such that $t'(x) = t(x)$ for $x \in \text{dom}(t') \cap \text{dom}(t)$, and (s', t') is again a valid position of the game.

Finite approximations $\sim_{\mathfrak{g}}^{\ell}$ of $\sim_{\mathfrak{g}}$ correspond to the existence of winning strategies for the second player for ℓ rounds in the guarded bisimulation game, and $\sim_{\mathfrak{g}}^{\omega}$ is defined as the common refinement of the finite levels $\sim_{\mathfrak{g}}^{\ell}$.

One obtains natural variants of the first-order Ehrenfeucht–Fraïssé Theorem for GF. The equivalence relations \equiv_{GF}^ℓ and \equiv_{GF} are defined as levels of elementary equivalence in GF, where the ℓ in \equiv_{GF}^ℓ refers to the nesting depth of guarded quantification (which is typically lower than the first-order quantifier rank, as guarded quantification may quantify over tuples in a single step). The relation $\equiv_{\text{GF}}^\infty$ similarly denotes equivalence w.r.t. the infinitary variant of GF, with infinite disjunctions and conjunctions. For more details, we refer to [15].

► **Theorem 9** (Ehrenfeucht–Fraïssé Theorems for GF). *For finite relational vocabularies, and for every $\ell \in \mathbb{N}$:*

$$\mathfrak{A}, s \sim_{\mathfrak{g}}^\ell \mathfrak{B}, t \iff \mathfrak{A}, s \equiv_{\text{GF}}^\ell \mathfrak{B}, t \quad \text{and} \quad \mathfrak{A}, s \sim_{\mathfrak{g}}^\omega \mathfrak{B}, t \iff \mathfrak{A}, s \equiv_{\text{GF}} \mathfrak{B}, t.$$

Further, without restriction on the size of the vocabulary, $\mathfrak{A}, s \sim_{\mathfrak{g}} \mathfrak{B}, t \iff \mathfrak{A}, s \equiv_{\text{GF}}^\infty \mathfrak{B}, t$.

Just as in the classical first-order case (cf. [7, 8]) the implication from \equiv_{GF}^ℓ to $\sim_{\mathfrak{g}}^\ell$ relies on the fact that both equivalence relations have finite index, and that the $\sim_{\mathfrak{g}}^\ell$ -equivalence classes of \mathfrak{A}, s are naturally definable by *characteristic formulae* $\chi_{\mathfrak{A}, s}^\ell \in \text{GF}$.

We now generalise the notion of guarded bisimulation equivalence from individual assignments to teams. It turns out that there are two different ways to do this, a basic one and a stronger one.

► **Definition 10.** *Guarded team bisimulation equivalence, $\mathfrak{A}, X \sim_{\mathfrak{g}} \mathfrak{B}, Y$ and its finite approximations $\mathfrak{A}, X \sim_{\mathfrak{g}}^\ell \mathfrak{B}, Y$ are defined in a flat manner. Horizontally guarded teams $X \in \mathbb{H}(\mathfrak{A})$ and $Y \in \mathbb{H}(\mathfrak{B})$, with the same domain, are *guarded team bisimilar*, $\mathfrak{A}, X \sim_{\mathfrak{g}} \mathfrak{B}, Y$ if for every $s \in X$ there is some $t \in Y$ such that $\mathfrak{A}, s \sim_{\mathfrak{g}} \mathfrak{B}, t$, and vice versa. Guarded team ℓ -bisimilarity, $\mathfrak{A}, X \sim_{\mathfrak{g}}^\ell \mathfrak{B}, Y$, is defined analogously.*

Ordinary guarded bisimulation equivalences between individual assignments, like $\mathfrak{A}, s \sim_{\mathfrak{g}} \mathfrak{B}, t$, are captured in this definition via the encodings of tuples as singleton teams: $\mathfrak{A}, \{s\} \sim_{\mathfrak{g}} \mathfrak{B}, \{t\}$, and for naked structures, we have that $\mathfrak{A} \sim_{\mathfrak{g}} \mathfrak{B}$ if, and only if, $\mathfrak{A}, \{\emptyset\} \sim_{\mathfrak{g}} \mathfrak{B}, \{\emptyset\}$. For the empty team, however, the above definition says that $\mathfrak{A}, \emptyset \sim_{\mathfrak{g}} \mathfrak{B}, \emptyset \not\sim_{\mathfrak{g}} \mathfrak{B}, Y$ for any $Y \neq \emptyset$ and all τ -structures $\mathfrak{A}, \mathfrak{B}$. It readily follows from the definition that GF and FO^{hg} are invariant under this notion of team bisimulation.

► **Proposition 11.** *If $\mathfrak{A}, X \sim_{\mathfrak{g}} \mathfrak{B}, Y$ then $\mathfrak{A} \models_X \varphi \iff \mathfrak{B} \models_Y \varphi$ for any $\varphi \in \text{GF}$, and $\mathfrak{A} \models_X^{\text{hg}} \varphi \iff \mathfrak{B} \models_Y^{\text{hg}} \varphi$ for every $\varphi \in \text{FO}$.*

Beyond this essentially flat notion of guarded team bisimulation, there is a stronger one which focuses on the relational encoding of those teams as guarded relations.

► **Definition 12.** *Strong guarded team bisimulation equivalence, $\mathfrak{A}, X \approx_{\mathfrak{g}} \mathfrak{B}, Y$ is defined for teams $X \in \mathbb{H}(\mathfrak{A})$ and $Y \in \mathbb{H}(\mathfrak{B})$ with the same finite domain by the condition that $(\mathfrak{A}, \llbracket X \rrbracket) \sim_{\mathfrak{g}} (\mathfrak{B}, \llbracket Y \rrbracket)$, in terms of ordinary guarded bisimulation equivalence between the expansions of the two τ -structures by the relational encoding of the teams as $(\tau \dot{\cup} \{T\})$ -structures for a new relation symbol T of the appropriate arity. Strong guarded team ℓ -bisimilarity, $\mathfrak{A}, X \approx_{\mathfrak{g}}^\ell \mathfrak{B}, Y$, is analogously defined.*

Obviously $\mathfrak{A}, X \approx_{\mathfrak{g}} \mathfrak{B}, Y$ implies $\mathfrak{A}, X \sim_{\mathfrak{g}} \mathfrak{B}, Y$, and $\mathfrak{A}, X \approx_{\mathfrak{g}}^{\ell+1} \mathfrak{B}, Y$ implies $\mathfrak{A}, X \sim_{\mathfrak{g}}^\ell \mathfrak{B}, Y$, for every $\ell \in \mathbb{N}$ (the formal offset of 1 in finite approximation levels is a consequence of the fact that $\approx_{\mathfrak{g}}^0$ is trivial.) In particular, any team property that is $\sim_{\mathfrak{g}}$ invariant, is also $\approx_{\mathfrak{g}}$ -invariant. We shall see in Sect. 7 that the converse fails in general.

5 The flatness of guarded team bisimulation

Horizontal guardedness is compatible with disjoint unions of relational structures (and teams). For any τ -structures \mathfrak{A}_1 and \mathfrak{A}_2 , we have $\mathbb{G}(\mathfrak{A}_1 \oplus \mathfrak{A}_2) = \mathbb{G}(\mathfrak{A}_1) \cup \mathbb{G}(\mathfrak{A}_2)$ and hence also $\mathbb{H}(\mathfrak{A}_1 \oplus \mathfrak{A}_2) = \{X_1 \cup X_2 : X_i \in \mathbb{H}(\mathfrak{A}_i) \text{ for } i = 1, 2\}$.

Also, for any two pairs of τ -structures $\mathfrak{A}_1, \mathfrak{A}_2$ and $\mathfrak{B}_1, \mathfrak{B}_2$ with horizontally guarded teams $X_i \in \mathbb{H}(\mathfrak{A}_i)$ and $Y_i \in \mathbb{H}(\mathfrak{B}_i)$ such that $\mathfrak{A}_i, X_i \sim_{\mathbf{g}}^{\ell} \mathfrak{B}_i, Y_i$ for $i = 1, 2$, it follows that

$$(\mathfrak{A}_1 \oplus \mathfrak{A}_2), X_1 \cup X_2 \sim_{\mathbf{g}}^{\ell} (\mathfrak{B}_1 \oplus \mathfrak{B}_2), Y_1 \cup Y_2$$

and similarly for $\sim_{\mathbf{g}}$. The main point here is that every guarded assignment is fully contained in one component: correspondingly, whenever the first player makes a move in one of the disjoint unions that goes from one component to the other, so can the second player in the opposite structure, since the assumptions in particular guarantee $\mathfrak{A}_i \sim_{\mathbf{g}}^{\ell} \mathfrak{B}_i$ for the naked component structures.

Given a τ -structure \mathfrak{A} and a team $X \in \mathbb{H}(\mathfrak{A})$, we write $\bigoplus_{s \in X} (\mathfrak{A}, \{s\})$ for the disjoint union of copies of \mathfrak{A} together with the singleton teams $\{s\}$ for the assignments $s \in X$. The following observation is immediate from the definition of guarded team bisimulation.

► **Proposition 13.** *For every structure \mathfrak{A} , we have $\mathfrak{A}, X \sim_{\mathbf{g}} \bigoplus_{s \in X} (\mathfrak{A}, \{s\})$.*

As a further illustration of the interesting interplay between guarded team bisimulation, ordinary guarded bisimulation, flatness and downward closure, we may look at representatives for $\sim_{\mathbf{g}}^{\ell}$ - or $\sim_{\mathbf{g}}$ -classes that are built from singleton team configurations. We just formulate these considerations for $\sim_{\mathbf{g}}^{\ell}$, but the situation for $\sim_{\mathbf{g}}$ is analogous (as long as we are not concerned about definability in GF or FO^{hg}). Let $[\mathfrak{A}, X]^{\ell}$ denote the $\sim_{\mathbf{g}}^{\ell}$ -class of the horizontally guarded team configuration \mathfrak{A}, X within the class of τ -structures with horizontally guarded teams. Let $[\mathfrak{A}, s]^{\ell}$ similarly stand for the $\sim_{\mathbf{g}}^{\ell}$ -class in the sense of ordinary guarded bisimulation within the class of τ -structures with guarded assignments. Then, for a singleton team $X = \{s\}$, we classically find

$$[\mathfrak{A}, s]^{\ell} = \{\mathfrak{B}, t : \mathfrak{B}, t \sim_{\mathbf{g}}^{\ell} \mathfrak{A}, s\} = \{\mathfrak{B}, t : \mathfrak{B}, Y \sim_{\mathbf{g}}^{\ell} \mathfrak{A}, \{s\}, t \in Y\} = \{\mathfrak{B}, t : \mathfrak{B} \models \chi_{\mathfrak{A}, s}^{\ell}[t]\}$$

where this definability relies on the characteristic formulae in the standard proof of the Ehrenfeucht–Fraïssé Theorem for GF, Theorem 9 above, and requires τ to be finite. On the other hand, in team terms,

$$\begin{aligned} [\mathfrak{A}, \{s\}]^{\ell} &= \{\mathfrak{B}, Y : \mathfrak{B}, Y \sim_{\mathbf{g}}^{\ell} \mathfrak{A}, \{s\}\} = \{\mathfrak{B}, Y : \mathfrak{B}, t \sim_{\mathbf{g}}^{\ell} \mathfrak{A}, s \text{ for all } t \in Y\} \\ &= \{\mathfrak{B}, Y : \mathfrak{B} \models_Y \chi_{\mathfrak{A}, s}^{\ell}\}, \end{aligned}$$

where the last equality relies on flatness (for the adequacy of the team semantic reading) and finiteness of τ (for the existence of the characteristic formulae χ , which we here use in negation normal form). For an arbitrary horizontally guarded team configuration \mathfrak{A}, X , correspondingly

$$\begin{aligned} [\mathfrak{A}, X]^{\ell} &= \{\mathfrak{B}, Y : \mathfrak{B}, Y \sim_{\mathbf{g}}^{\ell} \mathfrak{A}, X\} \\ &= \{\mathfrak{B}, \bigcup_{s \in X} Y_s : Y_s \neq \emptyset \text{ and, for all } t \in Y_s, \mathfrak{B}, t \sim_{\mathbf{g}}^{\ell} \mathfrak{A}, s\} \\ &= \{\mathfrak{B}, Y : \mathfrak{B} \models_Y \bigvee_{s \in X} (\text{NE} \wedge \chi_{\mathfrak{A}, s}^{\ell})\}. \end{aligned}$$

Note that the nonemptiness condition indicates that (as expected) team guarded bisimulation equivalence does not respect downward closure. W.r.t. the last equality, finiteness of τ also ensures that the disjunction is finite up to logical equivalence.

The downward closure of $[\mathfrak{A}, X]^\ell$, on the other hand, has the simpler form

$$\begin{aligned} [\mathfrak{A}, X]^\ell \downarrow &= \{ \mathfrak{B}, Y' : Y' \subseteq Y \text{ for some } \mathfrak{B}, Y \sim_g^\ell \mathfrak{A}, X \} \\ &= \{ \mathfrak{B}, \bigcup_{s \in X} Y_s : \text{for all } t \in Y_s, \mathfrak{B}, t \sim_g^\ell \mathfrak{A}, s \} \\ &= \{ \mathfrak{B}, Y : \mathfrak{B} \models_Y \bigvee_{s \in X} \chi_{\mathfrak{A}, s}^\ell \}, \end{aligned}$$

where definability according to the last equality again is only good for finite τ for the characteristic formulae $\chi_{\mathfrak{A}, \bar{x}}^\ell(\bar{x}) \in \text{GF}$. Their team semantic reading is again adequate by the flatness of $\text{GF} \subseteq \text{FO}$ and finiteness of τ also ensures that the disjunction is finite up to logical equivalence. As the team semantics for plain first-order logic $\text{FO}(\tau)$ is flat, we also obtain the following observation.

► **Lemma 14.** *Let \mathfrak{A} and \mathfrak{B} be τ -structures with teams X in A and Y in B such that for every $s \in X$ there is some $t \in Y$ such that $\mathfrak{A}, s \equiv_{\text{FO}}^q \mathfrak{B}, t$, and vice versa. Then also $\mathfrak{A}, X \equiv_{\text{FO}}^q \mathfrak{B}, Y$, i.e., the two team configurations are indistinguishable by $\text{FO}(\tau)$ -formulae $\varphi(\bar{x})$ of quantifier rank up to q .*

6 Expressive completeness for \sim_g -invariance

The following upgrading of equivalences and semantic invariance conditions is a typical ingredient in model-theoretic proofs of expressive completeness of some (fragment of a) logic for some semantically characterised class of properties. Here we want to use it for team properties that do not distinguish between guarded team-bisimilar situations. In relating invariance of first-order definable team properties under full guarded team bisimilarity to invariance under a sufficiently fine finite approximation of ℓ -bisimilarity it also may be seen as a *compactness property*, albeit one which does not rely on the classical compactness theorem for first-order logic – as is amply demonstrated by the finite model theory reading.

Recall that, with a team X in A (with a particular fixed enumeration of its domain), we associate the relation $\llbracket X \rrbracket = \{s(\bar{x}) : s \in X\}$ over A . There are several natural notions of first-order equivalence (up to a given quantifier rank q or unrestricted) between τ -structures with teams. First,

$$\mathfrak{A}, X \equiv_{\text{FO}}^q \mathfrak{B}, Y$$

holds if \mathfrak{A}, X and \mathfrak{B}, Y satisfy the same team properties that are definable in FO_q , FO with quantifier rank up to q , in terms of team semantics. Note that, due to flatness, $\mathfrak{A}, X \equiv_{\text{FO}}^q \mathfrak{B}, Y$ just requires X and Y to agree on all those $\varphi(\bar{x}) \in \text{FO}_q$ that are true or false across the whole team. Testing this for the characteristic formulae $\chi^q(\bar{x}) \in \text{FO}_q$ (in negation normal form), which characterise the full FO_q -types of tuples over finite relational signatures, we see that $\mathfrak{A}, X \equiv_{\text{FO}}^q \mathfrak{B}, Y$ implies that X and Y realise exactly the same FO_q -types of tuples. The seemingly stronger equivalence

$$\mathfrak{A}, X \equiv_{\text{FO}(\text{non})}^q \mathfrak{B}, Y$$

holds if \mathfrak{A}, X and \mathfrak{B}, Y are indistinguishable by $\text{FO}(\text{non})$ -formulae of quantifier rank up to q . An analysis of the team semantics of $\text{FO}(\text{non})$ shows, however, that also $\mathfrak{A}, X \equiv_{\text{FO}(\text{non})}^q \mathfrak{B}, Y$ if, and only if, the two teams realise exactly the same FO_q -types of tuples. So these two notions of team equivalence coincide. (The flatness character of these team equivalences is analogous to that of guarded team bisimulation \sim_g^ℓ ; it similarly casts the classical notion of q -partial isomorphy \simeq^q at the level of teams.)

$$(\mathfrak{A}, \llbracket X \rrbracket) \equiv_{\text{FO}^\tau}^q (\mathfrak{B}, \llbracket Y \rrbracket)$$

on the other hand refers to standard first-order semantics in the T -expansions and says that the $(\tau \dot{\cup} \{T\})$ -structures $(\mathfrak{A}, \llbracket X \rrbracket)$ and $(\mathfrak{B}, \llbracket Y \rrbracket)$ satisfy the same first-order *sentences* up to quantifier rank q , which by the classical Ehrenfeucht–Fraïssé theorem for finite relational vocabularies is the same as q -partial isomorphy, $(\mathfrak{A}, \llbracket X \rrbracket) \simeq^q (\mathfrak{B}, \llbracket Y \rrbracket)$. Simple Ehrenfeucht–Fraïssé arguments show that for singleton teams also this distinction does not matter. In specific situations below, this agreement can be extended further due to compatibility of \simeq^q with disjoint sums of structures.

► **Lemma 15.** *For singleton teams $X = \{s\}$ in \mathfrak{A} and $Y = \{t\}$ in \mathfrak{B} , with $s(\bar{x}) = \bar{a}$ and $t(\bar{x}) = \bar{b}$ such that $\mathfrak{A}, \bar{a} \equiv_{\text{FO}}^q \mathfrak{B}, \bar{b}$, we also have $(\mathfrak{A}, X) = (\mathfrak{A}, \{s\}) \equiv_{\text{FO}(\text{non})}^q (\mathfrak{B}, \{t\}) = (\mathfrak{B}, Y)$ as well as $(\mathfrak{A}, \llbracket X \rrbracket) = (\mathfrak{A}, \{\bar{a}\}) \equiv_{\text{FO}^T}^q (\mathfrak{B}, \{\bar{b}\}) = (\mathfrak{B}, \llbracket Y \rrbracket)$.*

We next show that, for suitable ℓ , ordinary guarded team ℓ -bisimulation equivalence $\sim_{\mathfrak{g}}^{\ell}$ can be upgraded to any one of these forms of first-order equivalence.

► **Lemma 16.** *For any ℓ that is sufficiently large in relation to q , any τ -structures \mathfrak{A} and \mathfrak{B} with horizontally guarded teams $X \in \mathbb{H}(\mathfrak{A})$ and $Y \in \mathbb{H}(\mathfrak{B})$ such that $\mathfrak{A}, X \sim_{\mathfrak{g}}^{\ell} \mathfrak{B}, Y$ admit $\sim_{\mathfrak{g}}$ -equivalent team configurations $\tilde{\mathfrak{A}}, \tilde{X} \sim_{\mathfrak{g}} \mathfrak{A}, X$ and $\tilde{\mathfrak{B}}, \tilde{Y} \sim_{\mathfrak{g}} \mathfrak{B}, Y$ such that simultaneously $\tilde{\mathfrak{A}}, \tilde{X} \equiv_{\text{FO}}^q \mathfrak{B}, \tilde{Y}$, $\tilde{\mathfrak{A}}, \tilde{X} \equiv_{\text{FO}(\text{non})}^q \mathfrak{B}, \tilde{Y}$, and $(\tilde{\mathfrak{A}}, \llbracket \tilde{X} \rrbracket) \equiv_{\text{FO}^T}^q (\mathfrak{B}, \llbracket \tilde{Y} \rrbracket)$. Moreover, for finite \mathfrak{A} and \mathfrak{B} , $\tilde{\mathfrak{A}}$ and $\tilde{\mathfrak{B}}$ can be chosen finite.*

Proof. The proof essentially uses the flatness features of ordinary guarded team bisimulation $\sim_{\mathfrak{g}}$ as expressed in Proposition 13 together with locality and flatness properties for first-order team semantics. This is combined with known model transformations that respect ordinary guarded bisimulation equivalence (guarded team bisimulation for singleton teams) and upgrade levels $\sim_{\mathfrak{g}}^{\ell}$ to levels of ordinary first-order equivalence \simeq^q or \equiv_{FO}^q between corresponding guarded tuples. The construction following [24] uses finite guarded bisimilar coverings, i.e., homomorphisms, with finite fibres, of the form $\pi: \mathfrak{A}^* \rightarrow \mathfrak{A}$ such that for all guarded assignments s of \mathfrak{A}^* , $\mathfrak{A}^*, s \sim_{\mathfrak{g}} \mathfrak{A}, \pi(s)$ due to natural guarded back-and-forth conditions for the map π . These finite coverings can be constructed such that, for any fixed level q there is a level ℓ such that for two such coverings $\pi: \mathfrak{A}^* \rightarrow \mathfrak{A}$ and $\pi': \mathfrak{B}^* \rightarrow \mathfrak{B}$ and guarded assignments s and t ,

$$(\dagger) \quad \mathfrak{A}, \pi(s) \sim_{\mathfrak{g}}^{\ell} \mathfrak{B}, \pi(t) \quad \Rightarrow \quad \mathfrak{A}^*, s \simeq^q \mathfrak{B}^*, t.$$

The combined upgrading steps are illustrated in Figure 1. The first stage (1) corresponds to an application of Proposition 13, which scatters the members of the two teams so that no two members are in the same component. The second stage (2) is by means of guarded coverings according to the above. By (\dagger) and Lemma 14 the resulting team configurations are in fact first-order team equivalent up to level q . For each individual matching pair of component structures $\mathfrak{A}^*, s(\bar{x}) \equiv_{\text{FO}}^q \mathfrak{B}^*, t(\bar{x})$, by Lemma 15, the equivalence translates into the corresponding equivalence at the level of $\text{FO}(\text{non})$ as well as for the T -expansions: $\mathfrak{A}^*, s(\bar{x}) \simeq^q \mathfrak{B}^*, t(\bar{x})$ implies $(\mathfrak{A}^*, \{s(\bar{x})\}) \simeq^q (\mathfrak{B}^*, \{t(\bar{x})\})$ and the compositionality of the Ehrenfeucht–Fraïssé game under disjoint sums then shows the desired equivalences. ◀

► **Corollary 17.**

(a) *Let $\varphi(\bar{x}) \in \text{FO}$ or $\varphi(\bar{x}) \in \text{FO}(\text{non})$ be invariant under guarded team bisimulation in the sense that $\mathfrak{A}, X \sim_{\mathfrak{g}} \mathfrak{B}, Y$ implies $\mathfrak{A} \models_X \varphi \Leftrightarrow \mathfrak{B} \models_Y \varphi$ for any $X \in \mathbb{H}(\mathfrak{A})$ and $Y \in \mathbb{H}(\mathfrak{B})$. Then φ is in fact already invariant under guarded team ℓ -bisimulation $\sim_{\mathfrak{g}}^{\ell}$ for some $\ell \in \mathbb{N}$, i.e. $\mathfrak{A}, X \sim_{\mathfrak{g}}^{\ell} \mathfrak{B}, Y$ suffices to imply $\mathfrak{A} \models_X \varphi \Leftrightarrow \mathfrak{B} \models_Y \varphi$.*

$$\begin{array}{ccc}
\mathfrak{A}, X & \xrightarrow{\sim_{\mathfrak{g}}^{\ell}} & \mathfrak{B}, Y \\
\downarrow \sim_{\mathfrak{g}} & (1) & \downarrow \sim_{\mathfrak{g}} \\
\bigoplus_{s \in X} (\mathfrak{A}, \{s\}) & \xrightarrow{\sim_{\mathfrak{g}}^{\ell}} & \bigoplus_{t \in Y} (\mathfrak{B}, \{t\}) \\
\downarrow \sim_{\mathfrak{g}} & (2) & \downarrow \sim_{\mathfrak{g}} \\
\bigoplus_{s \in X} (\mathfrak{A}^*, \{s\}) & \xrightarrow{\equiv_{\text{FO}}^q} & \bigoplus_{t \in Y} (\mathfrak{B}^*, \{t\})
\end{array}$$

■ **Figure 1** Upgrading of equivalences through structural transformations. The bottom rung simultaneously achieves q -equivalence at the level of $\text{FO}(\text{non})$ and FO^T (cf. Lemmas 15/16).

- (b) Any sentence $\varphi \in \text{FO}^T(\tau) = \text{FO}(\tau \dot{\cup} \{T\})$ that is invariant under guarded team bisimulation in the sense that $\mathfrak{A}, X \sim_{\mathfrak{g}} \mathfrak{B}, Y$ implies $(\mathfrak{A}, \llbracket X \rrbracket) \models \varphi \Leftrightarrow (\mathfrak{B}, \llbracket Y \rrbracket) \models \varphi$ for any $X \in \mathbb{H}(\mathfrak{A})$ and $Y \in \mathbb{H}(\mathfrak{B})$, is in fact $\sim_{\mathfrak{g}}^{\ell}$ -invariant for suitable ℓ such that already $\mathfrak{A}, X \sim_{\mathfrak{g}}^{\ell} \mathfrak{B}, Y$ implies $(\mathfrak{A}, \llbracket X \rrbracket) \models \varphi \Leftrightarrow (\mathfrak{B}, \llbracket Y \rrbracket) \models \varphi$.

Both assertions also hold true in the sense of finite model theory, i.e., if both the assumption and the conclusion are limited to corresponding criteria for just finite structures \mathfrak{A} and \mathfrak{B} .

Proof. In the diagram of Figure 1, φ is preserved along the vertical axes due to its preservation under guarded team bisimulation ($\sim_{\mathfrak{g}}$ -invariance). Overall, the detour through these guarded team bisimilar companions therefore shows that $\mathfrak{A} \models_X \varphi$ iff $\mathfrak{B} \models_Y \varphi$, or that $(\mathfrak{A}, \llbracket X \rrbracket) \models \varphi$ iff $(\mathfrak{B}, \llbracket Y \rrbracket) \models \varphi$, and thus establishes the desired $\sim_{\mathfrak{g}}^{\ell}$ -invariance. ◀

We are now ready to formulate two characterisation theorems for guarded team bisimulation and horizontally guarded team logics.

It follows from part (a) that any formula $\varphi(\bar{x}) \in \text{FO}(\tau)$ that is invariant under ordinary guarded team bisimulation $\sim_{\mathfrak{g}}$ is expressible in any logic with team semantic disjunction that is sufficiently expressive to define the classes

$$\begin{aligned}
\llbracket \mathfrak{A}, \{s\} \rrbracket^{\ell} \downarrow &= \{ \mathfrak{B}, Y' : Y' \subseteq Y \text{ for some } \mathfrak{B}, Y \sim_{\mathfrak{g}}^{\ell} \mathfrak{A}, \{s\} \} \\
&= \{ \mathfrak{B}, Y : \mathfrak{B}, t \sim_{\mathfrak{g}}^{\ell} \mathfrak{A}, s \text{ for all } t \in Y \}
\end{aligned}$$

for every τ -structure \mathfrak{A} , every guarded assignment s , and every $\ell \in \mathbb{N}$. The reason is that in this case $\varphi(\bar{x})$, which we know to define both a flat team property and a $\sim_{\mathfrak{g}}^{\ell}$ -closed team property for suitable ℓ , is then logically equivalent to the formula $\bigvee \{ \chi_{\mathfrak{A}, s}^{\ell} : \mathfrak{A}, \{s\} \models \varphi \}$. Here the formulae $\chi_{\mathfrak{A}, s}^{\ell} \in \text{GF}(\tau)$ are from the proof of Theorem 9 and define, in team semantics, the downward closures of the $\sim_{\mathfrak{g}}$ -equivalence classes $\llbracket \mathfrak{A}, \{s\} \rrbracket^{\ell}$. This conclusion holds true, in particular, for the logics $\text{FO}(\tau)$ with horizontally guarded team semantics and for $\text{GF}(\tau)$ with (horizontally guarded or general) team semantics. In other words, if \mathcal{C} is a class of team configurations \mathfrak{A}, X which is closed under $\sim_{\mathfrak{g}}^{\ell}$ and downward closed and union closed for teams, then $\mathcal{C} = \{ \mathfrak{B}, Y : \mathfrak{B} \models_Y \bigvee_{\mathfrak{A}, X \in \mathcal{C}} \bigvee_{s \in X} (\chi_{\mathfrak{A}, s}^{\ell}(\bar{x})) \}$, and, in terms of FO^T or GF^T ,

$$\mathcal{C} = \left\{ \mathfrak{B}, Y : (\mathfrak{B}, \llbracket Y \rrbracket) \models \bigvee_{\mathfrak{A}, X \in \mathcal{C}} \forall \bar{x} \left(T\bar{x} \rightarrow \bigvee_{s \in X} \chi_{\mathfrak{A}, s}^{\ell}(\bar{x}) \right) \right\}.$$

► **Corollary 18** (First Characterisation Theorem for Guarded Team Semantics).

$$\text{FO}/\sim_{\mathfrak{g}} \equiv \text{FO}^{\text{hg}} \equiv \text{GF} \equiv [\text{GF}]^T \equiv [\text{FO}]^T/\sim_{\mathfrak{g}}.$$

22:14 Guarded Teams: The Horizontally Guarded Case

This, and the following characterisation theorem, are to be read in the sense that these logical formalisms define exactly the same properties of horizontally guarded teams. Recall that, for any $L \subseteq \text{FO}$, $[L]^T$ denotes the set of all sentences $\forall \bar{x}(T\bar{x} \rightarrow \varphi(\bar{x}))$ (with classical Tarski semantics) such that $\varphi(\bar{x}) \in L$, so the last two equivalences follow from classical results. Similarly, part (b) of Corollary 17 shows the following for any team property expressed by a sentence $\varphi \in \text{FO}(\tau \dot{\cup} \{T\})$ in the classical first-order semantics with appeal to the relational encoding $\llbracket X \rrbracket$ of teams X (which does not imply downward closure or flatness!). If φ is invariant under guarded team bisimulation $\sim_{\mathbf{g}}$, then it is equivalently expressible in any logic with ordinary disjunction that is sufficiently expressive to define all unions of classes

$$\begin{aligned} [\mathfrak{A}, X]^\ell &= \{ \mathfrak{B}, Y : \mathfrak{B}, Y \sim_{\mathbf{g}}^\ell \mathfrak{A}, X \} \\ &= \{ \mathfrak{B}, \bigcup_{s \in X} Y_s : Y_s \neq \emptyset \text{ and, for all } t \in Y_s, \mathfrak{B}, t \sim_{\mathbf{g}}^\ell \mathfrak{A}, s \}, \end{aligned}$$

for every fixed $\ell \in \mathbb{N}$. As we saw above, the equivalence classes $[\mathfrak{A}, X]^\ell$ are definable, for instance in the extension of FO^{hg} or GF by nonemptiness NE by formulae

$$\chi_{\mathfrak{A}, X}^\ell(\bar{x}) = \bigvee_{s \in X} (\text{NE} \wedge \chi_{\mathfrak{A}, s}^\ell(\bar{x})) \equiv \bigvee_{s \in X} (\text{non} \perp \wedge \chi_{\mathfrak{A}, s}^\ell(\bar{x}))$$

derived from the classical characteristic formulae $\chi_{\mathfrak{A}, s}^\ell$. In order to define the union of (finitely many) such classes, however, we need to invoke the strong intuitionistic disjunction \otimes at the propositional level, as ordinary team disjunction would allow to mix team constituents (corresponding to unions of teams rather than an alternative between them). Now $\text{FO}(\text{non})^{\text{hg}}$ and $\text{GF}(\text{non})$ are invariant under $\sim_{\mathbf{g}}$ by Proposition 11, and sufficiently expressive for the $\chi_{\mathfrak{A}, s}^\ell(\bar{x})$ as well as to express nonemptiness ($\text{NE} \equiv \text{non} \perp$) and intuitionistic disjunction ($\varphi_1 \otimes \varphi_2 \equiv \text{non}(\text{non} \varphi_1 \wedge \text{non} \varphi_2)$). If \mathcal{C} is a class of team configurations \mathfrak{A}, X composed of τ -structures \mathfrak{A} for fixed finite τ , with teams $X \in \mathbb{H}(\mathfrak{A})$, which is $\sim_{\mathbf{g}}^\ell$ -closed, then $\mathcal{C} = \{ \mathfrak{B}, Y : \mathfrak{B} \models_Y \otimes_{\mathfrak{A}, X \in \mathcal{C}} \bigvee_{s \in X} (\text{NE} \wedge \chi_{\mathfrak{A}, s}^\ell(\bar{x})) \}$, and in terms of FO^T or GF^T ,

$$\mathcal{C} = \left\{ \mathfrak{B}, Y : (\mathfrak{B}, \llbracket Y \rrbracket) \models \bigvee_{\mathfrak{A}, X \in \mathcal{C}} \left(\begin{array}{l} \bigwedge_{s \in X} \exists \bar{x} (T\bar{x} \wedge \chi_{\mathfrak{A}, s}^\ell(\bar{x})) \\ \wedge \forall \bar{x} (T\bar{x} \rightarrow \bigvee_{s \in X} \chi_{\mathfrak{A}, s}^\ell(\bar{x})) \end{array} \right) \right\}.$$

► **Corollary 19** (Second Characterisation Theorem for Guarded Team Semantics).

$$\text{FO}^T / \sim_{\mathbf{g}} \equiv \text{GF}^T / \sim_{\mathbf{g}} \equiv \text{FO}^{\text{hg}}(\text{non}) \equiv \text{GF}(\text{non}) \equiv \text{FO}(\text{non}) / \sim_{\mathbf{g}}.$$

It is tempting to assume that this is also equivalent to GF^T , i.e. to sentences $\psi(T) \in \text{GF}(\tau \dot{\cup} \{T\})$ (with Tarski semantics). But this is not the case. As we shall see below, GF^T is only $\approx_{\mathbf{g}}$ -invariant, and not $\sim_{\mathbf{g}}$ -invariant.

7 Invariance under strong guarded team bisimulation

Recall that the strong guarded bisimulation equivalence $\mathfrak{A}, X \approx_{\mathbf{g}}^\ell \mathfrak{B}, Y$ is based, by definition, on ordinary guarded bisimulation equivalence between the expansions of the underlying τ -structures by the relational encodings of the teams, $(\mathfrak{A}, \llbracket X \rrbracket) \sim_{\mathbf{g}} (\mathfrak{B}, \llbracket Y \rrbracket)$. It is therefore clear that $\text{GF}^T(\tau)$, the classical guarded fragment applied to these same expansions, is preserved under $\approx_{\mathbf{g}}$. The classical characterisation theorem of Andr eka, van Benthem and N emeti as well as its finite model theory analogue from [24] can thus be phrased as follows.

► **Proposition 20.** $\text{FO}^T / \approx_{\mathbf{g}} \equiv \text{GF}^T$ (classically and in fmt).

Note that these formalisms are more expressive than FO/\approx_g and GF , since they can obviously express invariant team properties that are neither flat nor downward- or union-closed.

Unlike its plain counterpart, strong bisimulation is a priori not a flat notion, and it is in fact strictly stronger than \sim_g . This is also witnessed by some important and familiar atomic team properties.

► **Proposition 21.** *Inclusion and exclusion dependencies (as well as their strong negations) are GF^T -definable, and thus \approx_g -invariant. However, they are not \sim_g -invariant.*

Proof. For a team X with variables x, y, \bar{z} , we have

$$\mathfrak{A} \models_X (x \subseteq y) \iff (\mathfrak{A}, X) \models \forall xy\bar{z}(Txy\bar{z} \rightarrow \exists u\bar{v}Tux\bar{v})$$

$$\mathfrak{A} \models_X (x \mid y) \iff (\mathfrak{A}, X) \models \forall xy\bar{z}(Txy\bar{z} \rightarrow \neg \exists u\bar{v}Tux\bar{v}).$$

This extends in the obvious way to general inclusion and exclusion atoms between arbitrary tuples of variables to show that these are definable in GF^T .

To prove that exclusion atoms are not \sim_g -invariant, consider a graph \mathfrak{A} with vertices a, b, c and edges (a, b) and (b, c) , and the assignments $s : (x, y) \mapsto (a, b)$ and $s' : (x, y) \mapsto (b, c)$. On the other side let \mathfrak{B} be the graph with vertices u, v, u', v', w and edges $(u, v), (u', v'), (v, w), (v', w)$ with assignments $t : (x, y) \mapsto (u, v)$ and $t' : (x, y) \mapsto (v', w)$. Clearly $\mathfrak{A}, s \sim_g \mathfrak{B}, t$ and $\mathfrak{A}, s' \sim_g \mathfrak{B}, t'$. For the teams $X = \{s, s'\}$ and $Y = \{t, t'\}$ we thus have that $\mathfrak{A}, X \sim_g \mathfrak{B}, Y$. However, $\mathfrak{B} \models_Y (x \mid y)$ but $\mathfrak{A} \not\models (x \mid y)$. Notice, however, that $\mathfrak{A}, X \not\approx_g \mathfrak{B}, Y$, and indeed, even $\mathfrak{A}, X \not\approx_g^2 \mathfrak{B}, Y$ fails as the second player has no valid response if the first player makes a move in $(\mathfrak{A}, \llbracket X \rrbracket)$ from the assignment $s : (x, y) \mapsto (a, b) \in X$ to $s'' : (y, z) \mapsto (b, c)$. An almost identical argument applies to inclusion atoms. ◀

On the other side, the two notions of bisimulation invariance coincide as far as flat team properties are concerned. To prove this, we use the notion of guarded tree decompositions cf. [14, 23, 24], which are available in the tree unravellings induced by guarded bisimulations.

► **Proposition 22.** *If \mathfrak{A} and \mathfrak{B} are guarded tree-decomposable, then, for any two guarded assignments s and t , we have $\mathfrak{A}, s \sim_g \mathfrak{B}, t \Rightarrow \mathfrak{A}, \{s\} \approx_g \mathfrak{B}, \{t\}$.*

Proof. The second player can use a strategy whose trace in the underlying tree-like transition system of guarded configurations (induced by the guarded tree decompositions, which are themselves linked by a bisimulation), respects distances from the roots $s(\bar{x})$ and $t(\bar{x})$. ◀

► **Proposition 23.** *Properties of guarded teams that are flat and \approx_g -invariant are in fact also \sim_g -invariant.*

Proof. Assume that φ defines a flat property of guarded teams that is preserved under \approx_g , and let $\mathfrak{A}, X \sim_g \mathfrak{B}, Y$. We need to show that $\mathfrak{A} \models_X \varphi$ if, and only if, $\mathfrak{B} \models_Y \varphi$. Assume towards a contradiction that $\mathfrak{A} \models_X \varphi$ while $\mathfrak{B} \not\models_Y \varphi$. Consider the guarded tree unfoldings $(\mathfrak{A}^*, \llbracket X \rrbracket^*)$ and $(\mathfrak{B}^*, \llbracket Y \rrbracket^*)$ of $(\mathfrak{A}, \llbracket X \rrbracket)$ and $(\mathfrak{B}, \llbracket Y \rrbracket)$, respectively. Clearly

$$(\mathfrak{A}^*, \llbracket X \rrbracket^*) \sim_g (\mathfrak{A}, \llbracket X \rrbracket) \quad \text{and} \quad (\mathfrak{B}^*, \llbracket Y \rrbracket^*) \sim_g (\mathfrak{B}, \llbracket Y \rrbracket)$$

imply that $\mathfrak{A}^*, X^* \approx_g \mathfrak{A}, X$ and $\mathfrak{B}^*, Y^* \approx_g \mathfrak{B}, Y$ for the associated ‘unfolded teams’ X^* and Y^* . By \approx_g -invariance, therefore, $\mathfrak{A}^* \models_{X^*} \varphi$ and $\mathfrak{B}^* \not\models_{Y^*} \varphi$. By flatness of φ this implies on one hand that $\mathfrak{B}^* \not\models_{\{t\}} \varphi$ for some $t \in Y^*$. On the other hand there is some $s \in X^*$ such that $\mathfrak{A}^*, s \sim_g \mathfrak{B}^*, t$ for which, also by flatness, $\mathfrak{A}^* \models_{\{s\}} \varphi$. So $\mathfrak{A}^*, s \sim_g \mathfrak{B}^*, t$ while $\mathfrak{A}^* \models_{\{s\}} \varphi$ but $\mathfrak{B}^* \not\models_{\{t\}} \varphi$. In view of Proposition 22, this contradicts \approx_g -invariance of φ . ◀

► **Corollary 24.** $\text{FO}/\sim_{\mathbf{g}} \equiv \text{FO}/\approx_{\mathbf{g}}$ (classically, open in *fmt*).

On the other hand, the interplay of $\approx_{\mathbf{g}}$ - or $\approx_{\mathbf{g}}^{\ell}$ -invariance with team semantic constructs in stronger logics than FO is far less clear-cut. Clearly (team) conjunction and strong negation preserve $\approx_{\mathbf{g}}$ -invariance. However, the following example shows that $\approx_{\mathbf{g}}$ -invariance is not compatible with team disjunction, not even for atomic team properties.

► **Proposition 25.** *The formula $(x \mid y) \vee (x \mid y)$ is not $\approx_{\mathbf{g}}$ -invariant.*

Proof. Let C_n be the directed cycle of length n and let X_n be the team of all its edges. The formula $(x \mid y) \vee (x \mid y)$ says that the team can be split in a bipartite manner. We thus have that $C_n \models_{X_n} (x \mid y) \vee (x \mid y)$ if, and only if, n is even. On the other side, the guarded assignments on graphs are just singletons, edges, and inverse edges, so obviously, $(C_n, X_n) \approx_{\mathbf{g}} (C_m, X_m)$ for all $m, n > 2$. ◀

Together with Proposition 21, this example shows that GF^T is in particular not closed under team disjunction.

From the characterisation theorem for guarded fixed-point logic μGF in [14] it follows that $\mu\text{GF}^T \equiv \text{GSO}/\approx_{\mathbf{g}}$, for guarded second-order logic GSO. The question arises whether we can also obtain a characterisation theorem that relates guarded inclusion logic with (a fragment of) guarded fixed-point logic. From Proposition 7 we conclude that $\text{FO}^{\text{hg}}(\subseteq) \equiv \text{GF}(\subseteq)$, and this can be translated, by [10], into sentences in μGF^T of the form $\forall \bar{x}(T\bar{x} \rightarrow \psi(T, \bar{x}))$ where $\psi(T, \bar{x})$ has only greatest fixed points and is positive in T .

Question: Is this fragment equivalent with $\text{GF}(\subseteq)$ and/or $\text{FO}(\subseteq)/\approx_{\mathbf{g}}$?

References

- 1 S. Abramsky, J. Kontinen, J. Väänänen, and H. Vollmer, editors. *Dependence Logic. Theory and Applications*. Birkhäuser, 2016.
- 2 H. Andréka, J. van Benthem, and I. Németi. Modal Languages and Bounded Fragments of Predicate Logic. *Journal of Philosophical Logic*, 27:217–274, 1998.
- 3 V. Bárány, G. Gottlob, and M. Otto. Querying the Guarded Fragment. In *Proceedings of the 2010 25th Annual IEEE Symposium on Logic in Computer Science, LICS '10*, pages 1–10, 2010.
- 4 V. Bárány, B. ten Cate, and L. Segoufin. Guarded Negation. *J. ACM*, 62(3):22:1–22:26, 2015.
- 5 M. Benedikt, P. Bourhis, and M. Vanden Boom. Characterizing Definability in Decidable Fixpoint Logics. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, pages 107:1–107:14, 2017.
- 6 P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.
- 7 H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 2nd edition, 1999.
- 8 H.-D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Springer, 1994.
- 9 P. Galliani. Inclusion and exclusion in team semantics — On some logics of imperfect information. *Annals of Pure and Applied Logic*, 163:68–84, 2012.
- 10 P. Galliani and L. Hella. Inclusion logic and fixed-point logic. In *Computer Science Logic 2013*, pages 281–295, 2013.
- 11 V. Goranko and M. Otto. Model Theory of Modal Logic. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *Handbook of Modal Logic*, pages 249–329. Elsevier, 2007.
- 12 E. Grädel. On the restraining power of guards. *Journal of Symbolic Logic*, 64:1719–1742, 1999.
- 13 E. Grädel. Guarded fixed point logics and the monadic theory of countable trees. *Theoretical Computer Science*, 288:129–152, 2002.

- 14 E. Grädel, C. Hirsch, and M. Otto. Back and forth between guarded and modal logics. *ACM Transactions on Computational Logics*, 3:418–463, 2002.
- 15 E. Grädel and M. Otto. The Freedoms of (Guarded) Bisimulation. In *Trends in Logic: Johan van Benthem on Logical and Informational Dynamics*, pages 3–31. Springer, 2014.
- 16 E. Grädel and J. Väänänen. Dependence and Independence. *Studia Logica*, 101(2):399–410, 2013.
- 17 E. Grädel and I. Walukiewicz. Guarded fixed point logic. In *Proc. 14th IEEE Symp. on Logic in Computer Science, LICS 1999*, pages 45–54, 1999.
- 18 W. Hodges. Compositional semantics for a logic of imperfect information. *Logic Journal of IGPL*, 5:539–563, 1997.
- 19 I. Hodkinson. Loosely guarded fragment of first-order logic has the finite model property. *Studia Logica*, 70:205–240, 2002.
- 20 J. Kontinen, J. Müller, H. Schnoor, and H. Vollmer. A van Benthem Theorem for Modal Team Semantics. In *CSL 2015*, pages 277–291, 2015.
- 21 J. Kontinen and J. Väänänen. On definability in dependence logic. *Journal of Logic, Language, and Information*, 18:317–241, 2009.
- 22 A. Mann, G. Sandu, and M. Sevenster. *Independence-Friendly Logic. A Game-Theoretic Approach*, volume 386 of *London Mathematical Society Lecture Notes Series*. Cambridge University Press, 2012.
- 23 M. Otto. Model theoretic methods for fragments of FO and special classes of (finite) structures. In J. Esparza, C. Michaux, and C. Steinhorn, editors, *Finite and Algorithmic Model Theory*, volume 379 of *LMS Lecture Note Series*, pages 271–341. Cambridge University Press, 2011.
- 24 M. Otto. Highly acyclic groups, hypergraph covers and the guarded fragment. *Journal of the ACM*, 59 (1), 2012.
- 25 J. Väänänen. *Dependence Logic*. Cambridge University Press, 2007.

Order-Invariant First-Order Logic over Hollow Trees

Julien Grange

ENS Paris & PSL & INRIA & CNRS, France
jgrange@di.ens.fr

Luc Segoufin

INRIA & ENS Paris & PSL, France
luc.segoufin@inria.fr

Abstract

We show that the expressive power of order-invariant first-order logic collapses to first-order logic over hollow trees. A hollow tree is an unranked ordered tree where every non leaf node has at most four adjacent nodes: two siblings (left and right) and its first and last children. In particular there is no predicate for the linear order among siblings nor for the descendant relation. Moreover only the first and last nodes of a siblinghood are linked to their parent node, and the parent-child relation cannot be completely reconstructed in first-order.

2012 ACM Subject Classification Theory of computation → Finite Model Theory

Keywords and phrases order-invariance, first-order logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.23

Related Version <https://hal.inria.fr/hal-02310749/document>

Funding ANR QUID

1 Introduction

First-order logic (FO) is a classical formalism for expressing properties over finite structures. It is the building block of many other formalisms that are highly expressive such as MSO or logics using fixpoints such as LFP. An important and desirable feature of FO, and of all its extensions mentioned above, is that it expresses only intrinsic properties of the structure, i.e. properties invariant under isomorphisms. A limitation of FO is that it cannot express some simple properties. In particular, as it cannot distinguish between nodes that are related via some automorphism, it cannot always go through all the nodes of a structure in order to perform simple tasks such as counting them.

In many scenarios, in particular in computer science, the structures under investigation are stored on a disk: this yields an implicit order among the elements of the structure. It is then reasonable to use this order within the logical formalism. In the case of FO this means adding a new binary predicate that is interpreted as a linear order. However, we want to do this in such a way that closure under isomorphisms is retained: the expressible properties should only depend on the structure and not on the way it is stored on the disk, the latter being arbitrary and subject to change. When this property is verified we say that the formula is **order-invariant** and we denote by $<-inv$ FO the set of first-order formulas that are order-invariant. We stress that being order-invariant is not a decidable property [4] hence $<-inv$ FO is not a recursive set of formulas.

Obtaining a “real” logic (in the sense of Gurevich, in particular with a recursive syntax) that has exactly the same expressive power as $<-inv$ FO is a challenging question. Solving the same question for $<-inv$ LFP would solve the longstanding quest of finding a logic for PTime as it follows from Immermann-Vardi Theorem that $<-inv$ LFP captures PTime.

In order to find a logic for $<-inv$ FO, it is useful to understand a bit better its expressive power; such is the goal of this paper.



© Julien Grange and Luc Segoufin;
licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 23; pp. 23:1–23:16



Leibniz International Proceedings in Informatics

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

An example, attributed to Gurevich, shows that $<-inv$ FO is in general strictly more expressive than FO [1]. Another key result shows that $<-inv$ FO retains the local property of FO [7]. It seems that it requires dense structures for $<-inv$ FO to express strictly more than FO. For instance when the structures are trees it has been shown that $<-inv$ FO has exactly the same expressive power than FO [4]. In [4] a “tree” is either a binary tree, where every node has at most three neighbors: its parent, its left child and its right child or, an unranked unordered tree where every node is related to its parent and all of its children, but no order is assumed among siblings.

The question of whether $<-inv$ FO = FO over any class of structures of bounded treewidth was left open in [4], where it is only shown that, over structures of bounded treewidth, $<-inv$ FO can only express properties definable in MSO.

In order to show that $<-inv$ FO collapses to FO over a class of structures of bounded treewidth, it is tempting to reduce the case of bounded treewidth to the case of trees, using tree decompositions. When trying this strategy one immediately faces two difficulties. The first one is, given two FO similar structures (in this introduction we informally say that two structures are “FO similar” if they satisfy the same FO sentences of quantifier rank k for some k sufficiently large and depending on the context), to exhibit a tree decomposition for each of them such that the resulting tree decompositions are FO similar. Once this is done, we can apply the known result over trees showing that the tree decompositions actually agree on all order-invariant properties of a given quantifier rank: they are $<-inv$ FO similar. The second difficulty is then to lift the order-invariance similarity from the tree decompositions to the original structures.

The second difficulty could be solved easily if we could interpret the original structure within its tree decomposition. Unfortunately this cannot be done in first-order (this requires reachability as an element of the structure could appear in bags arbitrarily far away within the tree decomposition). This problem can be eliminated by assuming “domino treewidth”, i.e. that an element appears in a bounded number of bags, which is equivalent to assuming bounded degree of the structure on top of bounded treewidth [5].

Even when assuming bounded degree, the first difficulty remains and we still do not know the precise expressive power of $<-inv$ FO over structures of bounded degree and pathwidth 2! This paper is an attempt toward solving the pathwidth 2 case.

We show that $<-inv$ FO collapses to FO over the class of hollow trees. Hollow trees are first-order structures with two binary relations that are interpreted so that the resulting structure is a tree with the following features: each node has at most four neighbors: its first child, its last child and possibly a left and a right sibling. One of the binary relation denotes the sibling relation while the other one denotes the partial parent-child relation. This model strictly extends the case of binary trees as a node may have arbitrarily many children. However it is less powerful than the unranked ordered model as a node is not directly related to its parent, unless it is the first or last of its children. Note that because of its locality, FO cannot reconstruct the complete parent-child relation of every node within a hollow tree (this can be done in MSO or using the transitive closure of the sibling relation).

It is not immediate to see how hollow trees are related to structures of pathwidth 2 and of bounded degree. It turns out that if in the model of hollow trees we only had one binary relation and could not distinguish between the (partial) parent-child relation and the sibling one, then we would have a model that is FO equivalent to structures of bounded degree and pathwidth 2 in the sense that there exist FO-interpretations from one to the other (as depicted in the conclusion). In particular the collapse of $<-inv$ FO to FO in one of them would imply the collapse in the other as we explain in Section 2.4. We leave the extension of our result to this class of structures as an open problem.

Our proof follows a strategy similar to the case of binary trees: we first exhibit a set of operations over hollow trees (actually over structures FO similar to hollow trees) that preserve order-invariance similarity. We then show that if two hollow trees are FO similar then one of them can be transformed using our set of operations into the other, lifting FO similarity to $<$ -inv FO similarity. The first part is standard and makes use of the locality of $<$ -inv FO [7]. The second part is more combinatorial and forms the main technical contribution of this paper.

Related work. Besides the papers already mentioned above, there exist several other publications related to our work. We will make use in our proof of the fact that $<$ -inv FO \subseteq MSO over classes of graphs of bounded treewidth, which has been initially claimed in [4]. Another proof of this result, extended to a broader class called “decomposable structures”, can be found in [6].

If testing order invariance is undecidable for FO it is decidable for its two variable fragment [13].

Several authors considered order-invariance for more expressive logics (first-order with modulo predicates [11], MSO [6]) or with more expressive numerical predicates [9, 8, 2, 12]. Our proof technique follows lines similar to [4, 11] but is mildly related to the others.

Due to space limitations many of the proofs are omitted or just sketched in this long abstract. They can be found at <https://hal.inria.fr/hal-02310749/document>

2 Preliminaries

2.1 General notations

We consider relational structures and use classical terminology for them. We use Σ to denote a relational schema and Σ -structure to denote a structure over Σ . Our structures are always finite and are denoted through calligraphic upper-case letters and their domain through the corresponding standard upper-case letter. For instance, A would denote the domain of the structure \mathcal{A} . For a relation symbol $R \in \Sigma$ and a Σ -structure \mathcal{A} , we denote by $R^{\mathcal{A}}$ the interpretation of R in \mathcal{A} .

Given a relational signature Σ , first-order logic, $\text{FO}(\Sigma)$, and monadic second-order logic, $\text{MSO}(\Sigma)$, are defined in the standard way (see, e.g., [10]). The main formalism of interest here is order-invariant first-order logic, denoted $<$ -inv $\text{FO}(\Sigma)$. A sentence φ in $\text{FO}(\Sigma \cup \{<\})$ belongs to $<$ -inv $\text{FO}(\Sigma)$ if for every Σ -structure \mathcal{A} , whether $(\mathcal{A}, <^{\mathcal{A}}) \models \varphi$ is independent of the choice of the linear order $<^{\mathcal{A}}$ on A . In that case, we write $\mathcal{A} \models \varphi$. For any $\mathcal{L} \in \{\text{FO}(\Sigma), \text{MSO}(\Sigma), <$ -inv $\text{FO}(\Sigma)\}$ and two Σ -structures \mathcal{A} and \mathcal{B} , we write $\mathcal{A} \equiv_k^{\mathcal{L}} \mathcal{B}$ to mean that \mathcal{A} and \mathcal{B} satisfy the same sentences of \mathcal{L} of quantifier rank at most k . As usual we omit Σ when it is clear from the context.

We use the standard notion of FO-interpretations in order to define a new structure from an existing one. Given a FO-interpretation \mathcal{I} , we call **arity** of \mathcal{I} the number of free variables in the formula of \mathcal{I} which defines the domain of the new structure, and **depth** of \mathcal{I} the maximum among the quantifier ranks of the formulas defining the domain and the new relations. It is a well known result that for every \mathcal{A}, \mathcal{B} , and \mathcal{I} of arity a and depth d , and for every $k \in \mathbb{N}$, if $\mathcal{A} \equiv_{ak+d}^{\mathcal{L}} \mathcal{B}$ then $\mathcal{I}(\mathcal{A}) \equiv_k^{\mathcal{L}} \mathcal{I}(\mathcal{B})$.

Let \mathcal{A} be a structure over a vocabulary containing the binary relation symbol R . We say that $U \subseteq A$ is **R -stable** if $\forall x \in U, \forall y \in A, (R(x, y) \vee R(y, x)) \rightarrow y \in U$.

23:4 Order-Invariant First-Order Logic over Hollow Trees

For a set σ of symbols, we define the vocabulary $P_\sigma := \{P_s : s \in \sigma\}$, where every P_s is a unary relation symbol.

As usual the Gaifman graph of a relational structure \mathcal{A} is the (unoriented) graph whose vertices are the elements of the domain of the structure and the edges relate two vertices that appear in the same tuple of a relation of \mathcal{A} . We denote by $\text{dist}_{\mathcal{A}}(x, y)$ the distance between x and y in the Gaifman graph of \mathcal{A} . Given two sets S and T of elements of A and $m \in \mathbb{N}$, we say that S and T are m -**distant** in \mathcal{A} , if $\text{dist}_{\mathcal{A}}(x, y) \geq m$ for all $x \in S$ and all $y \in T$. The k -**neighborhood** $\mathcal{N}_{\mathcal{A}}^k(x)$ of some $x \in A$ is the substructure of \mathcal{A} induced by $\{y \in A : \text{dist}_{\mathcal{A}}(x, y) \leq k\}$ together with an additional constant interpreted as x . The k -**type** $\text{tp}_{\mathcal{A}}^k(x)$ of x in \mathcal{A} is the isomorphism class of its k -neighborhood. We extend those definitions to tuples of elements in the usual way, fixing the tuples pointwise.

For $k \in \mathbb{N}$, we define the k -**enrichment** $\mathcal{E}_k(\mathcal{A})$ of a Σ -structure \mathcal{A} as \mathcal{A} itself where each element has been recolored with its k -type. $\mathcal{E}_k(\mathcal{A})$ is a structure over the vocabulary Σ augmented with a unary predicate for every k -type over Σ : there are a finite number of them as long as we consider classes of structures of bounded degree.

2.2 Hollow trees

An unranked ordered tree is a tree with a successor relation among the children of any node. We see unranked ordered trees as structures over the signature composed of two binary relation symbols S and S' , where S is interpreted as the parent-child relation, and S' as the horizontal successor. A set of nodes that share the same parent is called a siblinghood.

We define a mapping H from the set of unranked ordered trees to structures over two binary predicates S and E . Given an unranked ordered tree \mathcal{T} , $H(\mathcal{T})$ is defined as follows:

- its domain is T
- $H(\mathcal{T}) \models S(x, y)$ iff $\mathcal{T} \models S(x, y)$ and y is either the first or the last of its siblings
- E is interpreted as the symmetrical closure of S'

The image of H is the set of **hollow trees**, denoted \mathbb{H} . If $\mathcal{P} = H(\mathcal{T})$ then \mathcal{T} is the underlying tree structure of \mathcal{P} .

In other words, within a hollow tree, only the two children at the endpoints of a siblinghood know their parent. Notice that we do not distinguish between the first and last child, nor do we between the left and right sibling. This makes the model more general, as explained in Section 2.4. An example of hollow tree is given in the left part of Figure 1.



■ **Figure 1** An example of hollow tree (left) and of hollow quasitree (right). The dotted arrows represent S and the plain (symmetrical) lines represent E .

Given a finite alphabet σ , we define \mathbb{H}_σ , the set of **hollow trees over σ** , as the set of colored extensions of hollow trees using the vocabulary P_σ , where the interpretations of the predicates of P_σ partition the domain.

2.3 Main result

If \mathcal{C} is a class of structures, we say that $<$ -inv FO = FO over \mathcal{C} if for each property definable in $<$ -inv FO, there exists a first-order formula expressing this property over all structures of \mathcal{C} . Notice that for every σ , \mathbb{H}_σ is a class of structures of treewidth 2. Therefore $<$ -inv FO \subseteq MSO over \mathbb{H}_σ [4]. The main result we prove in this paper is:

► **Theorem 1.** *For all σ , $<$ -inv FO = FO over \mathbb{H}_σ*

We outline the proof here, and give more details in the rest of this paper.

Proof sketch. Our goal is to find some function f such that, $\forall \alpha \in \mathbb{N}, \forall \mathcal{P}, \mathcal{Q} \in \mathbb{H}_\sigma$, if $\mathcal{P} \equiv_{f(\alpha)}^{\text{FO}} \mathcal{Q}$ then $\mathcal{P} \equiv_{\alpha}^{<\text{-inv FO}} \mathcal{Q}$. This means that the equivalence relation $\equiv_{f(\alpha)}^{\text{FO}}$ refines $\equiv_{\alpha}^{<\text{-inv FO}}$. Both equivalence relations being of finite index and the former being definable in FO for every fixed α , the result follows.

To show this we fix some $\alpha \in \mathbb{N}$ and consider two hollow trees \mathcal{P} and \mathcal{Q} , such that $\mathcal{P} \equiv_{f(\alpha)}^{\text{FO}} \mathcal{Q}$ for a large enough $f(\alpha)$. The general idea is to modify \mathcal{Q} through some operations that are invisible to all formulas of $<$ -inv FO of quantifier rank less than α , until we reach \mathcal{P} . This will ensure that $\mathcal{P} \equiv_{\alpha}^{<\text{-inv FO}} \mathcal{Q}$.

We will use two kinds of operations as described in Section 3: “swap operations”, which preserve $<$ -inv FO, and one which preserves MSO (and *a fortiori* $<$ -inv FO as $<$ -inv FO \subseteq MSO over \mathbb{H}_σ by [4]).

The MSO-preserving operation will be used in Section 3.3, in order to pump \mathcal{Q} to make sure that every neighborhood type is present at least as many times in \mathcal{Q} as in \mathcal{P} .

Once this is done, we explain in Section 4 how to transform \mathcal{Q} with swap operations in order to include \mathcal{P} into it. Since \mathcal{Q} may be larger than \mathcal{P} , there could be some extra material in \mathcal{Q} that we call “loops”. The last step is to remove those loops and this is the goal of Section 6.

When performing the swap operations, there will be a constant need for reorganizing the S -edges (in particular to make sure that the loops are S -stable). Section 5 and Section 6.3 compile the results that allow us to do so. ◀

2.4 Bi-FO-interpretations and corollaries

Before we give more details about the proof of our main result, we recall in this section a classical tool for reducing the collapse of $<$ -inv FO to FO from one class of structures to another. We then state a few corollaries of Theorem 1.

Let $\mathcal{C}_1, \mathcal{C}_2$ be two classes of structures over the respective vocabularies τ_1 and τ_2 .

We say that \mathcal{C}_1 is **bi-FO-interpretable** through \mathcal{C}_2 if there exist two FO-interpretations \mathcal{I}_{12} and \mathcal{I}_{21} , respectively from τ_1 to τ_2 , and from τ_2 to τ_1 , such that for every $\mathcal{A} \in \mathcal{C}_1$, $\mathcal{I}_{12}(\mathcal{A}) \in \mathcal{C}_2$ and $\mathcal{I}_{21}(\mathcal{I}_{12}(\mathcal{A})) \simeq \mathcal{A}$, where \simeq denotes the existence of an isomorphism between two structures. The following result is rather straightforward:

► **Lemma 2.** *If \mathcal{C}_1 is bi-FO-interpretable through \mathcal{C}_2 and $<$ -inv FO = FO over \mathcal{C}_2 , then $<$ -inv FO = FO over \mathcal{C}_1*

Recall that in the definition of hollow trees the relation E is symmetric. This turns out to be more general than choosing E as an arbitrary directed binary relation as shown in the following result where a **directed hollow tree** is defined as for hollow trees but with a directed binary relation E . Note that we do not assume that E is a successor relation among siblings, the direction of E could be arbitrary, but the result below works in particular when E is a successor relation. Via a simple bi-FO-interpretation which uses extra colors to encode the direction of the edges, we get the following result:

► **Corollary 3.** *For every σ , $<$ -inv FO = FO on the class of σ directed hollow trees*

Define a path over σ as a word over the alphabet σ , where the successor edges are symmetrical (the argument used in the proof of Corollary 3 guarantees that paths are a more general model than words). The class of paths over σ is obviously bi-FO-interpretable through \mathbb{H}_σ : just add a S -parent to the endpoints of the path, and then forget about it. Thus we get:

► **Corollary 4.** *For every alphabet σ , $<$ -inv FO = FO on the class of paths over σ .*

Similarly, a straightforward bi-FO-interpretation together with Theorem 1 give us back the result from [4] that $<$ -inv FO = FO on ranked trees.

3 Swaps and pumping

In this section we provide a few operations, denoted swaps, that preserve $\equiv_k^{<-inv \text{ FO}}$. Although the k -type of every element will be left unchanged, applying these operations may break the somewhat rigid structure of hollow trees. In order to work with the intermediate structures, we loosen the definition of hollow trees and define hollow quasitrees as follows:

► **Definition 5.** *For $k > 0$ and σ a set of colors, we define the set of **hollow k -quasitrees** on σ , $\text{quasi-}\mathbb{H}_\sigma^k$, as the set of all finite structures over $\{E, S\} \cup P_\sigma$ such that the k -type of any of their elements is the k -type of some element in some hollow tree in \mathbb{H}_σ , and which are such that their relation E is acyclic.*

In other words a hollow quasitree differs from a hollow tree by its relation S which may not induce a tree structure: a node may have its S -children in two distinct siblinghoods and a hollow quasitree may have cycles using the relation S (but not using only the relation E). Note that by definition $\mathbb{H}_\sigma \subseteq \text{quasi-}\mathbb{H}_\sigma^k$ for every k . An example of what a hollow quasitree could look like is given in the right part of Figure 1. Note that locally, it looks like a hollow tree.

Let $\mathcal{T} \in \text{quasi-}\mathbb{H}_\sigma^k$. We define the **support** of \mathcal{T} as its restriction to the vocabulary $P_\sigma \cup \{E\}$. The n -**enriched support** of \mathcal{T} , denoted $\text{Supp}_n(\mathcal{T})$, is the support of its n -enrichment (and not the other way around). Hence, it keeps in memory the local behavior within \mathcal{T} . The set $\text{End}(\mathcal{T})$ of **endpoints** of \mathcal{T} is the set of elements of the support having degree one. A connected component of the support of \mathcal{T} is called a **thread**¹. Note that by E -acyclicity of \mathcal{T} , each of its threads is a path, hence contains exactly two endpoints. We say that a hollow k -quasitree has the **matching endpoints property** if the two endpoints of each thread have the same S -parent. Note that a hollow tree has the matching endpoints property. Notice also that in a hollow k -quasitree, any thread of length less than $2k + 1$ has matching endpoints. For $x, y \in T$ belonging to the same thread, $[x, y]$ denotes the set of elements that lie between them (formally, those who disconnect x from y in $\text{Supp}_0(\mathcal{T})$), including x and y . We naturally define $[x, y[$ as $[x, y] \setminus \{y\}$.

The following lemma, implicit in the proof of locality of $<$ -inv FO by Grohe and Schwentick [7], will allow us to prove that our operations preserve order-invariance equivalence:

► **Lemma 6.** *Let Σ be a relational vocabulary and let $p, \alpha \in \mathbb{N}$. There exists $o_p^\Sigma(\alpha) \in \mathbb{N}$ such that for every structure \mathcal{A} over Σ , and for every p -tuples of elements $\bar{a}, \bar{b} \in A^p$ that have the same $o_p^\Sigma(\alpha)$ -type in \mathcal{A} , there are two orders $<_{\bar{a}\bar{b}}$ and $<_{\bar{b}\bar{a}}$ on A such that*

¹ A thread is nothing other than a siblinghood when the quasitree is a tree.

- $(\mathcal{A}, <_{\bar{a}\bar{b}}) \equiv_{\alpha}^{\text{FO}} (\mathcal{A}, <_{\bar{b}\bar{a}})$
- $\bar{a}\bar{b}$ is an initial segment of $<_{\bar{a}\bar{b}}$
- $\bar{b}\bar{a}$ is an initial segment of $<_{\bar{b}\bar{a}}$

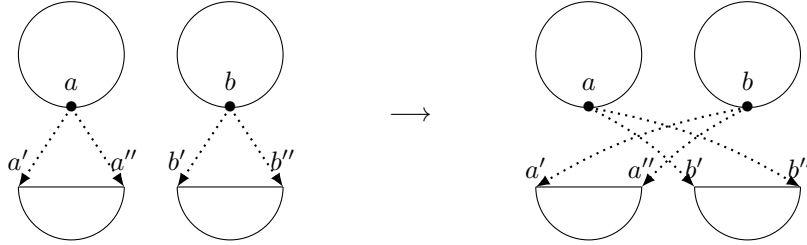
Our operations are divided into three families depending on whether we modify the relation S , the relation E , or whether we do a global pumping,

In the following, \mathcal{R} is a hollow $(m + 1)$ -quasitree on σ .

3.1 crossing- S -swaps

Let $a, a', a'', b, b', b'' \in R$ be such that $S(a, a'), S(a, a''), S(b, b'), S(b, b'')$ and such that $\text{tp}_{\mathcal{R}}^m(a, a', a'') = \text{tp}_{\mathcal{R}}^m(b, b', b'')$. Let $\mathcal{R}^- := \mathcal{R} \setminus \{S(a, a'), S(a, a''), S(b, b'), S(b, b'')\}$ and assume that the sets $\{a', a''\}, \{b', b''\}$ and $\{a, b\}$ are pairwise $(2m + 3)$ -distant in \mathcal{R}^- .

Then $\mathcal{R}' := \mathcal{R}^- \cup \{S(a, b'), S(a, b''), S(b, a'), S(b, a'')\}$ is called the m -guarded crossing- S -swap between a and b in \mathcal{R} (see Figure 2).



■ **Figure 2** The crossing- S -swap between a and b .

► **Note 7.** A particular case where the distance condition is met is when $\text{dist}_{\mathcal{R}}(a, b) \geq 2m + 5$.

► **Lemma 8.** For all $\alpha \in \mathbb{N}$ there exists $s(\alpha) \in \mathbb{N}$ such that for all $m \geq s(\alpha)$, and every hollow $(m + 1)$ -quasitree \mathcal{R} ,

if \mathcal{R}' is the m -guarded crossing- S -swap between a and b in \mathcal{R} ,

then $\mathcal{R}' \equiv_{\alpha}^{\leq\text{-inv FO}} \mathcal{R}$, and $\forall x \in R, \text{tp}_{\mathcal{R}'}^{m+1}(x) = \text{tp}_{\mathcal{R}}^{m+1}(x)$. Moreover $\mathcal{R}' \in \text{quasi-}\mathbb{H}_{\sigma}^{m+1}$ and $\text{Supp}_{m+1}(\mathcal{R}') = \text{Supp}_{m+1}(\mathcal{R})$.

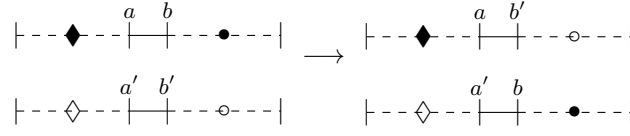
Proof sketch. In order to prove that $\mathcal{R}' \equiv_{\alpha}^{\leq\text{-inv FO}} \mathcal{R}$ we need to exhibit a linear order over \mathcal{R} and one over \mathcal{R}' such that we can play an α -round Ehrenfeucht-Fraïssé game between the resulting ordered structures. The linear orders are constructed using Lemma 6 applied to (a', a'') and (b', b'') and the structure \mathcal{R}^- . A simple FO-interpretation is then used to transfer the corresponding orders onto \mathcal{R} and \mathcal{R}' . Proving that the type of an element is unchanged is straightforward. ◀

3.2 E -swaps

We define four different kinds of E -swaps.

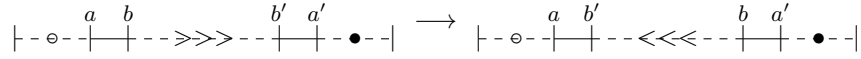
Let $a, b, a', b' \in R$ be such that $E(a, b), E(a', b')$, a, b and a', b' appear in two different threads of \mathcal{R} and such that $\{a, b, a', b'\}$ and $\text{End}(\mathcal{R})$ are $(2m + 3)$ -distant in $\text{Supp}_0(\mathcal{R})$. Furthermore, assume that $\text{tp}_{\mathcal{R}}^m(a, b) = \text{tp}_{\mathcal{R}}^m(a', b')$. Let $\mathcal{R}' := \mathcal{R} \setminus \{E(a, b), E(a', b')\} \cup \{E(a, b'), E(a', b)\}$.

Then \mathcal{R}' is called the m -guarded crossing- E -swap between ab and $a'b'$ in \mathcal{R} (c.f. Figure 3).



■ **Figure 3** Illustration of the m -guarded crossing- E -swap between ab and $a'b'$ in \mathcal{R} .

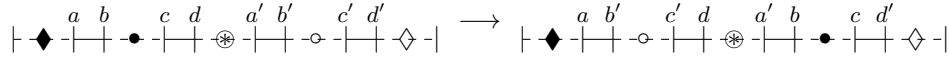
Let $a, b, b', a' \in R$ appear in that order in a single thread of \mathcal{R} , such that $E(a, b), E(a', b')$, and such that $\{a, b, a', b'\}$ and $\text{End}(\mathcal{R})$ are $(2m + 3)$ -distant in $\text{Supp}_0(\mathcal{R})$. Furthermore, assume that $\text{tp}_{\mathcal{R}}^m(a, b) = \text{tp}_{\mathcal{R}}^m(a', b')$. Let $\mathcal{R}' := \mathcal{R} \setminus \{E(a, b), E(a', b')\} \cup \{E(a, b'), E(a', b)\}$. Then \mathcal{R}' is called the m -guarded **mirror- E -swap at $[b, b']$ in \mathcal{R}** (c.f. Figure 4).



■ **Figure 4** Illustration of the m -guarded mirror- E -swap at $[b, b']$ in \mathcal{R} .

Consider now $a, b, c, d, a', b', c', d' \in R$ appearing in that order in a single thread of \mathcal{R} such that $E(a, b), E(c, d), E(a', b'), E(c', d')$ and such that $\{a, b, c, d, a', b', c', d'\}$ and $\text{End}(\mathcal{R})$ are $(2m + 3)$ -distant in $\text{Supp}_0(\mathcal{R})$. Furthermore, assume that $\text{tp}_{\mathcal{R}}^m(a, b) = \text{tp}_{\mathcal{R}}^m(a', b')$ and $\text{tp}_{\mathcal{R}}^m(c, d) = \text{tp}_{\mathcal{R}}^m(c', d')$.

Let $\mathcal{R}' := \mathcal{R} \setminus \{E(a, b), E(a', b'), E(c, d), E(c', d')\} \cup \{E(a, b'), E(a', b), E(c, d'), E(c', d)\}$. \mathcal{R}' is called the m -guarded **segment- E -swap between $[b, c]$ and $[b', c']$ in \mathcal{R}** (c.f. Figure 5).



■ **Figure 5** Illustration of the m -guarded segment- E -swap between $[b, c]$ and $[b', c']$ in \mathcal{R} .

Finally, let a, b, a', b', a'', b'' be elements of R appearing in that order in a single thread of \mathcal{R} , such that $E(a, b), E(a', b')$ and $E(a'', b'')$ and $\{a, b, a', b', a'', b''\}$ and $\text{End}(\mathcal{R})$ are $(2m + 3)$ -distant in $\text{Supp}_0(\mathcal{R})$. Furthermore, suppose that $\text{tp}_{\mathcal{R}}^m(a, b) = \text{tp}_{\mathcal{R}}^m(a', b') = \text{tp}_{\mathcal{R}}^m(a'', b'')$.

Let $\mathcal{R}' := \mathcal{R} \setminus \{E(a, b), E(a', b'), E(a'', b'')\} \cup \{E(a, b'), E(a', b''), E(a'', b)\}$. \mathcal{R}' is called the m -guarded **contiguous-segment- E -swap between $[b, a']$ and $[b', a'']$ in \mathcal{R}** (c.f. Figure 6).

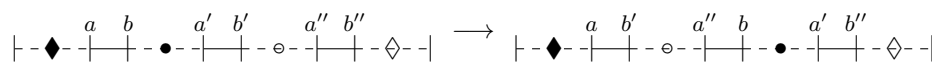
As long as m is large enough, all the m -guarded E -swaps preserve $\equiv_{\alpha}^{< \text{inv FO}}$ and the $(m + 1)$ -type of every element:

► **Lemma 9.** *For all $\alpha \in \mathbb{N}$ there exists $s(\alpha) \in \mathbb{N}$ such that for every $m \geq s(\alpha)$ and every hollow $(m + 1)$ -quasitree \mathcal{R} , if \mathcal{R}' is either*

- *the m -guarded crossing- E -swap between ab and $a'b'$ in \mathcal{R}*
- *the m -guarded mirror- E -swap at $[b, b']$ in \mathcal{R}*
- *the m -guarded contiguous-segment- E -swap between $[b, a']$ and $[b', a'']$ in \mathcal{R}*
- *the m -guarded segment- E -swap between $[b, c]$ and $[b', c']$ in \mathcal{R}*

then $\mathcal{R}' \equiv_{\alpha}^{< \text{inv FO}} \mathcal{R}$, $\forall x \in R$, $\text{tp}_{\mathcal{R}'}^{m+1}(x) = \text{tp}_{\mathcal{R}}^{m+1}(x)$ and $\mathcal{R}' \in \text{quasi-}\mathbb{H}_{\sigma}^{m+1}$.

Proof sketch. The proof is a tedious case analysis. Basically it amounts to the following idea: if the elements involved in the swap are far away from each other then we can use Lemma 6 in the structure \mathcal{R} minus the E -edges of interest, and get orders on \mathcal{R} and \mathcal{R}' which make these structures similar as in the proof of Lemma 8.



■ **Figure 6** Illustration of the m -guarded contiguous-segment- E -swap between $[b, a']$ and $[b', a'']$ in \mathcal{R} .

On the other hand, if the elements are close to each other, then the fact that they share the same type induces some periodicity on their neighborhoods. These neighborhoods can therefore be decomposed into several consecutive similar pieces. We can then apply Lemma 6 to these smaller components to conclude. ◀

3.3 Pumping

The next operation makes use of the fact that $<$ -inv FO \subseteq MSO over hollow trees. Hence our hollow trees can be “pumped” in order to duplicate some of their parts.

Given a structure \mathcal{A} and a k -type τ , we denote by $|\mathcal{A}|_\tau$ the number of elements of \mathcal{A} whose k -type is τ . We will essentially use 0-types as our structures will be enriched by recoloring each element by its k -type. In view of this we denote by $\llbracket \mathcal{A} \rrbracket$ the function $\tau \mapsto |\mathcal{A}|_\tau$ whose domain is the set of 0-types over the considered vocabulary.

Let $d, D \in \mathbb{N}$, and f, g be functions from a same domain to \mathbb{N} . We say that $f \leq_d^D g$ if for every x in the domain:

- if $f(x) \leq d$, then $f(x) = g(x)$
- if $f(x) \neq g(x)$, then $g(x) \geq f(x) + D$

By $f < g$, we mean that $\forall x, f(x) < g(x)$ or $f(x) = g(x) = 0$.

In the following proposition $<$ -inv FO can be replaced by MSO.

▶ **Proposition 10.** $\forall \alpha, n, d \in \mathbb{N}, \exists M \in \mathbb{N}, \forall D \in \mathbb{N}, \forall \mathcal{P}, \mathcal{Q} \in \mathbb{H}_\sigma$, if $\mathcal{P} \equiv_M^{\text{FO}} \mathcal{Q}$, then there exists $\mathcal{Q}' \in \mathbb{H}_\sigma$ such that $\mathcal{Q}' \equiv_\alpha^{< \text{-inv FO}} \mathcal{Q}$ and $\llbracket \mathcal{E}_{n+1}(\mathcal{P}) \rrbracket \leq_d^D \llbracket \mathcal{E}_{n+1}(\mathcal{Q}') \rrbracket$.

Proof sketch. This is a pumping argument: by setting M large enough, we make sure in FO that if a $(n + 1)$ -type has more occurrences in \mathcal{P} than in \mathcal{Q} , then it has enough occurrences in \mathcal{Q} so that we can find a context in \mathcal{Q} containing at least one occurrence, and no occurrence of a rare type, such that we can duplicate this context inside \mathcal{Q} without changing its MSO-type. ◀

4 Inclusion and pseudo-inclusion

Recall that our ultimate goal is to show that if two hollow trees agree on the same FO sentences of quantifier rank $f(\alpha)$ then they agree on all $<$ -inv FO sentences of quantifier rank α . For this, we will show that if \mathcal{P} and \mathcal{Q} are hollow trees that agree on all FO sentences of quantifier rank $f(\alpha)$ then we can use operations such as the swap operations described in Section 3 to transform \mathcal{Q} into \mathcal{P} . As these operations preserve $<$ -inv FO we get the desired result.

In this section we perform the first step towards transforming \mathcal{Q} into \mathcal{P} . We show that using the swap operations we can transform \mathcal{Q} into \mathcal{Q}' so that \mathcal{Q}' “includes” \mathcal{P} . The resulting structure \mathcal{Q}' will be a hollow quasitree. In the next sections we will continue the transformation and remove from \mathcal{Q}' all the extra material it contains, deriving \mathcal{P} .

In order to define what we mean by “inclusion” we need the notion of a n -abstract context of a hollow quasitree. Intuitively this is a S -stable n -enriched substructure. More formally, given a hollow quasitree $\mathcal{T} \in \text{quasi-}\mathbb{H}_\sigma^n$ and a set U of its domain that is S -stable, then $\mathcal{C} := \mathcal{T}|_U$, together with the function $\text{tp}^n(\cdot)$ that maps $x \in U$ to its n -type in \mathcal{T} , is called a **n -abstract context** denoted $\mathcal{C} = \text{Ctx}_n(\mathcal{T}|_U)$. The set of n -abstract contexts is denoted Ctx_σ^n . Note that $\text{tp}^n(x)$ denotes $\text{tp}_\sigma^n(x)$ and not $\text{tp}_\sigma^n(x)$. We need to remember, at least locally, how \mathcal{C} was glued to the rest of \mathcal{T} in order to preserve n -types when moving \mathcal{C} to some other place.

We are now ready to define the notion of “inclusion”. We actually define both “inclusions” and “pseudo-inclusions”. We will need to pseudo-include a hollow quasitree into another (Proposition 12), and then to include an abstract context into a hollow quasitree (Proposition 13). Since a hollow k -quasitree $\mathcal{T} \in \text{quasi-}\mathbb{H}_\sigma^k$ can be seen as a k -abstract context ($\mathcal{T} = \text{Ctx}_k(\mathcal{T}|_T)$), we only need to define (pseudo-)inclusions from an abstract context into a hollow quasitree.

► **Definition 11.** Let $k \in \mathbb{N}$, $\mathcal{U} \in \text{Ctx}_\sigma^k$ and $\mathcal{Q} \in \text{quasi-}\mathbb{H}_\sigma^k$. We say that $h : U \rightarrow \mathcal{Q}$ is a **k -pseudo-inclusion** if h is injective and for all $x, y, z \in U$ the following is verified:

1. $\text{tp}_\mathcal{Q}^k(h(x)) = \text{tp}^k(x)$,
2. if x and y are in the same thread of \mathcal{U} then $h(x)$ and $h(y)$ are also on the same thread of \mathcal{Q} and if moreover $z \in [x, y]$ then $h(z) \in [h(x), h(y)]$,
3. if $\mathcal{U} \models E(x, y)$ and t is the E -neighbor of $h(x)$ in $[h(x), h(y)]$ then t is the image of y by an isomorphism (induced by the fact that they share the same k -type) between the n -neighborhood of x and that of $h(x)$.

If $\mathcal{U} \models E(x, y)$ and $\mathcal{Q} \not\models E(h(x), h(y))$ then $\{x, y\}$ is said to be a **jumping pair** for h , and $\text{tp}_\mathcal{Q}^{k-1}(h(x), t)$, where t is the E -neighbor of $h(x)$ in $[h(x), h(y)]$, is called its **type**.²

A k -pseudo-inclusion is said to be **reduced** if there is at most one jumping pair of a given type.

A k -pseudo-inclusion is called a **k -inclusion** if it has no jumping pairs, that is if it preserves E .

The last condition of pseudo-inclusion is a complication induced by the fact that E is not oriented and that we thus cannot distinguish between the two siblings of a node. It ensures that h preserves the neighborhoods in the right order. We can now state the main result of this section. Note that the precondition that \mathcal{Q} has more realizations for each type than \mathcal{U} or \mathcal{P} will not be a problem in view of Proposition 10. The second proposition is stronger than the first one as it derives inclusion instead of pseudo-inclusion, but it requires the stronger hypothesis that every occurring type has strictly more realizations in \mathcal{Q} than in \mathcal{U} .

► **Proposition 12.** For every $\alpha, m \in \mathbb{N}$, there exists $N \in \mathbb{N}$ such that $\forall \mathcal{P}, \mathcal{Q} \in \text{quasi-}\mathbb{H}_\sigma^{N+1}$, if $\llbracket \mathcal{E}_{N+1}(\mathcal{P}) \rrbracket \leq \llbracket \mathcal{E}_{N+1}(\mathcal{Q}) \rrbracket$, then there exists $\mathcal{Q}' \in \text{quasi-}\mathbb{H}_\sigma^{m+1}$ such that $\mathcal{Q}' \equiv_\alpha^{<\text{inv FO}} \mathcal{Q}$, $\llbracket \mathcal{E}_{m+1}(\mathcal{Q}') \rrbracket = \llbracket \mathcal{E}_{m+1}(\mathcal{Q}) \rrbracket$ and h that is a $(m+1)$ -pseudo-inclusion from \mathcal{P} into \mathcal{Q}' .

► **Proposition 13.** For every $\alpha, m \in \mathbb{N}$, there exists $N \in \mathbb{N}$ such that $\forall \mathcal{U} \in \text{Ctx}_\sigma^{N+1}$, $\forall \mathcal{Q} \in \text{quasi-}\mathbb{H}_\sigma^{N+1}$, if $\llbracket \mathcal{E}_{N+1}(\mathcal{U}) \rrbracket < \llbracket \mathcal{E}_{N+1}(\mathcal{Q}) \rrbracket$, then there exists $\mathcal{Q}' \in \text{quasi-}\mathbb{H}_\sigma^{m+1}$ such that $\mathcal{Q}' \equiv_\alpha^{<\text{inv FO}} \mathcal{Q}$, $\llbracket \mathcal{E}_{m+1}(\mathcal{Q}') \rrbracket = \llbracket \mathcal{E}_{m+1}(\mathcal{Q}) \rrbracket$ and \mathcal{U} is $(m+1)$ -included in \mathcal{Q}' .

Proof sketch. Both propositions have a similar proof: we first prove Proposition 12, and explain afterwards how to move from pseudo-inclusions to inclusions.

² This is an ease of notation; to be more precise, we should make the type of a jumping pair symmetrical.

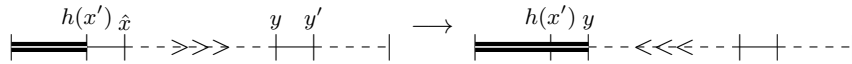
We define the pseudo-inclusion h step by step, extending the domain of h thread by thread and, inside each thread, from one of its endpoint to the other. At each step we modify \mathcal{Q} using E -swaps, if necessary.

We give a special treatment to short threads and portions of the long threads that are close to the endpoints: in that case, no modification of \mathcal{Q} is required as the cardinality precondition ensures the presence of the necessary sequences within \mathcal{Q} . We then move to the parts of the long threads that are far from the endpoints, adding them one node at a time to the domain of the pseudo-inclusion. Note that as all the elements involved in the E -swaps to come are distant from the endpoints, the E -swaps involved are guarded.

Let x' be the last node of the current thread t that has been given an image by h , and let x be the next node to which we want to extend the domain of h . By hypothesis, we know that there exists a node $y \notin \text{Im}(h)$ far from any endpoint, that has the same $(m + 1)$ -type as x . We denote by y' the neighbor of y that has the same m -type as x' , and by \hat{x} the neighbor of $h(x')$ having the same m -type as x .

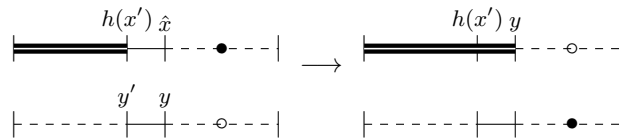
We proceed to a case analysis depending on the relative position of $y, y', h(x')$ and \hat{x} . If y', y are on the same thread as $h(x'), \hat{x}$ and in the same direction (in particular when $y = \hat{x}$), we simply set $h(x)$ to y and we are done. If not, one of the E -swaps will place y to the desired position.

For instance, if y', y are on the same thread as $h(x'), \hat{x}$ but in the reverse direction (c.f. Figure 7, where the double line represents $\text{Im}(h)$), then we consider the m -guarded mirror- E -swap at $[\hat{x}, y]$ in \mathcal{Q} and extend h by setting $h(x)$ to y .



■ **Figure 7** $h(x'), \hat{x}$ and y', y are in the same thread, but in reverse order: we use a mirror- E -swap.

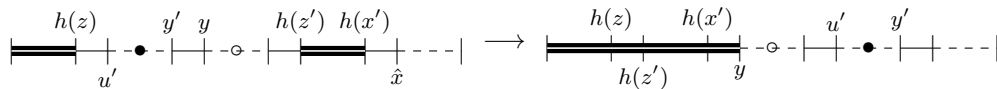
Now, if y is on a thread that does not intersect $\text{Im}(h)$ (c.f. Figure 8), we consider the m -guarded crossing- E -swap between $h(x')\hat{x}$ and $y'y$ in \mathcal{Q} , and extend h by setting $h(x)$ to y .



■ **Figure 8** y is on a thread disjoint from $\text{Im}(h)$: we use a crossing- E -swap.

If y', y are in the same direction as $h(x'), \hat{x}$, and are between $h(z)$ and $h(z')$ where z and z' are consecutive node of the current thread (c.f. Figure 9).

Then we consider the m -guarded segment- E -swap between $[u', y']$ and $[h(z'), h(x')]$ in \mathcal{Q} , and extend h by setting $h(x)$ to y .



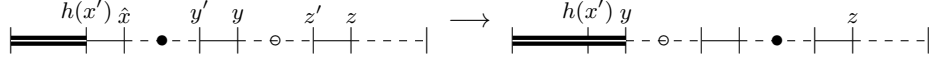
■ **Figure 9** y', y are between the images of two already included neighbors: we use a segment- E -swap.

There are a few other cases that are treated similarly. This concludes the proof for pseudo-inclusion.

23:12 Order-Invariant First-Order Logic over Hollow Trees

For Proposition 13, as we wish to construct an inclusion, we need to make sure that there is no “jump” in the mapping.

Note that among all the previously mentioned cases, only one didn’t guarantee the absence of a jump, namely when y', y are on the same thread as $h(x'), \hat{x}$ and in the right direction, but when $y \neq \hat{x}$. We then use the stronger hypothesis on the number of types in \mathcal{Q} , which guarantees that there also exist z, z' verifying the same conditions as y, y' (cf. Figure 10). We consider the m -guarded contiguous-segment- E -swap between $[\hat{x}, y']$ and $[y, z']$ in \mathcal{Q} , and extend h by setting $h(x)$ to y . h is now an inclusion.



■ **Figure 10** y', y, z', z and $h(x'), \hat{x}$ are on the same thread, in the same order: we use a contiguous-segment- E -swap to avoid a jump in the inclusion. ◀

5 Tools for reorganizing S -edges

In the previous section, we have seen how to “rewrite” \mathcal{Q} using E -swap operations in order to pseudo-include \mathcal{P} into the resulting quasitree. By definition, the pseudo-inclusion h of \mathcal{P} into \mathcal{Q} respects the enriched support but can be completely wild relatively to the S -edges. For instance, in \mathcal{Q} , the endpoints of a thread may not have the same S -parent. In this section we show how to use S -swaps in order to ensure that our pseudo-inclusion mapping takes into account (to various degrees) the S -edges. We say that two nodes of a quasitree are S -siblings if they share the same S -parent.

In Section 5.1, we show how to make sure that the pseudo-inclusion respects the S -siblings relation. In Section 5.2 we show how to ensure that the image of a pseudo-inclusion is S -stable. S -stability is required to define and operate on the loops, as will be established in Section 6.

5.1 S -siblings re-association

The following Lemma shows how to modify a pseudo-inclusion in order for it to preserve the S -siblings relation. Note that it doesn’t necessarily mean that the image structure has the matching endpoint property because the initial structure itself may not have this property as it is derived from a quasitree.

► **Lemma 14.** $\forall \alpha, m \in \mathbb{N}, \exists N \in \mathbb{N}, \forall \mathcal{W} \in \text{Ctxt}_\sigma^N, \forall \mathcal{Q} \in \text{quasi-}\mathbb{H}_\sigma^N$, if $h : \mathcal{W} \rightarrow \mathcal{Q}$ is a N -pseudo-inclusion, then there exists some $\mathcal{Q}' \in \text{quasi-}\mathbb{H}_\sigma^{m+1}$ and some $(m+1)$ -pseudo-inclusion $h' : \mathcal{W} \rightarrow \mathcal{Q}'$ such that $\mathcal{Q}' \equiv_\alpha^{\text{inv FO}} \mathcal{Q}$, $\text{Supp}_{m+1}(\mathcal{Q}') \simeq \text{Supp}_{m+1}(\mathcal{Q})$ and, if x and y are S -siblings in \mathcal{W} , then so are $h'(x)$ and $h'(y)$ in \mathcal{Q}' .

Proof sketch. We correct the S -edges two by two: let x, y be two S -siblings in \mathcal{W} such that $h(x), h(y)$ are not S -siblings in \mathcal{Q} , and let $z \in \mathcal{Q}$ be the S -sibling of $h(x)$.

z and $h(y)$ must have the same $(N-2)$ -type: we can use a crossing- E -swap or a mirror- E -swap (depending on whether they are the endpoints of a same thread) to exchange their positions and make sure $h(x)$ and $h(y)$ are S -siblings.

However, for these swaps to be guarded, we must operate far enough from the endpoints. This can be done as long as we choose N large enough. ◀

A particular case of the previous lemma is when \mathcal{W} is a hollow tree and h is surjective: then \mathcal{Q}' has the matching endpoints property. This result will be useful in the proof of Proposition 18.

5.2 S -stabilization

The image of a pseudo-inclusion has no reason to be S -stable, thus neither has its complement. However, this is a crucial requirement to apply the results presented in the next section, Section 6, in order to remove the extra material not in the image of the pseudo-inclusion.

The next result provides a method to ensure that the image (and its complement) of a pseudo-inclusion is S -stable.

Recall that a pseudo-inclusion is said to be reduced if there is at most one jumping pair of a given type. At the end of this process, we get a reduced pseudo-inclusion, which will allow us to minimize the complement of its image in Section 6.1.

► **Proposition 15.** *For every $\alpha, m \in \mathbb{N}$, there exist $N, d, D \in \mathbb{N}$ such that, for every $\mathcal{P} \in \mathbb{H}_\sigma$, $\mathcal{Q} \in \text{quasi-}\mathbb{H}_\sigma^{N+1}$ such that $[\mathcal{E}_{N+1}(\mathcal{P})] \leq_d^D [\mathcal{E}_{N+1}(\mathcal{Q})]$ and \mathcal{P} is $(N + 1)$ -pseudo-included in \mathcal{Q} through some h , there are some h' and $\mathcal{Q}' \in \text{quasi-}\mathbb{H}_\sigma^{m+1}$ such that $\mathcal{Q}' \equiv_\alpha^{<\text{inv FO}} \mathcal{Q}$, $\text{Supp}_{m+1}(\mathcal{Q}') \simeq \text{Supp}_{m+1}(\mathcal{Q})$, h' is a reduced $(m + 1)$ -pseudo-inclusion of \mathcal{P} in \mathcal{Q}' and $\mathcal{Q}' \setminus \text{Im}(h')$ is S -stable in \mathcal{Q}' .*

Proof sketch. We consider all the pairs of elements x, y which break the S -stability of $\text{Im}(h)$, i.e. such that $S(x, y)$, $x \in \text{Im}(h)$ and $y \notin \text{Im}(h)$. If there are many of them, then at least two of them are far from each other and we can apply a crossing- S -swap to correct the mapping h . We end up with a bounded number of problematic pairs that can be corrected separately. ◀

6 Removing unnecessary material

In this section we show how to remove the material in \mathcal{Q} that is not present in the image of the pseudo-inclusion of \mathcal{P} . From the previous section we can assume that the pseudo-inclusion mapping preserves the S -siblings relation and that its image is S -stable. The remaining part of \mathcal{Q} is then a union of “loops” in the sense that they connect nodes that have the same type. After defining properly the notion of loop, we will use in Section 6.1 a pumping argument in order to reduce the size of the loop to some constant while preserving $\equiv_\alpha^{<\text{inv FO}}$. In Section 6.2 we then show how to remove small loops without affecting the order-invariant equivalence class. Finally, in Section 6.3 we show that if a hollow tree and a hollow quasitree have the same enriched support, then they are $\equiv_\alpha^{<\text{inv FO}}$: this concludes the proof of Theorem 1.

We start with the definition of an abstract loop.

Let $n \in \mathbb{N}$. Let $\text{Type}_\sigma^n[2]$ denote the set of $(n - 1)$ -types for pairs over the vocabulary $P_\sigma \cup \{E, S\}$, of degree ≤ 4 . Let Σ_n be the vocabulary enriching $P_\sigma \cup \{E, S\}$ with two unary symbols J_τ^1 and J_τ^2 for every $\tau \in \text{Type}_\sigma^n[2]$.

Let h be a reduced n -pseudo-inclusion from $\mathcal{P} \in \mathbb{H}_\sigma$ to $\mathcal{Q} \in \text{quasi-}\mathbb{H}_\sigma^n$, such that $V := \mathcal{Q} \setminus \text{Im}(h)$ is S -stable.

Let \mathcal{Q}_+ be an extension of \mathcal{Q} to Σ_n obtained in the following way. Since h is reduced, for every $\tau \in \text{Type}_\sigma^n[2]$, there is at most one jumping pair of type τ . If there isn't, J_τ^1 and J_τ^2 are interpreted as the empty set. Else, let $\{x, x'\}$ be this pair, and u' (resp. u) be the E -neighbor of $h(x)$ (resp. $h(x')$) in $[h(x), h(x')]$. Interpret J_τ^1 as $\{h(x), u'\}$ and J_τ^2 as $\{h(x'), u\}$ (the assignments $x \mapsto 1$ and $x' \mapsto 2$ are arbitrary). This is illustrated on the left part of Figure 11, where the double line represents $\text{Im}(h)$. We say that \mathcal{Q}_+ is a **h -jump-extension of \mathcal{Q}** .

23:14 Order-Invariant First-Order Logic over Hollow Trees

We define $\mathcal{V}_+ = \text{Ctx}_n(\mathcal{Q}_+|_V)$ as the extension of $\text{Ctx}_n(\mathcal{Q}|_V)$ to Σ_n where every J_τ^i is defined consistently with \mathcal{Q}_+ (i.e. $\forall x \in V, \mathcal{V}_+ \models J_\tau^i(x)$ iff $\mathcal{Q}_+ \models J_\tau^i(x)$). This process is illustrated in Figure 11. \mathcal{V}_+ is called an n -**abstract loop**. Let \mathbb{L}_σ^n be the set of n -abstract loops.



■ **Figure 11** Example of a h -jump-extension \mathcal{Q}_+ of \mathcal{Q} (on the left), and its associated abstract loop \mathcal{V}_+ of support $V := \mathcal{Q} \setminus \text{Im}(h)$ (on the right).

Every Σ_n -structure will have a '+' symbol in its name. When we omit it, we mean the reduction of the structure to $P_\sigma \cup \{E, S\}$ (for instance, from $\mathcal{V}_+ \in \mathbb{L}_\sigma^n$, we get $\mathcal{V} := \text{Ctx}_n(\mathcal{Q}|_V) \in \text{Ctx}_n(\mathcal{Q}|_V)$).

6.1 Loop minimization

It will be crucial to bound the size of the loops left by a pseudo-inclusion. The following result does this using a simple pumping argument.

► **Proposition 16.** *For every $\alpha, n \in \mathbb{N}$, there exists $N \in \mathbb{N}$ such that for every $\mathcal{P} \in \mathbb{H}_\sigma$, $\mathcal{Q} \in \text{quasi-}\mathbb{H}_\sigma^n$ and reduced n -pseudo-inclusion $h : P \rightarrow \mathcal{Q}$, if $V := \mathcal{Q} \setminus \text{Im}(h)$ is S -stable then there exists some $\mathcal{Q}' \in \text{quasi-}\mathbb{H}_\sigma^n$ and a reduced n -pseudo-inclusion $h' : P \rightarrow \mathcal{Q}'$ such that $\mathcal{Q}' \equiv_\alpha^{<- \text{inv FO}} \mathcal{Q}$, $U := \mathcal{Q}' \setminus \text{Im}(h')$ is S -stable and $|U| \leq N$.*

6.2 Loop elimination

It now remains to get rid of the small loops. This is a consequence of the ‘aperiodicity’ of $<$ -inv FO: we cannot distinguish in $<$ -inv FO between k and $k + 1$ copies of the same object if k is sufficiently large. Starting from a small loop, we can use the inclusion results of Section 4 to recreate many copies of the loop within \mathcal{Q} , then, according to the following proposition, get rid of one copy using aperiodicity.

► **Proposition 17.** *$\forall \alpha \in \mathbb{N}, \exists l \in \mathbb{N}, \forall m \in \mathbb{N}, \exists n \in \mathbb{N}, \forall M \in \mathbb{N}, \exists K \in \mathbb{N}$ such that for every abstract loop $\mathcal{U}_+ \in \mathbb{L}_\sigma^{n+1}$ and every $\mathcal{Q} \in \text{quasi-}\mathbb{H}_\sigma^{n+1}$ such that $|U| \leq M$, $(l+1) \cdot \llbracket \mathcal{E}_{n+1}(\mathcal{U}) \rrbracket < \llbracket \mathcal{E}_{n+1}(\mathcal{Q}) \rrbracket$ and such that for every $(n+1)$ -type χ that occurs in \mathcal{U} , $|\mathcal{Q}|_\chi \geq K$, there exists $\mathcal{Q}' \in \text{quasi-}\mathbb{H}_\sigma^n$ such that $\mathcal{Q}' \equiv_\alpha^{<- \text{inv FO}} \mathcal{Q}$ and $\llbracket \mathcal{E}_m(\mathcal{Q}) \rrbracket = \llbracket \mathcal{E}_m(\mathcal{Q}') \rrbracket + \llbracket \mathcal{E}_m(\mathcal{U}) \rrbracket$*

Proof sketch. The proof is based on the well known result that first-order formulas of quantifier-rank k cannot distinguish between a linear order of length 2^k and a linear order of length $2^k + 1$ (see, for instance, [10]). Hence if a loop is repeated at least $2^k + 1$ times, we can eliminate one instance without changing the $\equiv_k^{<- \text{inv FO}}$ class of the structure.

First, we include many copies of the loop in \mathcal{Q} . The inclusion may not preserve S -edges: the next step is to re-associate these S -edges with crossing- S -swaps in order for these copies to be isomorphic. This is made possible by the hypothesis on the number of occurrences of types appearing in \mathcal{U} : it gives us room to make sure the crossing- S -swaps are guarded.

Once this is done, we can remove one copy in a $<$ -inv FO-indistinguishable way. ◀

6.3 S -parents re-association

We now turn to the last step of the proof of Theorem 1.

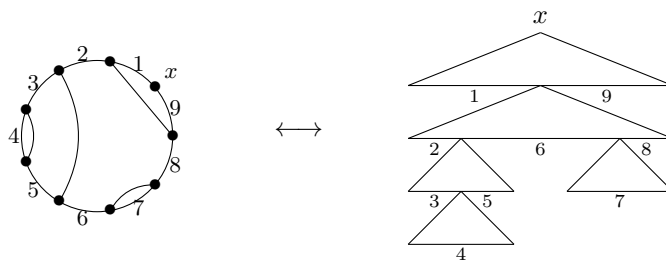
After the removal of the extra material in \mathcal{Q} , we have transformed our initial hollow tree \mathcal{Q} into a hollow quasitree having the same number of occurrences of any type as the initial \mathcal{P} . They both have the same threads but may differ with their S -edges. The following proposition states that they are $\equiv_{\alpha}^{<-inv\ FO}$, thus ending the proof of Theorem 1.

The techniques used in the proof of the following proposition are strongly reminiscent of those used in [3]; it requires a notion of vertical- S -swaps adapted to hollow trees.

► **Proposition 18.** $\forall \alpha \in \mathbb{N}$, there exists $n_1 \in \mathbb{N}$ such that $\forall \mathcal{P} \in \mathbb{H}_{\sigma}, \forall \mathcal{Q} \in \text{quasi-}\mathbb{H}_{\sigma}^{n_1}$, if $\text{Supp}_{n_1}(\mathcal{P}) \simeq \text{Supp}_{n_1}(\mathcal{Q})$ then $\mathcal{P} \equiv_{\alpha}^{<-inv\ FO} \mathcal{Q}$.

7 Conclusion

We have shown that $<-inv\ FO = FO$ over hollow trees. In order to lift this result to structures of pathwidth 2 and bounded degree, it suffices to show that $<-inv\ FO = FO$ over structures that have the same underlying graph than hollows trees, but without the possibility to distinguish a sibling from a child. In other words, there is only one binary relation that is the union of E and S . It turns out that there is a bi-FO-interpretation from structures of pathwidth 2 and bounded degree through this class of structures, as illustrated in Figure 12.



■ **Figure 12** From a typical pathwidth 2 graph of degree 3 to a hollow tree where E and S are indistinguishable.

Unfortunately our proof does not extend to this class of structures as it was crucial in our proof to distinguish between E -swaps and S -swaps. We leave this generalization as an open problem.


We also have no idea yet on what to do when the degree is not assumed to be bounded, as we are then also facing the second difficulty mentioned in the introduction, namely reinterpreting the initial structure within its tree representation.

In this paper we bypassed the first problem mentioned in the introduction, finding similar tree decompositions given similar structures, by working directly on trees. This problem seems unavoidable when working with graphs. There are examples of similar structures of treewidth 2 that do not have any similar tree decompositions of width 2. It might even be the case that for all k there are two similar structures of treewidth 2 that do not have similar tree decomposition of width k . If that were true, completely new ideas would be needed to solve the treewidth 2 case.

References

- 1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: <http://webdam.inria.fr/Alice/>.
- 2 Matthew Anderson, Dieter van Melkebeek, Nicole Schweikardt, and Luc Segoufin. Locality from Circuit Lower Bounds. *SIAM J. Comput.*, 2012. doi:10.1137/110856873.
- 3 Michael Benedikt and Luc Segoufin. Regular tree languages definable in FO and in FO_{mod}. *ACM Trans. Comput. Log.*, 2009. doi:10.1145/1614431.1614435.
- 4 Michael Benedikt and Luc Segoufin. Towards a characterization of order-invariant queries over tame graphs. *J. Symb. Log.*, 2009. doi:10.2178/jsl/1231082307.
- 5 Hans L. Bodlaender and Joost Engelfriet. Domino Treewidth. *J. Algorithms*, 1997. doi:10.1006/jagm.1996.0854.
- 6 Michael Elberfeld, Marlin Frickenschmidt, and Martin Grohe. Order Invariance on Decomposable Structures. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, 2016. doi:10.1145/2933575.2934517.
- 7 Martin Grohe and Thomas Schwentick. Locality of order-invariant first-order formulas. *ACM Trans. Comput. Log.*, 2000. doi:10.1145/343369.343386.
- 8 Frederik Harwath and Nicole Schweikardt. Regular tree languages, cardinality predicates, and addition-invariant FO. In *29th International Symposium on Theoretical Aspects of Computer Science, STACS*, 2012. doi:10.4230/LIPIcs.STACS.2012.489.
- 9 Frederik Harwath and Nicole Schweikardt. On the locality of arb-invariant first-order logic with modulo counting quantifiers. In *Computer Science Logic, CSL*, 2013. doi:10.4230/LIPIcs.CSL.2013.363.
- 10 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.
- 11 Hannu Niemistö. On Locality and Uniform Reduction. In *20th IEEE Symposium on Logic in Computer Science, LICS*, 2005. doi:10.1109/LICS.2005.32.
- 12 Nicole Schweikardt and Luc Segoufin. Addition-Invariant FO and Regularity. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS*, 2010. doi:10.1109/LICS.2010.16.
- 13 Thomas Zeume and Frederik Harwath. Order-Invariance of Two-Variable Logic is Decidable. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS*, 2016. doi:10.1145/2933575.2933594.

Glueability of Resource Proof-Structures: Inverting the Taylor Expansion

Giulio Guerrieri 

University of Bath, Department of Computer Science, Bath, UK
g.guerrieri@bath.ac.uk

Luc Pellissier 

Université de Paris, IRIF, CNRS, F-75013 Paris, France
pellissier@irif.fr

Lorenzo Tortora de Falco

Università Roma Tre, Dipartimento di Matematica e Fisica, Rome, Italy
tortora@uniroma3.it

Abstract

A Multiplicative-Exponential Linear Logic (MELL) proof-structure can be expanded into a set of resource proof-structures: its Taylor expansion. We introduce a new criterion characterizing those sets of resource proof-structures that are part of the Taylor expansion of some MELL proof-structure, through a rewriting system acting both on resource and MELL proof-structures.

2012 ACM Subject Classification Theory of computation → Linear logic

Keywords and phrases linear logic, Taylor expansion, proof-net, natural transformation

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.24

Related Version A full version [17] of the paper is available at <http://arxiv.org/abs/1910.07936>.

Funding *Giulio Guerrieri*: EPSRC grant EP/R029121/1 “Typed Lambda-Calculi with Sharing and Unsharing”

Luc Pellissier: French ANR project Rapido (ANR-14-CE35-0007)

1 Introduction

Resource λ -calculus and the Taylor expansion. Girard’s linear logic (LL, [15]) is a refinement of intuitionistic and classical logic that isolates the infinitary parts of reasoning in two (dual) modalities: the *exponentials* ! and ?. They give a logical status to the operations of memory management such as *copying* and *erasing*: a linear proof corresponds – via Curry–Howard isomorphism – to a program that uses its argument *linearly*, *i.e.* exactly once, while an exponential proof corresponds to a program that can use its argument at will.

The intuition that linear programs are analogous to linear functions (as studied in linear algebra) while exponential programs mirror a more general class of analytic functions got a technical incarnation in Ehrhard’s work [9, 10] on LL-based denotational semantics for the λ -calculus. This investigation has been then internalized in the syntax, yielding the *resource λ -calculus* [5, 11, 14]: there, copying and erasing are forbidden and replaced by the possibility to apply a function to a *bag* of resource λ -terms which specifies how many times an argument can be linearly passed to the function, so as to represent only bounded computations.

The *Taylor expansion* associates with an ordinary λ -term a (generally infinite) set of resource λ -terms, recursively approximating the usual application: the Taylor expansion of the λ -term MN is made of resource λ -terms of the form $t[u_1, \dots, u_n]$, where t is a resource λ -term in the Taylor expansion of M , and $[u_1, \dots, u_n]$ is a bag of arbitrarily finitely many (possibly 0) resource λ -terms in the Taylor expansion of N . Roughly, the idea is to decompose a program into a set of purely “resource-sensitive programs”, all of them containing only bounded (although possibly non-linear) calls to inputs. The notion of Taylor expansion has



© Giulio Guerrieri, Luc Pellissier, and Lorenzo Tortora de Falco;
licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 24; pp. 24:1–24:18



Leibniz International Proceedings in Informatics

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

many applications in the theory of the λ -calculus, *e.g.* in the study of linear head reduction [12], normalization [24, 27], Böhm trees [4, 19], λ -theories [20], intersection types [22]. More generally, understanding the relation between a program and its Taylor expansion renews the logical approach to the quantitative analysis of computation started with the inception of LL.

A natural question is the *inverse Taylor expansion problem*: how to characterize which sets of resource λ -terms are contained in the Taylor expansion of a same λ -term? Ehrhard and Regnier [14] defined a simple *coherence* relation such that a finite set of resource λ -terms is included in the Taylor expansion of a λ -term if and only if the elements of this set are pairwise coherent. Coherence is crucial in many structural properties of the resource λ -calculus, such as in the proof that in the λ -calculus normalization and Taylor expansion commute [12, 14].

We aim to solve the inverse Taylor expansion problem in the more general context of LL, more precisely in the *multiplicative-exponential fragment* MELL of LL, being aware that for MELL no coherence relation can solve the problem (see below).

Proof-nets, proof-structures and their Taylor expansion: seeing trees behind graphs. In MELL, linearity and the sharp analysis of computations naturally lead to represent proofs in a more general *graph*-like syntax instead of a term-like or tree-like one.¹ Indeed, linear negation is involutive and classical duality can be interpreted as the possibility of juggling between different conclusions, without a distinguished output. Graphs representing proofs in MELL are called *proof-nets*: their syntax is richer and more expressive than the λ -calculus. Contrary to λ -terms, proof-nets are special inhabitants of the wider land of *proof-structures*: they can be characterized, among proof-structures, by abstract (geometric) conditions called correctness criteria [15]. The procedure of cut-elimination can be applied to proof-structures, and proof-nets can also be seen as the proof-structures with a good behavior with respect to cut-elimination [1]. Proof-structures can be interpreted in denotational models and proof-nets can be characterized among them by semantic means [25]. It is then natural to attack problems in the general framework of proof-structures. In this work, correctness plays no role at all, hence we will consider proof-structures and not only proof-nets. MELL proof-structures are a particular kind of graphs, whose edges are labeled by MELL formulæ and vertices by MELL connectives, and for which special subgraphs are highlighted, the *boxes*, representing the parts of the proof-structure that can be copied and discarded (*i.e.* called an unbounded number of times). A box is delimited from the rest of a proof-structure by exponential modalities: its border is made of one !-cell, its principal door, and arbitrarily many ?-cells, its auxiliary doors. Boxes are nested or disjoint (they cannot partially overlap), so as to add a tree-like structure to proof-structures *aside* from their graph-like nature.

As in λ -calculus, one can define [13] box-free *resource proof-structures*², where !-cells make resources available boundedly, and the *Taylor expansion* of MELL proof-structures into these resource proof-structures, that recursively copies the content of the boxes an arbitrary number of times. In fact, as somehow anticipated by Boudes [3], such a Taylor expansion operation can be carried on any tree-like structure. This primitive, abstract, notion of Taylor expansion can then be pulled back to the structure of interest, as shown in [18] and put forth again here.

The question of coherence for proof-structures. The inverse Taylor expansion problem has a natural counterpart in the world of MELL proof-structures: given a set of resource proof-structures, is there a MELL proof-structure the expansion of which contains the set?

¹ A term-like object is essentially a tree, with one output (its root) and many inputs (its other leaves).

² Also known as differential proof-structures [6] or differential nets [13, 21, 7] or simple nets [23].

Pagani and Tasson [23] give the following answer: it is possible to decide whether a finite set of resource proof-structures is a subset of the Taylor expansion of a same MELL proof-structure (and even possible to do it in non-deterministic polynomial time); but unlike the λ -calculus, the structure of the relation “being part of the Taylor expansion of a same proof-structure” is *much more* complicated than a binary (or even n -ary) coherence. Indeed, for any $n > 1$, it is possible to find $n + 1$ resource proof-structures such that any n of them are in the Taylor expansion of some MELL proof-structure, but there is no MELL proof-structure whose Taylor expansion has all the $n + 1$ as elements (see our Example 21 and [26, pp. 244-246]).

In this work, we introduce a new combinatorial criterion, *glueability*, for deciding whether a set of resource proof-structures is a subset of the Taylor expansion of some MELL proof structure, based on a rewriting system on sequences of MELL formulæ. Our criterion is more general (and, we believe, simpler) than the one of [23], which is limited to the *cut-free* case with *atomic axioms* and characterizes only *finite* sets: we do not have these limitations. We believe that our criterion is a useful tool for studying proof-structures. We conjecture that it can be used to show that, for a suitable geometric restriction, a binary coherence relation does exist for resource proof-structures. It might also shed light on correctness and sequentialization.

As the proof-structures we consider are typed, an unrelated difficulty arises: a resource proof-structure might not be in the Taylor expansion of any MELL proof-structure, not because it does not respect the structure imposed by the Taylor expansion, but because its type is impossible.³ To solve this issue we enrich the MELL proof-structure syntax with a “universal” proof-structure: a special \boxtimes -cell (*daimon*) that can have any number of outputs of any types, and we allow it to appear inside a box, representing information plainly missing (see Section 8 for more details and the way this matter is handled by Pagani and Tasson [23]).

2 Outline and technical issues

The rewritings. The essence of our rewriting system is not located on proof-structures but on lists of MELL formulæ (Definition 9). In a very down-to-earth way, this rewriting system is generated by elementary steps akin to rules of sequent calculus read from the *bottom up*: they act on a list of conclusions, analogous to a monolaterous right-handed sequent. These steps are actually more sequentialized than sequent calculus rules, as they do not allow for commutation. For instance, the rule corresponding to the introduction of a \otimes on the i -th formula, is defined as $\otimes_i : (\gamma_1, \dots, \gamma_{i-1}, A \otimes B, \gamma_{i+1}, \dots, \gamma_n) \rightarrow (\gamma_1, \dots, \gamma_{i-1}, A, B, \gamma_{i+1}, \dots, \gamma_n)$.

$$\begin{array}{c}
 \begin{array}{ccc}
 \textcircled{\text{ax}} & & \textcircled{\text{ax}} \\
 \downarrow & & \downarrow \\
 A & A^\perp & A & A^\perp \\
 \downarrow & & \downarrow & \\
 \otimes & & & \\
 \downarrow & & & \\
 A \otimes A^\perp & & &
 \end{array}
 \end{array}
 \xrightarrow{\otimes_1}
 \begin{array}{ccc}
 \textcircled{\text{ax}} & & \textcircled{\text{ax}} \\
 \downarrow & & \downarrow \\
 A & A^\perp & A & A^\perp
 \end{array}$$

These rewrite steps then act on MELL proof-structures, coherently with their type, by modifying (most of the times, erasing) the cells directly connected to the conclusion of the proof-structure. Formally, this means that there is a functor $\mathbf{qMELL}^{\boxtimes}$ from the rewrite steps into the category \mathbf{Rel} of sets and relations, associating with a list of formulæ the set of MELL proof-structures with these conclusions, and with a rewrite step a relation implementing it (Definition 12). The rules *deconstruct* the proof-structure, starting from its conclusions. The rule \otimes_1 acts by removing a \otimes -cell on the first conclusion, replacing it by two conclusions.

³ Similarly, in the λ -calculus, there is no closed λ -term of type $X \rightarrow Y$ with $X \neq Y$ atomic, but the resource λ -term $(\lambda f.f)[\]$ can be given that type: the empty bag $[\]$ kills any information on the argument.

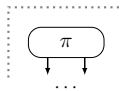
24:4 Glueability of Resource Proof-Structures: Inverting the Taylor Expansion

These rules can only act on specific proof-structures, and indeed, capture a lot of their structure: \otimes_i can be applied to a MELL proof-structure R if and only if R has a \otimes -cell in the conclusion i (as opposed to, say, an axiom). So, in particular, every proof-structure is completely characterized by any sequence rewriting it to the empty proof-structure.

Naturality. The same rules act also on sets of resource proof-structures, defining the functor $\mathfrak{PqDiLL}_0^{\mathfrak{X}}$ from the rewrite steps into the category \mathbf{Rel} (Definition 17). When carefully defined, the Taylor expansion induces a *natural transformation* from $\mathfrak{PqDiLL}_0^{\mathfrak{X}}$ to $\mathbf{qMELL}^{\mathfrak{X}}$ (Theorem 18). By applying this naturality repeatedly, we get our characterization (Theorem 20): a set of resource proof-structures Π is a subset of the Taylor expansion of a MELL proof-structure iff there is a sequence rewriting Π to the singleton of the *empty* proof-structure.

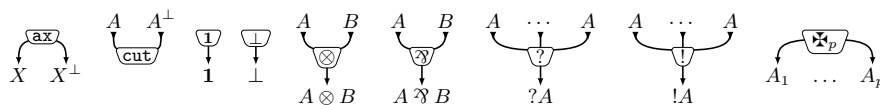
The naturality property is not only a mean to get our characterization, but also an interesting result in itself: natural transformations can often be used to express fundamental properties in a mathematical context. In this case, the *Taylor expansion is natural* with respect to the possibility to build a (MELL or resource) proof-structure by adding a cell to its conclusions or boxing it. Said differently, naturality of the Taylor expansion roughly means that the rewrite rules that deconstruct a MELL proof-structure R and a set of resource proof-structures in the Taylor expansion of R mimic each other.

Quasi-proof-structures and mix. Our rewrite rules consume proof-structures from their conclusions. The rule corresponding to boxes in MELL opens a box by deleting its principal door (a $!$ -cell) and its border, while for a resource proof-structure it deletes a $!$ -cell and separates the different copies of the content of the box (possibly) represented by such a $!$ -cell. This operation is problematic in a twofold way. In a resource proof-structure, where the border of boxes is not marked, it is not clear how to identify such copies. On the other side, in a MELL proof-structure the content of a box is not to be treated as if it were at the same level as what is outside of the box: it can be copied many times or erased, while what is outside boxes cannot, and treating the content in the same way as the outside suppresses this distinction, which is crucial in LL. So, we need to remember that the content of a box, even if it is at depth 0 (*i.e.* not contained in any other box) after erasing the box wrapping it by means of our rewrite rules, is not to be mixed with the rest of the structure at depth 0.



In order for our proof-structures to provide this information, we need to generalize them and consider that a proof-structure can have not just a tree of boxes, but a *forest*: this yields the notion of *quasi-proof-structure* (Definition 1). In this way, according to our rewrite rules, opening a box by deleting its principal door amounts to taking a box in the tree and disconnecting it from its root, creating a new tree. We draw this in a quasi-proof-structure by surrounding elements having the same root with a dashed line, open from the bottom, remembering the phantom presence of the border of the box, even if it was erased. This allows one to open the box only when it is “alone”, surrounded by a dashed line (see Definition 11).

This is not merely a technical remark, as this generalization gives a status to the *mix* rule of LL: indeed, mixing two proofs amounts to taking two proofs and considering them as one, without any other modifications. Here, it amounts to taking two proofs, each with its box-tree, and considering them as one by merging the roots of their trees (see the *mix* step in Definition 11). We embed this design decision up to the level of formulæ, which



■ **Figure 1** Cells, with their labels and their typed inputs and outputs (ordered from left to right).

are segregated in different zones that have to be mixed before interacting (see the notion of partition of a finite sequence of formulæ in Section 3).

Geometric invariance and emptiness: the filled Taylor expansion. The use of forests instead of trees for the nesting structure of boxes, where the different roots are thought of as the contents of long-gone boxes, has an interesting consequence in the Taylor expansion: indeed, an element of the Taylor expansion of a proof-structure contains an arbitrary number of copies of the contents of the boxes, in particular *zero*. If we think of the part at depth 0 of a MELL proof-structure as inside an invisible box, its content can be deleted in some elements of the Taylor expansion just as any other box.⁴ As erasing completely conclusions would cause the Taylor expansion not preserve the conclusions (which would lead to technical complications), we introduce the *filled Taylor expansion* (Definition 8), which contains not only the elements of the usual Taylor expansion, but also elements of the Taylor expansion where one component has been erased and replaced by a \boxtimes -cell (*daimon*), representing a lack of information, apart from the number and types of the conclusions.

Atomic axioms. Our paper first focuses on the case where proof-structures are restricted to *atomic axioms*. In Section 7 we sketch how to adapt our method to the non-atomic case.

3 Proof-structures and the Taylor expansion

MELL formulæ and (quasi-)proof-structures. Given a countably infinite set of propositional variables X, Y, Z, \dots , MELL *formulæ* are defined by the following inductive grammar:

$$A, B ::= X \mid X^\perp \mid \mathbf{1} \mid \perp \mid A \otimes B \mid A \wp B \mid !A \mid ?A$$

Linear negation is defined via De Morgan laws $\mathbf{1}^\perp = \perp$, $(A \otimes B)^\perp = A^\perp \wp B^\perp$ and $(!A)^\perp = ?A$, so as to be involutive, *i.e.* $A^{\perp\perp} = A$. Given a list $\Gamma = (A_1, \dots, A_m)$ of MELL formulæ, a *partition* of Γ is a list $(\Gamma_1, \dots, \Gamma_n)$ of lists of MELL formulæ such that there are $0 = i_0 < \dots < i_n = m$ with $\Gamma_j = (A_{i_{j-1}+1}, \dots, A_{i_j})$ for all $1 \leq j \leq n$; such a partition of Γ is also denoted by $(A_1, \dots, A_{i_1}; \dots; A_{i_{n-1}+1}, \dots, A_m)$, with lists separated by semi-colons.

We reuse the syntax of proof-structures given in [18] and sketch here its main features. We suppose known definitions of (directed) graph, rooted tree, and morphism of these structures. In what follows we will speak of *tails* in a graph: “hanging” edges with only one vertex. This can be implemented either by adding special vertices or using [2]’s graphs.

If an edge e is incoming in (resp. outgoing from) a vertex v , we say that e is a *input* (resp. *output*) of v . The reflexive-transitive closure of a tree τ is denoted by τ° : the operator $(\cdot)^\circ$ lifts to a functor from the category of trees to the category of directed graphs.

⁴ The dual case, of copying the contents of a box, poses no problem in our approach.

► **Definition 1.** A module M is a (finite) directed graph with:

- vertices v labeled by $\ell(v) \in \{\mathbf{ax}, \mathbf{cut}, \mathbf{1}, \perp, \otimes, \wp, ?, !\} \cup \{\mathfrak{X}_p \mid p \in \mathbb{N}\}$, the type of v ;
- edges e labeled by a MELL formula $c(e)$, the type of e ;
- an order $<_M$ that is total on the tails of $|M|$ and on the inputs of each vertex of type \wp, \otimes .

Moreover, all the vertices verify the conditions of Figure 1.⁵

A quasi-proof-structure is a triple $R = (|R|, \mathcal{F}, \mathbf{box})$ where:

- $|R|$ is a module with no input tails, called the module of R ;
 - \mathcal{F} is a forest of rooted trees with no input tails, called the box-forest of R ;
 - $\mathbf{box}: |R| \rightarrow \mathcal{F}^\circ$ is a morphism of directed graphs, the box-function of R , which induces a partial bijection from the inputs of the vertices of type $!$ and the edges in \mathcal{F} , and such that:
 - for any vertices v, v' with an edge from v' to v , if $\mathbf{box}(v) \neq \mathbf{box}(v')$ then $\ell(v) \in \{!, ?\}$.⁶
- Moreover, for any output tails e_1, e_2, e_3 in $|R|$ which are outputs of the vertices v_1, v_2, v_3 , respectively, if $e_1 <_{|R|} e_2 <_{|R|} e_3$ then it is impossible that $\mathbf{box}(v_1) = \mathbf{box}(v_3) \neq \mathbf{box}(v_2)$.⁷

A quasi-proof-structure $R = (|R|, \mathcal{F}, \mathbf{box})$ is:

1. $\text{MELL}^{\mathfrak{X}}$ if all vertices in $|R|$ of type $!$ have exactly one input, and the partial bijection induced by \mathbf{box} from the inputs of the vertices of type $!$ in $|R|$ and the edges in \mathcal{F} is total.
2. MELL if it is $\text{MELL}^{\mathfrak{X}}$ and, for every vertex v in $|R|$ of type \mathfrak{X} , one has $\mathbf{box}^{-1}(\mathbf{box}(v)) = \{v\}$ and $\mathbf{box}(v)$ is not a root of the box-forest \mathcal{F} of R .
3. $\text{DiLL}_0^{\mathfrak{X}}$ if the box-forest \mathcal{F} of R is just a juxtaposition of roots.
4. DiLL_0 (or resource) if it is $\text{DiLL}_0^{\mathfrak{X}}$ and there is no vertex in $|R|$ of type \mathfrak{X} .

For the previous systems, a proof-structure is a quasi-proof-structure whose box-forest is a tree.

Our MELL proof-structure (i.e. a MELL quasi-proof-structure that is also a proof-structure) corresponds to the usual notion of MELL proof-structure (as in [8]) except that we also allow the presence of a box filled only by a *daimon* (i.e. a vertex of type \mathfrak{X}). The *empty* (DiLL_0 and MELL) proof-structure – whose module and box-forest are empty graphs – is denoted by ε .

Given a quasi-proof-structure $R = (|R|, \mathcal{F}, \mathbf{box})$, the output tails of $|R|$ are the *conclusions* of R . So, the pre-images of the roots of \mathcal{F} via \mathbf{box} partition the conclusions of R in a list of lists of such conclusions. The *type* of R is the list of lists of the types of these conclusions. We often identify the conclusions of R with a finite initial segment of \mathbb{N} .

By definition of graph morphism, two conclusions in two distinct lists in the type of a quasi-proof-structure R are in two distinct connected components of $|R|$; so, if R is not a proof-structure then $|R|$ contains several connected components. Thus, R can be seen as a list of proof-structures, its *components*, one for each root in its box-forest.

A non-root vertex v in the box-forest \mathcal{F} induces a subgraph of \mathcal{F}° of all vertices above it and edges connecting them. The pre-image of this subgraph through \mathbf{box} is the *box* of v and the conditions on \mathbf{box} in Definition 1 translate the usual nesting condition for LL boxes.

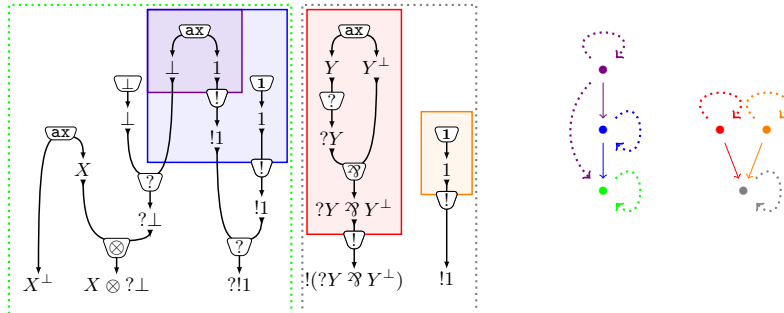
In quasi-proof-structures, we speak of *cells* instead of vertices, and, for a cell of type ℓ , of a ℓ -*cell*. A \mathfrak{X} -cell is a \mathfrak{X}_p -cell for some $p \in \mathbb{N}$. An *hypothesis cell* is a cell without inputs.

► **Example 2.** The graph in Figure 2 is a MELL quasi-proof-structure. The colored areas represent the pre-images of boxes, and the dashed boxes represent the pre-images of roots.

⁵ Note that there are no conditions on the types of the outputs of vertices of type \mathfrak{X} (i.e. of type \mathfrak{X}_p for some $p \in \mathbb{N}$); and the outputs of vertices of type \mathbf{ax} must have *atomic* types.

⁶ Roughly, it says that the border of a box is made of (inputs of) vertices of type $!$ or $?$.

⁷ This is a technical condition that simplifies the definition of the rewrite rules in Section 4. Note that $\mathbf{box}(v_1), \mathbf{box}(v_2), \mathbf{box}(v_3)$ are necessarily roots in \mathcal{F} , since \mathbf{box} is a morphism of directed graphs.



■ **Figure 2** A MELΛ quasi-proof-structure R , its box-forest \mathcal{F}_R (without dotted lines) and the reflexive-transitive closure \mathcal{F}_R^O of \mathcal{F}_R (with also dotted lines).

The Taylor expansion. Proof-structures have a tree structure made explicit by their box-function. Following [18], the definition of the Taylor expansion uses this tree structure: first, we define how to “*expand*” a tree – and more generally a forest – via a generalization of the notion of thick subtree [3] (Definition 3; roughly, a thick subforest of a box-forest says the number of copies of each box to be taken, iteratively), we then take all the expansions of the tree structure of a proof-structure and we *pull* the approximations *back* to the underlying graphs (Definition 5), finally we *forget* the tree structures associated with them (Definition 6).

► **Definition 3** (thick subforest). *Let τ be a forest of rooted trees. A thick subforest of τ is a pair (σ, h) of a forest σ of rooted trees and a graph morphism $h: \sigma \rightarrow \tau$ whose restriction to the roots of σ is bijective.*

► **Example 4.** The following is a graphical presentation of a thick subforest (τ, h) of the box-forest \mathcal{F} of the quasi-proof-structure in Figure 2, where the graph morphism $h: \tau \rightarrow \mathcal{F}$ is depicted chromatically (same color means same image via h).

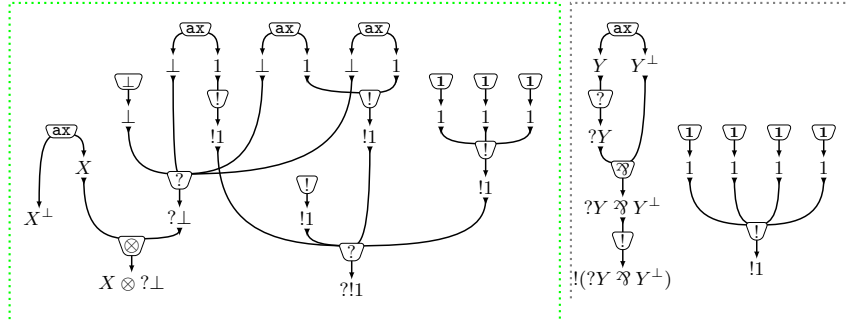


Intuitively, it means that τ is obtained from \mathcal{F} by taking 3 copies of the blue box, 1 copy of the red box and 4 copies of the orange box; in the first (resp. second; third) copy of the blue box, 1 copy (resp. 0 copies; 2 copies) of the purple box has been taken.

► **Definition 5** (proto-Taylor expansion). *Let $R = (|R|, \mathcal{F}_R, \text{box}_R)$ be a quasi-proof-structure. The proto-Taylor expansion of R is the set $\mathcal{T}^{\text{proto}}(R)$ of thick subforests of \mathcal{F}_R .*

Let $t = (\tau_t, h_t) \in \mathcal{T}^{\text{proto}}(R)$. The t -expansion of R is the pullback (R_t, p_t, p_R) below, computed in the category of directed graphs and graph morphisms.

$$\begin{array}{ccc}
 R_t & \xrightarrow{p_t} & \tau_t^O \\
 \downarrow p_R & \lrcorner & \downarrow h_t^O \\
 |R| & \xrightarrow{\text{box}_R} & \mathcal{F}_R^O
 \end{array}$$



■ **Figure 3** The element of the Taylor expansion of the MELL quasi-proof-structure R in Figure 2, obtained from the element of $\mathcal{T}^{\text{proto}}(R)$ depicted in Example 4.

Given a quasi-proof-structure R and $t = (\tau_t, h_t) \in \mathcal{T}^{\text{proto}}(R)$, the directed graph R_t inherits labels on vertices and edges by composition with the graph morphism $p_R: R_t \rightarrow |R|$.

Let $[\tau_t]$ be the forest made up of the roots of τ_t and $\iota: \tau_t \rightarrow [\tau_t]$ be the graph morphism sending each vertex of τ_t to the root below it; ι° induces by post-composition a morphism $\bar{h}_t = \iota^\circ \circ p_t: R_t \rightarrow [\tau_t]^\circ$. The triple $(R_t, [\tau_t], \bar{h}_t)$ is a DiLL_0 quasi-proof-structure, and it is a DiLL_0 proof-structure if R is a proof-structure. We can then define the *Taylor expansion* $\mathcal{T}(R)$ of a quasi-proof-structure R (an example of an element of a Taylor expansion is in Figure 3).

► **Definition 6** (Taylor expansion). *Let R be a quasi-proof-structure. The Taylor expansion of R is the set of DiLL_0 quasi-proof-structures $\mathcal{T}(R) = \{(R_t, [\tau_t], \bar{h}_t) \mid t = (\tau_t, h_t) \in \mathcal{T}^{\text{proto}}(R)\}$.*

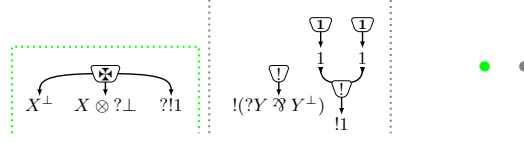
An element $(R_t, [\tau_t], \bar{h}_t)$ of the Taylor expansion of a quasi-proof-structure R has much less structure than the pullback (R_t, p_t, p_R) : the latter indeed is a DiLL_0 quasi-proof-structure R_t coming with its projections $|R| \xleftarrow{p_R} R_t \xrightarrow{p_t} \tau_t^\circ$, which establish a precise correspondence between cells and edges of R_t and cells and edges of R : a cell in R_t is labeled (via the projections) by both the cell of $|R|$ and the branch of the box-forest of R it arose from. But $(R_t, [\tau_t], \bar{h}_t)$ where R_t is without its projections p_t and p_R loses the correspondence with R .

► **Remark 7.** By definition, the Taylor expansion preserves conclusions: there is a bijection φ from the conclusions of a quasi-proof-structure R to the ones in each element ρ of $\mathcal{T}(R)$ such that i and $\varphi(i)$ have the same type and the same root (*i.e.* $\text{box}_R(i) = \text{box}_\rho(\varphi(i))$) up to isomorphism). Therefore, the types of R and ρ are the same (as a list of lists).

The filled Taylor expansion. As discussed in Section 2 (p. 5), our method needs to “represent” the emptiness introduced by the Taylor expansion (taking 0 copies of a box) so as to preserve the conclusions. So, an element of the *filled Taylor expansion* $\mathcal{T}^\boxtimes(R)$ of a quasi-proof-structure R (an example is in Figure 4) is obtained from an element of $\mathcal{T}(R)$ where a whole component can be erased and replaced by a \boxtimes -cell with the same conclusions (hence $\mathcal{T}(R) \subseteq \mathcal{T}^\boxtimes(R)$).

► **Definition 8** (filled Taylor expansion). *An emptying of a DiLL_0 quasi-proof-structure $\rho = (|\rho|, \mathcal{F}, \text{box})$ is the DiLL_0 quasi-proof-structure with the same conclusions as ρ , obtained from ρ by replacing each of the components of some roots of \mathcal{F} with a \boxtimes -cell whose outputs are tails.*

The filled Taylor expansion $\mathcal{T}^\boxtimes(R)$ of a quasi-proof-structure R is the set of all the emptyings of every element of its Taylor expansion $\mathcal{T}(R)$.



■ **Figure 4** An element of the filled Taylor expansion of the MELL quasi-proof-structure in Figure 2.

$$\begin{array}{l}
(\Gamma_1; \dots; \Gamma_k, c(i), c(i+1), \Gamma'_k; \dots; \Gamma_n) \xrightarrow{\text{exc}_i} (\Gamma_1; \dots; \Gamma_k, c(i+1), c(i), \Gamma'_k; \dots; \Gamma_n) \\
(\Gamma_1; \dots; \Gamma_k, c(i), c(i+1), \Gamma'_k; \dots; \Gamma_n) \xrightarrow{\text{mix}_i} (\Gamma_1; \dots; \Gamma_k, c(i), c(i+1), \Gamma'_k; \dots; \Gamma_n) \\
(\Gamma_1; \dots; \Gamma_k; c(i), c(i+1); \Gamma_{k+2}; \dots; \Gamma_n) \xrightarrow{\text{ax}_i} (\Gamma_1; \dots; \Gamma_k; \Gamma_{k+2}; \dots; \Gamma_n) \quad \text{with } c(i) = A = c(i+1)^\perp \\
(\Gamma_1; \dots; \Gamma_k; \dots; \Gamma_n) \xrightarrow{\text{cut}_i} (\Gamma_1; \dots; \Gamma_k, c(i), c(i+1); \dots; \Gamma_n) \quad \text{with } c(i) = A = c(i+1)^\perp \\
(\Gamma_1; \dots; \Gamma_k; \Gamma_{k+1}, c(i); \Gamma_{k+2}; \dots; \Gamma_n) \xrightarrow{\boxtimes_i} (\Gamma_1; \dots; \Gamma_k; \Gamma_{k+2}; \dots; \Gamma_n) \\
(\Gamma_1; \dots; \Gamma_k; c(i); \Gamma_{k+2}; \dots; \Gamma_n) \xrightarrow{1_i} (\Gamma_1; \dots; \Gamma_k; \Gamma_{k+2}; \dots; \Gamma_n) \quad \text{with } c(i) = 1 \\
(\Gamma_1; \dots; \Gamma_k; c(i); \Gamma_{k+2}; \dots; \Gamma_n) \xrightarrow{\perp_i} (\Gamma_1; \dots; \Gamma_k; \Gamma_{k+2}; \dots; \Gamma_n) \quad \text{with } c(i) = \perp \\
(\Gamma_1; \dots; \Gamma_k, c(i); \dots; \Gamma_n) \xrightarrow{\otimes_i} (\Gamma_1; \dots; \Gamma_k, A, B; \dots; \Gamma_n) \quad \text{with } c(i) = A \otimes B \\
(\Gamma_1; \dots; \Gamma_k, c(i); \dots; \Gamma_n) \xrightarrow{\wp_i} (\Gamma_1; \dots; \Gamma_k, A, B; \dots; \Gamma_n) \quad \text{with } c(i) = A \wp B \\
(\Gamma_1; \dots; \Gamma_k, c(i); \dots; \Gamma_n) \xrightarrow{?_i} (\Gamma_1; \dots; \Gamma_k, ?A, ?A; \dots; \Gamma_n) \quad \text{with } c(i) = ?A \\
(\Gamma_1; \dots; \Gamma_k, c(i); \dots; \Gamma_n) \xrightarrow{!_i} (\Gamma_1; \dots; \Gamma_k, A; \dots; \Gamma_n) \quad \text{with } c(i) = ?A \\
(\Gamma_1; \dots; \Gamma_k; c(i); \Gamma_{k+2}; \dots; \Gamma_n) \xrightarrow{?_{i+1}} (\Gamma_1; \dots; \Gamma_k; \Gamma_{k+2}; \dots; \Gamma_n) \quad \text{with } c(i) = ?A \\
(\Gamma_1; \dots; ?\Gamma_k, c(i); \dots; \Gamma_n) \xrightarrow{\text{Box}_i} (\Gamma_1; \dots; ?\Gamma_k, A; \dots; \Gamma_n) \quad \text{with } c(i) = !A
\end{array}$$

■ **Figure 5** The generators of **Path**. In the source $\Gamma = (A_1, \dots, A_{i_1}; \dots; A_{i_{m-1}+1}, \dots, A_{i_n})$ of each arrow, $c(i)$ denotes the i^{th} formula in the flattening $(A_1, \dots, A_{i_1}, \dots, A_{i_{m-1}+1}, \dots, A_{i_n})$ of Γ .

4 Means of destruction: unwinding MELL quasi-proof-structures

Our aim is to deconstruct proof-structures (be they MELL[⊗] or DiLL₀) from their conclusions. To do that, we introduce a category of rules of deconstruction. The morphisms of this category are sequences of deconstructing rules, acting on lists of lists of formulæ. These morphisms act through functors on quasi-proof-structures, exhibiting their sequential structure.

► **Definition 9** (the category **Path**). *Let **Path** be the category whose*


- *objects are lists $\Gamma = (\Gamma_1; \dots; \Gamma_n)$ of lists of MELL formulæ;*
- *arrows are freely generated by the elementary paths in Figure 5.*

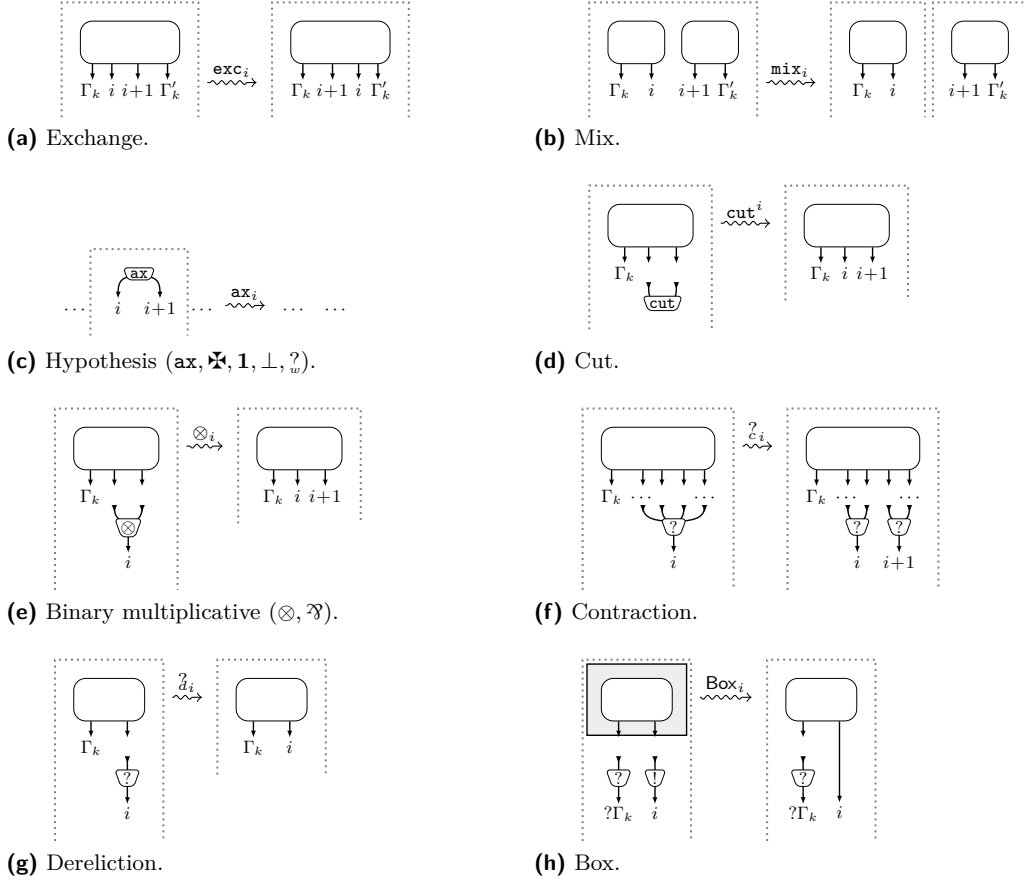
We call a path any arrow $\xi: \Gamma \rightarrow \Gamma'$. We write the composition of paths without symbols and in the diagrammatic order, so, if $\xi: \Gamma \rightarrow \Gamma'$ and $\xi': \Gamma' \rightarrow \Gamma''$, $\xi\xi': \Gamma \rightarrow \Gamma''$.

► **Example 10.** $\wp_1 \wp_2 \wp_3 \otimes_1 \otimes_3 \text{exc}_1 \text{exc}_2 \text{mix}_2 \text{ax}_1 \text{exc}_2 \text{mix}_2 \text{ax}_1 \text{ax}_1$ is a path of type $((X \otimes Y^\perp) \wp ((Y \otimes Z^\perp) \wp (X^\perp \wp Z))) \rightarrow \varepsilon$, where ε is the empty list of lists of formulæ.

We will tend to forget about exchanges and perform them silently (as it is customary, for instance, in most presentations of sequent calculi).

The category **Path** acts on MELL[⊗] quasi-proof-structures, exhibiting a sequential structure in their construction. For Γ a list of lists of MELL formulæ, $\mathbf{qMELL}^\otimes(\Gamma)$ is the set of MELL[⊗] quasi-proof-structures of type Γ . To ease the reading of the rewrite rules acting on a MELL[⊗] quasi-proof-structures R , we will only draw the parts of R belonging to the relevant component; *e.g.*, if we are interested in an **ax**-cell whose outputs are the conclusions i and

$i+1$, and it is the only cell in a component, we will write  ignoring the rest.



■ **Figure 6** Actions of elementary paths on $\text{MELL}^{\mathbf{x}}$ quasi-proof-structures.

► **Definition 11** (action of paths on MELL quasi-proof-structures). An elementary path $a: \Gamma \rightarrow \Gamma'$ defines a relation $\mathfrak{a} \subseteq \mathbf{qMELL}^{\mathbf{x}}(\Gamma) \times \mathbf{qMELL}^{\mathbf{x}}(\Gamma')$ (the action of a) as the smallest relation containing all the cases in Figure 6, with the following remarks:

mix read in reverse, a quasi-proof-structure with two components is in relation with a quasi-proof-structure with the same module but the two roots of such components merged.

hypothesis if $a \in \{\text{ax}_i, \mathbf{x}_i, \mathbf{1}_i, \perp_i, \text{?}_i\}$, the rules have all in common to act by deleting a cell without inputs that is the only cell in its component. We have drawn the axiom case in Figure 6c, the others vary only by their number of conclusions.

cut read in reverse, a quasi-proof-structure with two conclusions i and $i+1$ is in relation with the quasi-proof-structure where these two conclusions are cut. This rule, from left to right, is non-deterministic (as there are many possible cuts).

binary multiplicatives these rules delete a binary connective. We have only drawn the \otimes case in Figure 6e, the \wp case is similar.

contraction splits a ? -cell with $h+k+2$ inputs into two ? -cells with $h+1$ and $k+1$ inputs, respectively.

dereliction only applies if the ? -cell (with 1 input) does not shift a level in the box-forest.

box only applies if a box (and its border) is alone in its component.

This definition of the rewrite system is extended to define a relation $\mathfrak{L} \subseteq \mathbf{qMELL}^{\mathbf{x}}(\Gamma) \times \mathbf{qMELL}^{\mathbf{x}}(\Gamma')$ (the action of any path $\xi: \Gamma \rightarrow \Gamma'$) by composition of relations.

Given two MELL^{\boxtimes} quasi-proof-structures R and R' , we say that a rule a applies to R if there is a finite sequence of exchanges $\text{exc}_{i_1} \cdots \text{exc}_{i_n}$ such that $R \xrightarrow{\text{exc}_{i_1} \cdots \text{exc}_{i_n} a} R'$.

- **Definition 12** (the functor $\mathbf{qMELL}^{\boxtimes}$). We define a functor $\mathbf{qMELL}^{\boxtimes} : \mathbf{Path} \rightarrow \mathbf{Rel}$ by:
- on objects: $\mathbf{qMELL}^{\boxtimes}(\Gamma)$ is the set of MELL^{\boxtimes} quasi-proof-structures of type Γ ;
 - on morphisms: for $\xi : \Gamma \rightarrow \Gamma'$, $\mathbf{qMELL}^{\boxtimes}(\xi) = \overset{\xi}{\rightsquigarrow}$ (see Definition 11).

Our rewrite rules enjoy two useful properties, expressed by Propositions 13 and 15.

- **Proposition 13** (co-functionality). Let $\xi : \Gamma \rightarrow \Gamma'$ be a path. The relation $\overset{\xi}{\rightsquigarrow}$ is a co-function on the sets of underlying graphs, that is, a function $\overset{\xi}{\rightsquigarrow}^{\text{op}} : \mathbf{qMELL}^{\boxtimes}(\Gamma') \rightarrow \mathbf{qMELL}^{\boxtimes}(\Gamma)$.

- **Lemma 14** (applicability of rules). Let R be a non-empty MELL^{\boxtimes} quasi-proof-structure. There exists a conclusion i such that:

- either a rule in $\{\mathbf{ax}_i, \mathbf{1}_i, \perp_i, \otimes_i, \wp_i, ?_i, ?_i, ?_i, \text{cut}^i, \boxtimes_i, \text{Box}_i\}$ applies to R ;
- or $R \xrightarrow{\text{mix}_i} R'$ (where the conclusions affected by mix_i are $i-k, \dots, i, i+1, \dots, i+\ell$) and $i-k, \dots, i$ are all the conclusions of either a box or an hypothesis cell, and one of the components of R' coincides with this cell or box (and its border).

Proposition 13 and Lemma 14 are proven by simple inspection of the rewrite rules of Figure 6.

- **Proposition 15** (termination). Let R be a MELL^{\boxtimes} quasi-proof-structure of type Γ . There exists a path $\xi : \Gamma \rightarrow \varepsilon$ such that $R \xrightarrow{\xi} \varepsilon$.

To prove Proposition 15, it is enough to apply Lemma 14 and show that the size of MELL^{\boxtimes} quasi-proof-structures decreases for each application of the rules in Figure 6, according to the following definition of size. The *size* of a proof-structure R is the couple (p, q) where

- p is the (finite) multiset of the number of inputs of each $?$ -cell in R ;
- q is the number of cells not labeled by \boxtimes in R .

The *size* of a quasi-proof-structure R is the (finite) multiset of the sizes of its components. Multisets are ordered as usual, couples are ordered lexicographically.

5 Naturality of unwinding $\text{DiLL}_0^{\boxtimes}$ quasi-proof-structures

For Γ a list of lists of MELL formulæ, $\mathbf{qDiLL}_0^{\boxtimes}(\Gamma)$ is the set of $\text{DiLL}_0^{\boxtimes}$ quasi-proof-structures of type Γ . For any set X , its powerset is denoted by $\mathfrak{P}(X)$.

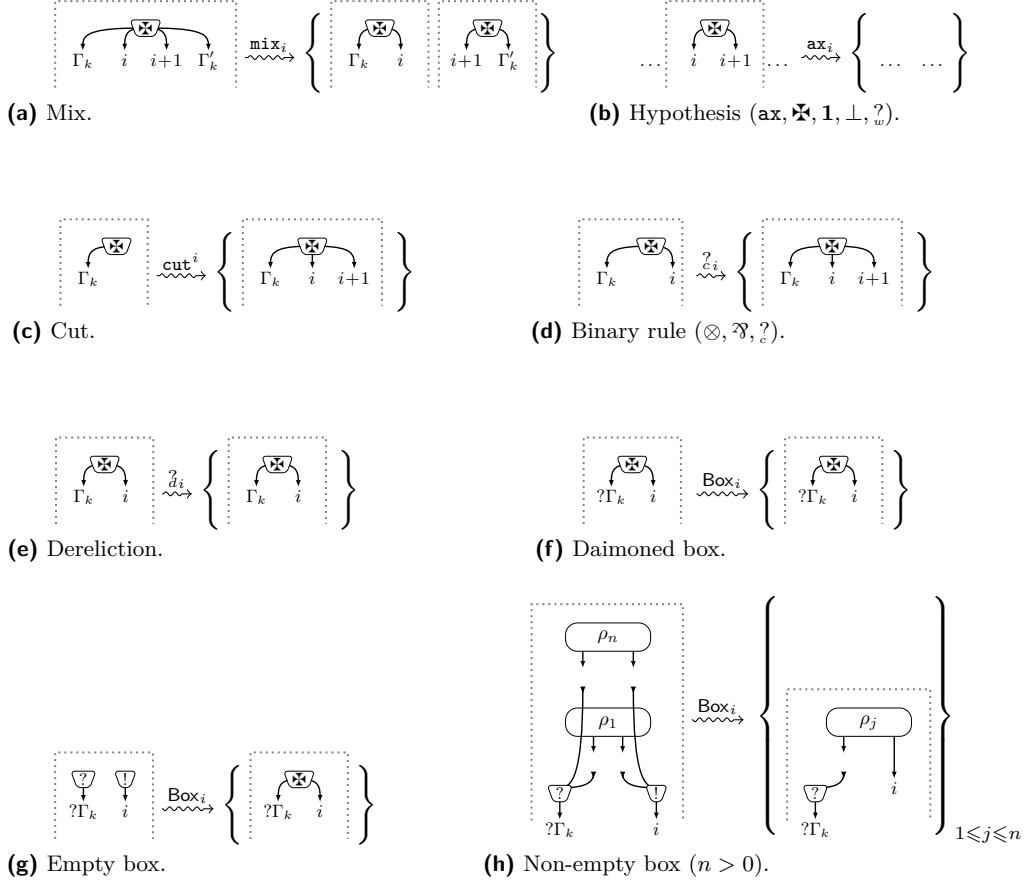
- **Definition 16** (action of paths on $\text{DiLL}_0^{\boxtimes}$ quasi-proof-structures). An elementary path $a : \Gamma \rightarrow \Gamma'$ defines a relation $\overset{a}{\rightsquigarrow} \subseteq \mathbf{qDiLL}_0^{\boxtimes}(\Gamma) \times \mathfrak{P}(\mathbf{qDiLL}_0^{\boxtimes}(\Gamma'))$ (the action of a) by the rules in Figure 6 (except Figure 6h, and with all the already remarked notes) and in Figure 7.

We extend this relation on $\mathfrak{P}(\mathbf{qDiLL}_0^{\boxtimes}(\Gamma)) \times \mathfrak{P}(\mathbf{qDiLL}_0^{\boxtimes}(\Gamma'))$ by the monad multiplication of $X \mapsto \mathfrak{P}(X)$ and define $\overset{\xi}{\rightsquigarrow}$ (the action of any path $\xi : \Gamma \rightarrow \Gamma'$) by composition of relations.

Roughly, all the rewrite rules in Figure 7 – except Figure 7h – mimic the behavior of the corresponding rule in Figure 6 using a \boxtimes -cell. Note that in Figure 7g a \boxtimes -cell is created.

The non-empty box rule in Figure 7h requires that, on the left of $\overset{\text{Box}_i}{\rightsquigarrow}$, ρ_j is not connected to $\rho_{j'}$ for $j \neq j'$, except for the $!$ -cell and the $?$ -cells in the conclusions. Read in reverse, the rule associates with a non-empty finite set of DiLL_0 quasi-proof-structures $\{\rho_1, \dots, \rho_n\}$ the merging of ρ_1, \dots, ρ_n , that is the DiLL_0 quasi-proof-structure depicted on the left of $\overset{\text{Box}_i}{\rightsquigarrow}$.

- **Definition 17** (the functor $\mathfrak{PqDiLL}_0^{\boxtimes}$). We define a functor $\mathfrak{PqDiLL}_0^{\boxtimes} : \mathbf{Path} \rightarrow \mathbf{Rel}$ by:
- on objects: for Γ a list of lists of MELL formulæ, $\mathfrak{PqDiLL}_0^{\boxtimes}(\Gamma) = \mathfrak{P}(\mathbf{qDiLL}_0^{\boxtimes}(\Gamma))$, the set of sets of $\text{DiLL}_0^{\boxtimes}$ quasi-proof-structures of type Γ ;
 - on morphisms: for $\xi : \Gamma \rightarrow \Gamma'$, $\mathfrak{PqDiLL}_0^{\boxtimes}(\xi) = \overset{\xi}{\rightsquigarrow}$ (see Definition 16).



■ **Figure 7** Actions of elementary paths on \mathfrak{X} -cells and on a box in $\mathbf{qDiLL}_0^{\mathfrak{X}}$.

► **Theorem 18** (naturality). *The filled Taylor expansion defines a natural transformation $\mathfrak{T}^{\mathfrak{X}}: \mathfrak{PqDiLL}_0^{\mathfrak{X}} \Rightarrow \mathbf{qMELL}^{\mathfrak{X}}: \mathbf{Path} \rightarrow \mathbf{Rel}$ by: $(\Pi, R) \in \mathfrak{T}_F^{\mathfrak{X}}$ iff $\Pi \subseteq \mathcal{T}^{\mathfrak{X}}(R)$ and the type of R is Γ . Moreover, if Π is a set of \mathbf{DiLL}_0 proof-structures with $\Pi \xrightarrow{\xi} \Pi'$ and $\Pi' \subseteq \mathcal{T}(R')$, then R is a \mathbf{MELL} proof-structure and $\Pi \subseteq \mathcal{T}(R)$, where R is such that $R \xrightarrow{\xi} R'$.⁸*

In other words, the following diagram commutes for every path $\xi: \Gamma \rightarrow \Gamma'$.

$$\begin{array}{ccc}
 \mathfrak{PqDiLL}_0^{\mathfrak{X}}(\Gamma) & \xrightarrow{\mathfrak{PqDiLL}_0^{\mathfrak{X}}(\xi)} & \mathfrak{PqDiLL}_0^{\mathfrak{X}}(\Gamma') \\
 \downarrow \mathfrak{T}_\Gamma^{\mathfrak{X}} & & \downarrow \mathfrak{T}_{\Gamma'}^{\mathfrak{X}} \\
 \mathbf{qMELL}^{\mathfrak{X}}(\Gamma) & \xrightarrow{\mathbf{qMELL}^{\mathfrak{X}}(\xi)} & \mathbf{qMELL}^{\mathfrak{X}}(\Gamma')
 \end{array}$$

It means that given $\Pi \xrightarrow{\xi} \Pi'$, where $\Pi' \subseteq \mathcal{T}^{\mathfrak{X}}(R')$, we can simulate backwards the rewriting to R (this is where the co-functionality of the rewriting steps expressed by Proposition 13

⁸ The part of the statement after “moreover” is our way to control the presence of \mathfrak{X} -cells.

comes handy) so that $R \xrightarrow{\xi} R'$ and $\Pi \subseteq \mathcal{T}^{\mathfrak{X}}(R)$; and conversely, given $R \xrightarrow{\xi} R'$, we can simulate the rewriting for any $\Pi \subseteq \mathcal{T}^{\mathfrak{X}}(R)$, so that $\Pi \xrightarrow{\xi} \Pi'$ for some $\Pi' \subseteq \mathcal{T}^{\mathfrak{X}}(R')$.

6 Glueability of DiLL₀ quasi-proof-structures

Naturality (Theorem 18) allows us to characterize the sets of DiLL₀ proof-structures that are in the Taylor expansion of some MELL proof-structure (Theorem 20 below).

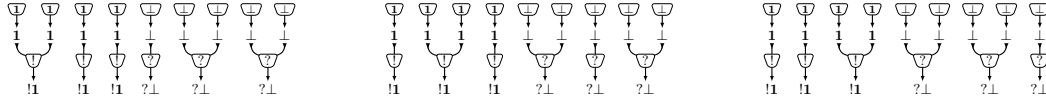
► **Definition 19** (glueability). *We say that a set Π of DiLL₀[ⓧ] quasi-proof-structures is glueable, if there exists a path ξ such that $\Pi \xrightarrow{\xi} \{\varepsilon\}$.*

► **Theorem 20** (glueability criterion). *Let Π be a set of DiLL₀ proof-structures: Π is glueable if and only if $\Pi \subseteq \mathcal{T}(R)$ for some MELL proof-structure R .*

Proof. If $\Pi \subseteq \mathcal{T}(R)$ for some MELL proof-structure R , then by termination (Proposition 15) $R \xrightarrow{\xi} \varepsilon$ for some path ξ , and so $\Pi \xrightarrow{\xi} \{\varepsilon\}$ by naturality (Theorem 18, as $\mathcal{T}^{\mathfrak{X}}(\varepsilon) = \{\varepsilon\}$).

Conversely, if $\Pi \xrightarrow{\xi} \{\varepsilon\}$ for some path ξ , then by naturality (Theorem 18, as $\mathcal{T}(\varepsilon) = \{\varepsilon\}$ and Π is a set of DiLL₀ proof-structures) $\Pi \subseteq \mathcal{T}(R)$ for some MELL proof-structure R . ◀

► **Example 21.** The three DiLL₀ proof-structures ρ_1, ρ_2, ρ_3 below are not glueable as a whole, but are glueable two by two. In fact, there is no MELL proof-structure whose Taylor expansion contains ρ_1, ρ_2, ρ_3 , but any pair of them is in the Taylor expansion of some MELL proof-structure. This is a slight variant of the example in [26, pp. 244-246].



An example of the action of a path starting from a DiLL₀ proof-structure ρ and ending in $\{\varepsilon\}$ can be found in Figures 8 and 9. Note that it is by no means the shortest possible path. When replayed backwards, it induces a MELL proof-structure R such that $\rho \in \mathcal{T}(R)$.

7 Non-atomic axioms

From now on, we relax the definition of quasi-proof-structure (Definition 1 and Figure 1) so that the outputs of any ax-cell are labeled by dual MELL formulæ, not necessarily atomic. We can extend our results to this more general setting, with some technical complications. Indeed, the rewrite rule for contraction has to be modified. Consider a set of DiLL₀ proof-structures consisting of just a singleton which is a \mathfrak{X} -cell. The contraction rule rewrites it as:

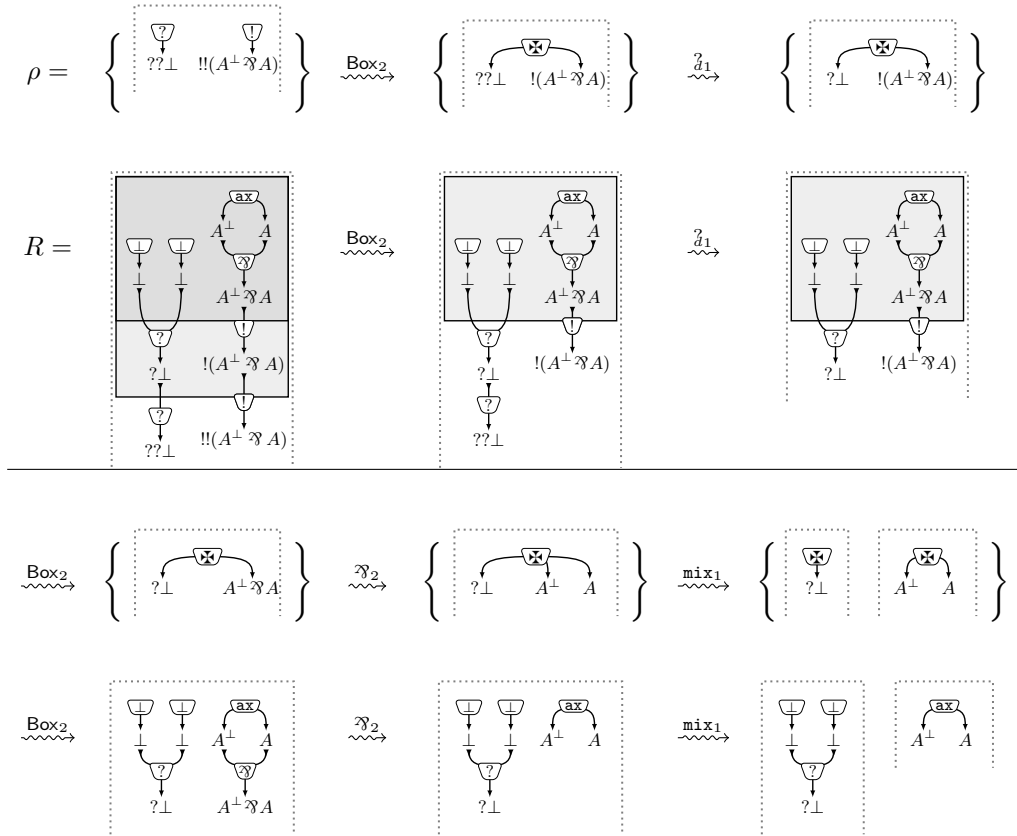
$$\begin{array}{c} \text{ax} \\ \curvearrowright \\ !A^\perp \quad !A^\perp \quad ?A \end{array} \xrightarrow{\mathfrak{C}_3} \left\{ \begin{array}{c} \text{ax} \\ \curvearrowright \\ !A^\perp \quad !A^\perp \quad ?A \quad ?A \end{array} \right\} \text{ which is then in the Taylor expansion of } \begin{array}{c} \text{ax} \\ \text{ax} \\ \curvearrowright \\ !A^\perp \quad !A^\perp \quad ?A \quad ?A \end{array}$$

on which no contraction rewrite rule \mathfrak{C} can be applied backwards, breaking the naturality. The failure of the naturality is actually due to the failure of Proposition 13 in the case of the rewrite rule \mathfrak{C} : \mathfrak{C}^{op} (i.e. \mathfrak{C} read from the right to the left) is functional but not total.

The solution to this conundrum lies in changing the contraction rule for DiLL₀[ⓧ] quasi-proof-structures, by explicitly adding \mathfrak{C} -cells. Hence, the application of a contraction step \mathfrak{C} in the DiLL₀[ⓧ] quasi-proof-structures precludes the possibility of anything else but a \mathfrak{C} -cell on the MELL[ⓧ] side, which allows the contraction step \mathfrak{C} to be applied backwards.

In turn, this forces us to change the definition of the filled Taylor expansion into a η -filled Taylor expansion, which has to include elements where a \mathfrak{X} -cell (representing an empty component) has some of its outputs connected to \mathfrak{C} -cells.

24:14 Glueability of Resource Proof-Structures: Inverting the Taylor Expansion



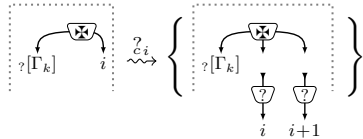
■ **Figure 8** The path $\text{Box}_2 \ ?_{d1} \ \text{Box}_2 \ ?_2 \ \text{mix}_1 \ \text{ax}_2 \ ?_{c1} \ ?_{d2} \ \text{mix}_1 \ \perp_2 \ ?_{d1} \ \perp_1$ witnessing that $\rho \in \mathcal{T}(R)$ (to be continued on Figure 9).

► **Definition 22** (η -filled Taylor expansion). An η -emptying of a DiLL_0 quasi-proof-structure $\rho = (|\rho|, \mathcal{F}, \text{box})$ is a DiLL_0 quasi-proof-structure with the same conclusions as ρ , obtained from ρ by replacing each of the components of some roots of \mathcal{F} with a \boxtimes -cell whose outputs are either tails or inputs of a $?$ -cell whose output i is a tail, provided that i is the output tail of a $?$ -cell in ρ .

The η -filled Taylor expansion $\mathcal{T}_\eta^{\boxtimes}(R)$ of a quasi-proof-structure R is the set of all the η -emptyings of every element of its Taylor expansion $\mathcal{T}(R)$.

Note that the η -filled Taylor expansion contains all the elements of the filled Taylor expansion and some more, such as the one in Figure 10.

Functors $\mathbf{qMELL}^{\boxtimes}$ and $\mathfrak{P}\mathbf{qDiLL}_0^{\boxtimes}$ are defined as before (Def. 12 and 17, respectively),⁹ except that the image of $\mathfrak{P}\mathbf{qDiLL}_0^{\boxtimes}$ on the generator $?_i$ (Figure 7d) is changed to



⁹ Remember that now, for Γ a list of lists of MELL formulæ, $\mathbf{qMELL}^{\boxtimes}(\Gamma)$ (resp. $\mathbf{qDiLL}_0^{\boxtimes}(\Gamma)$) is the set of MELL^{\boxtimes} (resp. $\text{DiLL}_0^{\boxtimes}$) quasi-proof-structures of type Γ , possibly with non-atomic axioms.

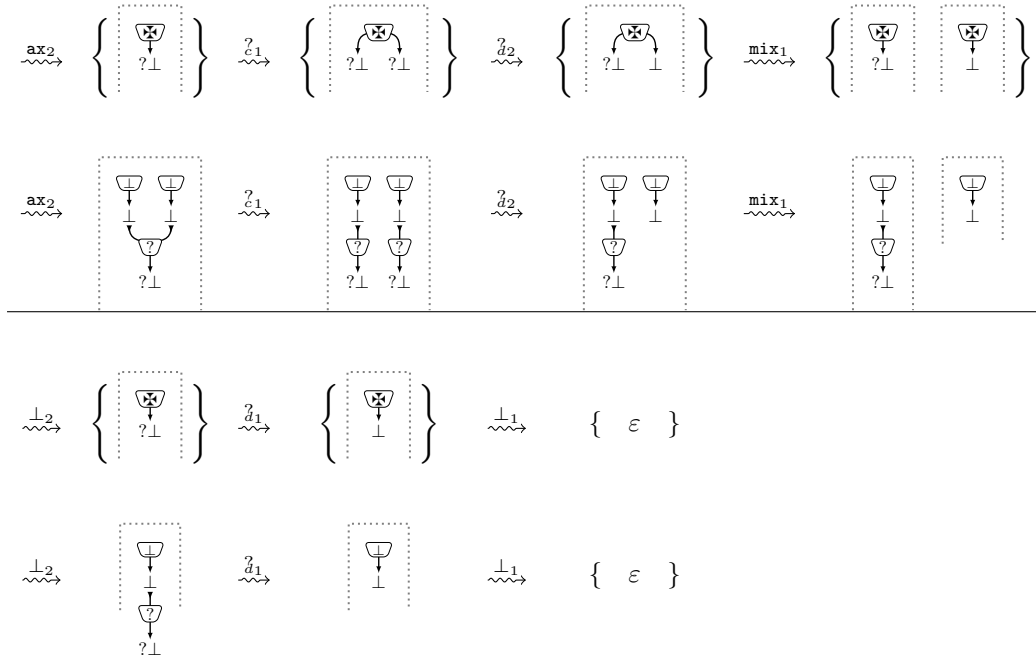


Figure 9 The path $\text{Box}_2 \overset{?}{d}_1 \text{Box}_2 \overset{?}{d}_2 \text{mix}_1 \text{ax}_2 \overset{?}{c}_1 \overset{?}{d}_2 \text{mix}_1 \perp_2 \overset{?}{d}_1 \perp_1$ witnessing that $\rho \in \mathcal{T}(R)$ (continued from Figure 8).

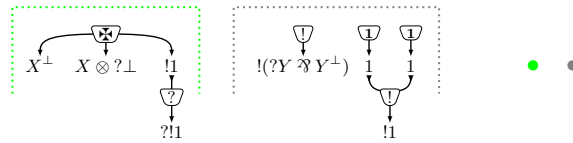


Figure 10 An element of the η -filled Taylor expansion of the MELL quasi-proof-structure in Fig. 2.

where $?\Gamma_k$ signifies that some of the conclusions of Γ_k might be connected to the \boxtimes -cell through a $?$ -cell. We can prove similarly our main results.

► **Theorem 23** (naturality with η). *The η -filled Taylor expansion defines a natural transformation $\mathfrak{T}_\eta^\boxtimes : \mathfrak{PqDiLL}_0^\boxtimes \Rightarrow \mathfrak{qMELL}^\boxtimes : \mathbf{Path} \rightarrow \mathbf{Rel}$ by: $(\Pi, R) \in \mathfrak{T}_\eta^\boxtimes \Gamma$ iff $\Pi \subseteq \mathcal{T}_\eta^\boxtimes(R)$ and the type of R is Γ . Moreover, if Π is a set of DiLL_0 proof-structures with $\Pi \xrightarrow{\xi} \Pi'$ and $\Pi' \subseteq \mathcal{T}(R')$, then R is a MELL proof-structure and $\Pi \subseteq \mathcal{T}(R)$, where R is such that $R \xrightarrow{\xi} R'$.*

► **Theorem 24** (glueability criterion with η). *Let Π be a set of DiLL_0 proof-structures, not necessarily with atomic axioms: Π is glueable iff $\Pi \subseteq \mathcal{T}(R)$ for some MELL proof-structure R .*

8 Conclusions and perspectives

\boxtimes -cells inside boxes. Our glueability criterion (Theorem 20) solves the inverse Taylor expansion problem in a “asymmetric” way: we characterize the sets of DiLL_0 proof-structures that are included in the Taylor expansion of some MELL proof-structure, but DiLL_0 proof-structures have no occurrences of \boxtimes -cells, while a MELL proof-structure possibly contains \boxtimes -cells inside boxes (see Definition 1). Not only this asymmetry is technically inevitable, but

it reflects on the fact that some glueable set of DiLL_0 proof-structures might not contain any information on the content of some box (which is reified in MELL by a \boxtimes -cell), or worse that, given the types, no content can fill that box. Think of the DiLL_0 proof-structure ρ made only of a $!$ -cell with no inputs and one output of type $!X$, where X is atomic: $\{\rho\}$ is glueable but the only MELL proof-structure R such that $\{\rho\} \subseteq \mathcal{T}(R)$ is made of a box containing a \boxtimes -cell.

This asymmetry is also present in Pagani and Tasson's characterization [23], even if not particularly emphasized: their Theorem 2 (analogous to the left-to-right part of our Theorem 20) assumes not only that the rewriting starting from a finite set of DiLL_0 proof-structures terminates but also that it ends on a MELL proof-structure (without \boxtimes -cells, which ensures that there exists a MELL proof-structure without \boxtimes -cells filling all the empty boxes).

The λ -calculus, connectedness and coherence. Our rewriting system and glueability criterion should help to prove the existence of a binary coherence for elements of the Taylor expansion of a fragment of MELL proof-structures (despite the impossibility for full MELL proved in [26]), extending the one that exists for resource λ -terms. We can remark that our glueability criterion is actually an extension of the criterion for resource λ -terms. Indeed, in the case of the λ -calculus, there are three rewrite steps, corresponding to abstraction, application and variable (which can be encoded in our rewrite steps), and coherence is defined inductively: if a set of resource λ -terms is coherent, then any set of resource λ -term that rewrites to it is also coherent.

Presented in this way, the main difference between the λ -calculus and MELL (concerning the inverse Taylor expansion problem) would not be because of the rewriting system but because the structure of any resource λ -term univocally determines the rewriting path, while, for DiLL_0 proof-structures, we have to quantify existentially over all possible paths. This is an unavoidable consequence of the fact that proof-structures do not have a tree-structure, contrary to λ -terms and resource λ -terms.

Moreover, it is possible to match and mix different sequences of rewriting. Indeed, consider three DiLL_0 proof-structures pairwise glueable. Proving that they are glueable as a whole amounts to computing a rewriting path from the rewriting paths witnessing the three glueabilities. Our paths were designed with that mixing-and-matching operation in mind, in the particular case where the boxes are connected. This is reminiscent of [16], where we also showed that a certain property enjoyed by the λ -calculus can be extended to proof-structures, provided they are connected inside boxes. We leave that work to a subsequent paper.

Functoriality and naturality. Our functorial point of view on proof-structures might unify many results. Let us cite two of them:

- a sequent calculus proof of $\vdash \Gamma$ can be translated into a path from the empty sequence into Γ . This could be the starting point for the formulation of a new correctness criterion;
- the category **Path** can be extended with higher structure, allowing to represent cut-elimination. The functors $\mathbf{qMELL}^{\boxtimes}$ and $\mathfrak{P}\mathbf{qDiLL}_0^{\boxtimes}$ can also be extended to such higher functors, proving via naturality that cut-elimination and the Taylor expansion commute.

References

- 1 Denis B echet. Minimality of the correctness criterion for multiplicative proof nets. *Mathematical Structures in Computer Science*, 8(6):543–558, 1998. URL: <http://journals.cambridge.org/action/displayAbstract?aid=44779>.

- 2 Dennis V. Borisov and Yuri I. Manin. *Generalized Operads and Their Inner Cohomomorphisms*, volume 265 of *Progress in Mathematics*, pages 247–308. Birkhäuser Basel, Basel, 2007. doi:10.1007/978-3-7643-8608-5_4.
- 3 Pierre Boudes. Thick Subtrees, Games and Experiments. In *Typed Lambda Calculi and Applications, 9th International Conference (TLCA 2009)*, volume 5608 of *Lecture Notes in Computer Science*, pages 65–79. Springer, 2009. doi:10.1007/978-3-642-02273-9_7.
- 4 Pierre Boudes, Fanny He, and Michele Pagani. A characterization of the Taylor expansion of lambda-terms. In *Computer Science Logic 2013 (CSL 2013)*, volume 23 of *LIPICs*, pages 101–115. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPICs.CSL.2013.101.
- 5 Gérard Boudol. The Lambda-Calculus with Multiplicities (Abstract). In *4th International Conference on Concurrency Theory (CONCUR '93)*, volume 715 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 1993. doi:10.1007/3-540-57208-2_1.
- 6 Daniel de Carvalho. The Relational Model Is Injective for Multiplicative Exponential Linear Logic. In *25th Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *LIPICs*, pages 41:1–41:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPICs.CSL.2016.41.
- 7 Daniel de Carvalho. Taylor expansion in linear logic is invertible. *Logical Methods in Computer Science*, 14(4), 2018. doi:10.23638/LMCS-14(4:21)2018.
- 8 Daniel de Carvalho and Lorenzo Tortora de Falco. The relational model is injective for multiplicative exponential linear logic (without weakenings). *Annals of Pure and Applied Logic*, 163(9):1210–1236, 2012. doi:10.1016/j.apal.2012.01.004.
- 9 Thomas Ehrhard. On Köthe sequence spaces and linear logic. *Mathematical Structures in Computer Science*, 12(5):579–623, 2002. doi:10.1017/S0960129502003729.
- 10 Thomas Ehrhard. Finiteness spaces. *Mathematical Structures in Computer Science*, 15(4):615–646, 2005. doi:10.1017/S0960129504004645.
- 11 Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1-3):1–41, 2003. doi:10.1016/S0304-3975(03)00392-X.
- 12 Thomas Ehrhard and Laurent Regnier. Böhm Trees, Krivine’s Machine and the Taylor Expansion of Lambda-Terms. In *Second Conference on Computability in Europe (CiE 2006)*, volume 3988 of *Lecture Notes in Computer Science*, pages 186–197. Springer, 2006. doi:10.1007/11780342_20.
- 13 Thomas Ehrhard and Laurent Regnier. Differential interaction nets. *Theoretical Computer Science*, 364(2):166–195, 2006. doi:10.1016/j.tcs.2006.08.003.
- 14 Thomas Ehrhard and Laurent Regnier. Uniformity and the Taylor expansion of ordinary lambda-terms. *Theoretical Computer Science*, 403(2-3):347–372, 2008. doi:10.1016/j.tcs.2008.06.001.
- 15 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987. doi:10.1016/0304-3975(87)90045-4.
- 16 Giulio Guerrieri, Luc Pellissier, and Lorenzo Tortora de Falco. Computing Connected Proof(-Structure)s From Their Taylor Expansion. In *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*, volume 52 of *LIPICs*, pages 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPICs.FSCD.2016.20.
- 17 Giulio Guerrieri, Luc Pellissier, and Lorenzo Tortora de Falco. Glueability of resource proof-structures: inverting the Taylor expansion (long version). *CoRR*, abs/1910.07936, 2019. arXiv:1910.07936.
- 18 Giulio Guerrieri, Luc Pellissier, and Lorenzo Tortora de Falco. Proof-Net as Graph, Taylor Expansion as Pullback. In *Logic, Language, Information, and Computation - 26th International Workshop (WoLLIC 2019)*, volume 11541 of *Lecture Notes in Computer Science*, pages 282–300. Springer, 2019. doi:10.1007/978-3-662-59533-6_18.

- 19 Emma Kerinec, Giulio Manzonetto, and Michele Pagani. Revisiting call-by-value Böhm trees in light of their Taylor expansion. *CoRR*, abs/1809.02659, 2018. URL: <http://arxiv.org/abs/1809.02659>.
- 20 Giulio Manzonetto and Domenico Ruoppolo. Relational Graph Models, Taylor Expansion and Extensionality. *Electronic Notes in Theoretical Computer Science*, 308:245–272, 2014. doi:10.1016/j.entcs.2014.10.014.
- 21 Damiano Mazza and Michele Pagani. The Separation Theorem for Differential Interaction Nets. In *Logic for Programming, Artificial Intelligence, and Reasoning, 14th International Conference (LPAR 2007)*, volume 4790 of *Lecture Notes in Computer Science*, pages 393–407. Springer, 2007. doi:10.1007/978-3-540-75560-9_29.
- 22 Damiano Mazza, Luc Pellissier, and Pierre Vial. Polyadic approximations, fibrations and intersection types. *PACMPL*, 2(POPL):6:1–6:28, 2018. doi:10.1145/3158094.
- 23 Michele Pagani and Christine Tasson. The Inverse Taylor Expansion Problem in Linear Logic. In *24th Annual Symposium on Logic in Computer Science (LICS 2009)*, pages 222–231. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.35.
- 24 Michele Pagani, Christine Tasson, and Lionel Vaux. Strong Normalizability as a Finiteness Structure via the Taylor Expansion of λ -terms. In *Foundations of Software Science and Computation Structures - 19th International Conference (FOSSACS 2016)*, volume 9634 of *Lecture Notes in Computer Science*, pages 408–423. Springer, 2016. doi:10.1007/978-3-662-49630-5_24.
- 25 Christian Retoré. A Semantic Characterisation of the Correctness of a Proof Net. *Mathematical Structures in Computer Science*, 7(5):445–452, 1997. doi:10.1017/S096012959700234X.
- 26 Christine Tasson. *Sémantiques et Syntaxes Vectorielles de la Logique Linéaire*. PhD thesis, Université Paris Diderot, France, December 2009. URL: <https://tel.archives-ouvertes.fr/tel-00440752>.
- 27 Lionel Vaux. Taylor Expansion, lambda-Reduction and Normalization. In *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, volume 82 of *LIPICs*, pages 39:1–39:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPICs.CSL.2017.39.

On the Union Closed Fragment of Existential Second-Order Logic and Logics with Team Semantics

Matthias Hoelzel

Mathematical Foundations of Computer Science, RWTH Aachen University, Aachen, Germany
hoelzel@logic.rwth-aachen.de

Richard Wilke

Mathematical Foundations of Computer Science, RWTH Aachen University, Aachen, Germany
wilke@logic.rwth-aachen.de

Abstract

We present syntactic characterisations for the union closed fragments of existential second-order logic and of logics with team semantics. Since union closure is a semantical and undecidable property, the normal form we introduce enables the handling and provides a better understanding of this fragment. We also introduce inclusion-exclusion games that turn out to be precisely the corresponding model-checking games. These games are not only interesting in their own right, but they also are a key factor towards building a bridge between the semantic and syntactic fragments. On the level of logics with team semantics we additionally present restrictions of inclusion-exclusion logic to capture the union closed fragment. Moreover, we define a team based atom that when adding it to first-order logic also precisely captures the union closed fragment of existential second-order logic which answers an open question by Galliani and Hella.

2012 ACM Subject Classification Theory of computation → Higher order logic

Keywords and phrases Higher order logic, Existential second-order logic, Team semantics, Closure properties, Union closure, Model-checking games, Syntactic characterisations of semantical fragments

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.25

Related Version A full version of the paper is available at <https://arxiv.org/abs/1910.06057>.

Funding *Matthias Hoelzel*: This author is supported by the German Research Foundation (DFG).
Richard Wilke: This author is supported by the DFG, Research Training Group 2236 UnRAVeL.

1 Introduction

One branch of model theory engages with the characterisation of semantical fragments, which typically are undecidable, as syntactical fragments of the logics under consideration. Prominent examples are van Benthem's Theorem characterising the bisimulation invariant fragment of first-order logic as the modal-logic [10] or preservation theorems like the Łoś-Tarski Theorem, which states that formulae preserved in substructures are equivalent to universal formulae [6]. In this paper we consider formulae $\varphi(X)$ of existential second-order logic, Σ_1^1 , in a free relational variable X and investigate the property of being closed under unions, meaning that whenever a family of relations X_i all satisfy φ , then their union $\bigcup_i X_i$ should also do so. Certainly closure under unions is an undecidable property. We provide a syntactical characterisation of all formulae of existential second-order logic obeying this property via a normal form called *myopic*- Σ_1^1 , a notion based on ideas of Galliani and Hella [2]. By Fagin's Theorem, Σ_1^1 is the logical equivalent of the complexity class NP which highlights the importance to understand its fragments. Towards this end we employ game theoretic concepts and introduce a novel game type, called *inclusion-exclusion games*, suited for formulae $\varphi(X)$ with a free relational variable. In these games a strategy no longer is



© Matthias Hoelzel and Richard Wilke;
licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 25; pp. 25:1–25:16



Leibniz International Proceedings in Informatics

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

simply winning for one player – and hence proving whether a sentence is satisfied – but it is moreover adequate for a certain relation Y over \mathfrak{A} showing that the formula is satisfied by \mathfrak{A} and Y , in symbols $\mathfrak{A} \models \varphi(Y)$. We construct myopic- Σ_1^1 formulae that can define the winning regions of specifically those inclusion-exclusion games that are (semantically) closed under unions. Conceptually such games are eligible for any Σ_1^1 -formula, but since our interest lies in those formulae that are closed under unions, we introduce a restricted version of such games, called *union games*, that precisely correspond to the model-checking games of union closed Σ_1^1 -formulae. Consequently, the notion of union closure is captured on the level of formulae by the myopic fragment of Σ_1^1 and on the game theoretic level by union games.

Existential second-order logic has a tight connection to modern logics of dependence and independence that are based on the concept of teams, introduced by Hodges [7], and later refined by Väänänen in 2007 [9]. In contrast to classical logics, formulae of such a logic are evaluated against a set of assignments, called a *team*. One main characteristic of these logics is that dependencies between variables, such as “ x depends solely on y ”, are expressed as atomic properties of teams. Widely used dependency atoms include dependence ($=(x, y)$), inclusion ($x \subseteq y$), exclusion ($x \mid y$) and independence ($x \perp y$). It is known that both independence logic $\text{FO}(\perp)$ and inclusion-exclusion logic $\text{FO}(\subseteq, \mid)$ have the same expressive power as full existential second-order logic Σ_1^1 [1]. The team in such logics corresponds to the free relational variable in existential second-order formulae, enabling us to ask the same questions about fragments with certain closure properties in both frameworks. One example of a well understood closure property is *downwards closure* stating that if a formula is satisfied by a team then it is also satisfied by all subteams (i.e. subsets of that team). It is well known that exclusion logic $\text{FO}(\mid)$ corresponds to the downwards closed fragment of Σ_1^1 [1, 8]. The issue of union closure is different. Galliani and Hella have shown that inclusion logic $\text{FO}(\subseteq)$ corresponds to greatest fixed-point-logic GFP^+ and, hence, by using the Immerman-Vardi Theorem, it captures all PTIME computable queries on ordered structures [2]. They also proved that every union closed dependency notion that itself is first-order definable (where the formula has access to a predicate for the team) is already definable in inclusion logic. However, there are union closed properties that are not definable in inclusion logic (think of a union closed NP property). For a concrete example we refer to the atom \mathcal{R} from [2]. Thus Galliani and Hella asked the question whether there is a union closed atomic dependency notion β , such that the logic $\text{FO}(\beta)$ captures precisely the union closed fragment of $\text{FO}(\subseteq, \mid)$. In the present work we answer this question positively with the aid of inclusion-exclusion games. Furthermore, we present a syntactical restriction of all $\text{FO}(\subseteq, \mid)$ formulae that also precisely describe the union closed fragment. This syntactical fragment corresponds to myopic- Σ_1^1 and is in harmony with the game theoretical view, which is described by union games.

Sections 3, 4 and 5 deal with second-order logic, while the other sections, 6 and 7, address logics with team semantics. In section three the central notion of this paper, inclusion-exclusion games, are introduced, which are used in section four to characterise the union closed fragment within existential second-order logic. Section five provides a restriction of the games specifically suited for this fragment. The sections dealing with team semantics can be read mostly independently of each other. Based on section four, section six describes the union closed fragment of inclusion-exclusion logic in terms of syntactical restrictions. The question of Galliani and Hella, whether there is a union closed atom that constitutes the union closed fragment, is answered positively in section seven, for which the reader should be familiar with union games introduced in section five.

Omitted proofs can be found in the full version or can be done without much effort.

2 Preliminaries

We assume familiarity with first-order logic and existential second-order logic, FO and Σ_1^1 for short. For a background we refer to the textbook [4].

The neighbourhood of a vertex v in a graph G is denoted by $N_G(v)$. For a given τ -structure \mathfrak{A} and formula $\varphi(\bar{x})$ we define $\varphi^{\mathfrak{A}} := \{\bar{a} : \mathfrak{A} \models \varphi(\bar{a})\}$, $\text{free}(\varphi)$ is the set of free first-order variables and $\text{subf}(\psi)$ is the set of subformulae of ψ . Notations like \bar{v}, \bar{w} always indicate that $\bar{v} = (v_1, \dots, v_k)$ and $\bar{w} = (w_1, \dots, w_\ell)$ are some (finite) tuples. Here $k = |\bar{v}|$ and $\ell = |\bar{w}|$, so \bar{v} is a k -tuple while \bar{w} is an ℓ -tuple. We write $\{\bar{v}\}$ or $\{\bar{v}, \bar{w}\}$ as abbreviations for $\{v_1, \dots, v_k\}$ resp. $\{v_1, \dots, v_k, w_1, \dots, w_\ell\}$ while $\{(\bar{v}), (\bar{w})\}$ is the set consisting of the two tuples \bar{v} and \bar{w} (as elements). The *concatenation* of \bar{v} and \bar{w} is $(\bar{v}, \bar{w}) := (v_1, \dots, v_k, w_1, \dots, w_\ell)$. The power set of a set A is denoted by $\mathcal{P}(A)$ and $\mathcal{P}^+(A) := \mathcal{P}(A) \setminus \{\emptyset\}$.

Team Semantics. A *team* X over \mathfrak{A} is a *set* of assignments mapping a common domain $\text{dom}(X) = \{\bar{x}\}$ of variables into A .¹ The restriction of X to some first-order formula $\varphi(\bar{x})$ is $X|_{\varphi} := \{s \in X : \mathfrak{A} \models_s \varphi\}$. For a given subtuple $\bar{y} = (y_1, \dots, y_\ell) \subseteq \bar{x}$ and every $s \in X$ we define $s(\bar{y}) := (s(y_1), \dots, s(y_\ell))$. Furthermore, we frequently use $X(\bar{y}) := \{s(\bar{y}) : s \in X\}$, which is an ℓ -ary relation over \mathfrak{A} . For an assignment s , a variable x and $a \in A$ we use $s[x \mapsto a]$ to denote the assignment resulting from s by adding x to its domain (if it is not already contained) and declaring a as the image of x .

► **Definition 1.** Let \mathfrak{A} be a τ -structure, X a team of \mathfrak{A} . In the following λ denotes a first-order τ -literal and φ, ψ arbitrary formulae in negation normal form.

- $\mathfrak{A} \models_X \lambda \iff \mathfrak{A} \models_s \lambda$ for all $s \in X$
- $\mathfrak{A} \models_X \varphi \wedge \psi \iff \mathfrak{A} \models_X \varphi$ and $\mathfrak{A} \models_X \psi$
- $\mathfrak{A} \models_X \varphi \vee \psi \iff \mathfrak{A} \models_Y \varphi$ and $\mathfrak{A} \models_Z \psi$ for some $Y, Z \subseteq X$ such that $Y \cup Z = X$
- $\mathfrak{A} \models_X \forall x \varphi \iff \mathfrak{A} \models_{X[x \mapsto A]} \varphi$
- $\mathfrak{A} \models_X \exists x \varphi \iff \mathfrak{A} \models_{X[x \mapsto F]} \varphi$ for some $F: X \rightarrow \mathcal{P}^+(A)$

Here $X[x \mapsto A] := \{s[x \mapsto a] : s \in X, a \in A\}$ and $X[x \mapsto F] := \{s[x \mapsto a] : s \in X, a \in F(s)\}$.

Team semantics for a first-order formula φ (without any dependency concepts) boils down to evaluating φ against every single assignment, i.e. more formally we have $\mathfrak{A} \models_X \varphi \iff \mathfrak{A} \models_s \varphi$ for every $s \in X$ (in usual Tarski semantics). This is also known as the flatness property of FO. The reason for considering teams instead of single assignments is that they allow the formalisation of dependency statements in the form of dependency atoms. Among the most common atoms are the following.

- $\mathfrak{A} \models_X =(\bar{x}, \bar{y}) \iff s(\bar{x}) = s'(\bar{x})$ implies $s(\bar{y}) = s'(\bar{y})$ for all $s, s' \in X$
- $\mathfrak{A} \models_X \bar{x} \subseteq \bar{y} \iff X(\bar{x}) \subseteq X(\bar{y})$
- $\mathfrak{A} \models_X \bar{x} \perp \bar{y} \iff X(\bar{x}) \cap X(\bar{y}) = \emptyset$
- $\mathfrak{A} \models_X \bar{x} \perp \bar{y} \iff$ for all $s, s' \in X$ exists $s'' \in X$ s.t. $s(\bar{x}) = s''(\bar{x})$ and $s'(\bar{y}) = s''(\bar{y})$

These are called *dependence* [9], *inclusion*, *exclusion* [1] and *independence* [5] atoms, respectively. When we speak about a logic that may use certain atomic dependency notions, for example inclusion, we denote it by writing FO(\subseteq) and so forth. These logics have the empty team property, which means that $\mathfrak{A} \models_{\emptyset} \varphi$ is always true. This is also the reason why *sentences* are not evaluated against \emptyset but rather against $\{\emptyset\}$, which is the team consisting

¹ We use the corresponding Latin letters to denote universes of structures.

25:4 On the Union Closed Fragment

of the empty assignment. Let φ be a first-order formula and ψ be any formula of a logic with team semantics. We define $\varphi \rightarrow \psi$ as $\text{nmf}(\neg\varphi) \vee (\varphi \wedge \psi)$ where $\text{nmf}(\neg\varphi)$ is the negation normal form of $\neg\varphi$. It is easy to see that $\mathfrak{A} \models_X \varphi \rightarrow \psi \iff \mathfrak{A} \models_{X \upharpoonright \varphi} \psi$.

Union Closure. A formula φ of a logic with team semantics is said to be *union closed* if $\mathfrak{A} \models_{X_i} \varphi$ for all $i \in I$ implies $\mathfrak{A} \models_X \varphi$, where $X = \bigcup_{i \in I} X_i$. Analogously, a formula $\varphi(X)$ of Σ_1^1 with a free relational variable X is union closed if $\mathfrak{A} \models \varphi(X_i)$ for all i implies $\mathfrak{A} \models \varphi(X)$.

FO Interpretations. A first-order interpretation from σ to τ of arity k is a sequence $\mathcal{I} = (\delta, \varepsilon, (\psi_S)_{S \in \tau})$ of FO(σ)-formulae, called the domain, equality and relation formulae respectively. We say that \mathcal{I} interprets a τ -structure \mathfrak{B} in some σ -structure \mathfrak{A} and write $\mathfrak{B} \cong \mathcal{I}(\mathfrak{A})$ if and only if there exists a surjective function h , called the coordinate map, that maps $\delta^{\mathfrak{A}} = \{\bar{a} \in A^k : \mathfrak{A} \models \delta(\bar{a})\}$ to B preserving and reflecting the equalities and relations provided by ε and ψ_S , such that h induces an isomorphism between the quotient structure $(\delta^{\mathfrak{A}}, (\psi_S^{\mathfrak{A}})_{S \in \tau}) / \varepsilon^{\mathfrak{A}}$ and \mathfrak{B} . A more detailed explanation can be found in [4]. For a τ -formula φ we associate the σ -formula $\varphi^{\mathcal{I}}$ by relativising quantifiers to δ , using ε as equality and ψ_S instead of S . We extend this translation to Σ_1^1 by the following rules for additional free/quantified relation symbols S .

- $(\exists S \vartheta)^{\mathcal{I}} := \exists S^* (\forall \bar{x}_1 \cdots \bar{x}_{\text{ar}(S)} (S^* \bar{x}_1 \cdots \bar{x}_{\text{ar}(S)} \rightarrow \bigwedge_{j=1}^{\text{ar}(S)} \delta(\bar{x}_j)) \wedge \vartheta^{\mathcal{I}}),$
- $(S v_1 \cdots v_{\text{ar}(S)})^{\mathcal{I}} := \exists \bar{w}_1 \cdots \bar{w}_{\text{ar}(S)} (\bigwedge_{j=1}^{\text{ar}(S)} (\delta(\bar{w}_j) \wedge \varepsilon(\bar{v}_j, \bar{w}_j)) \wedge S^* \bar{w}_1 \cdots \bar{w}_{\text{ar}(S)}).$

An assignment $s: \{\bar{x}_1, \dots, \bar{x}_m\} \rightarrow A$ is well-formed (w.r.t. \mathcal{I}), if $s(\bar{x}_i) \in \delta^{\mathfrak{A}}$ ($= \text{dom}(h)$) for every $i = 1, \dots, m$. Such an assignment encodes $h \circ s: \{x_1, \dots, x_m\} \rightarrow B$ with $(h \circ s)(x_i) := h(s(\bar{x}_i))$ which is an assignment over \mathfrak{B} . Similarly, a relation Q is well-formed (w.r.t. \mathcal{I}), if $Q \subseteq (\delta^{\mathfrak{A}})^\ell$ where $\ell = \frac{\text{ar}(Q)}{k} \in \mathbb{N}$, and we define $h(Q) := \{(h(\bar{a}_1), \dots, h(\bar{a}_\ell)) : (\bar{a}_1, \dots, \bar{a}_\ell) \in Q\}$, which is the ℓ -ary relation over \mathfrak{B} that was described by Q . The connection between $\varphi^{\mathcal{I}}$ and φ is made precise in the well-known interpretation lemma.

► **Lemma 2** (Interpretation Lemma for Σ_1^1). *Let $\varphi(S_1, \dots, S_n) \in \Sigma_1^1$. Let $R_i^* \subseteq A^{k \cdot \text{ar}(S_i)}$ for all i and $s: \{\bar{x}_1, \dots, \bar{x}_m\} \rightarrow A$ be well-formed. Then: $(\mathfrak{A}, R_1^*, \dots, R_n^*) \models_s \varphi^{\mathcal{I}} \iff (\mathfrak{B}, h(R_1^*), \dots, h(R_n^*)) \models_{h \circ s} \varphi$.*

3 Inclusion-Exclusion Games

Classical model-checking games are designed to express satisfiability of sentences, i.e. formulae without free variables. Since our focus lies on formulae in a free relational variable we are in need for a game that is able to not only express that a formula is satisfied, but moreover that it is satisfied by a certain relation. In the games we are about to describe a set of designated positions is present – called the *target set* – which corresponds to the full relation A^k (where the free relational variable has arity k). A winning strategy is said to be *adequate* for a subset X of the target positions, if the target vertices visited by it are X . On the level of logics this matches the relation satisfying the corresponding formula, i.e. there is a winning strategy adequate for X if and only if the formula is satisfied by X .

An *inclusion-exclusion game* $\mathcal{G} = (V, V_0, V_1, E, I, T, E_{\text{ex}})$ is played by two players 0 and 1 where

- V_σ is the set of vertices of player σ ,
- $V = V_0 \cup V_1$,
- $E \subseteq V \times V$ is a set of possible moves,
- $I \subseteq V$ is the (possibly empty) set of initial positions,

- $T \subseteq V$ is the set of target vertices and
- $E_{\text{ex}} \subseteq V \times V$ is the exclusion condition, which defines the winning condition for player 0.² The edges going into T , that is $E_{\text{in}} := E \cap (V \times T)$, are called *inclusion edges*, while E_{ex} is the set of *exclusion edges* (sometimes also called conflicting pairs). Inclusion-exclusion games are *second-order* games, so instead of single plays we are more interested in sets of plays that are admitted by some winning strategy for player 0.

For a subset $X \subseteq T$ the aim of player 0 is to provide a winning strategy (which can be viewed as a set of plays respecting the exclusion condition and containing all possible strategies of player 1) such that the vertices of T that are visited by this strategy correspond precisely to X .

► **Definition 3.** A winning strategy (for player 0) \mathcal{S} is a possibly empty subgraph $\mathcal{S} = (W, F)$ of $G = (V, E)$ ensuring the following four consistency conditions.

- (i) For every $v \in W \cap V_0$ holds $N_{\mathcal{S}}(v) \neq \emptyset$.
- (ii) For every $v \in W \cap V_1$ holds $N_{\mathcal{S}}(v) = N_G(v)$.
- (iii) $I \subseteq W$.
- (iv) $(W \times W) \cap E_{\text{ex}} = \emptyset$.

Intuitively, the conditions (i) and (ii) state that the strategy must provide at least one move from each node of player 0 used by the strategy but does not make assumptions about the moves that player 1 may make whenever the strategy plays a node belonging to player 1. In particular, the strategy must not play any terminal vertices that are in V_0 . Furthermore, (iii) enforces that at least the initial vertices are contained while (iv) disallows playing with conflicting pairs $(v, w) \in E_{\text{ex}}$, i.e. v and w must not coexist in any winning strategy for player 0. If $I = \emptyset$, then (\emptyset, \emptyset) is the trivial winning strategy. Since we do *not* have a notion for a winning strategy for player 1, inclusion-exclusion games can be viewed as solitaire games.

Of course, the winning condition of an inclusion-exclusion game \mathcal{G} is first-order definable. The formula $\varphi_{\text{win}}(W, F)$ has the property that $\mathcal{G} \models \varphi_{\text{win}}(W, F)$ if and only if (W, F) is a winning strategy for player 0 in \mathcal{G} , where

$$\begin{aligned} \varphi_{\text{win}}(W, F) := & \forall v (Wv \rightarrow ((V_0v \wedge \exists w (Evw \wedge Ww \wedge Fvw)) \vee \\ & (V_1v \wedge \forall w (Evw \rightarrow Ww \wedge Fvw)))) \wedge \\ & \forall v (Iv \rightarrow Wv) \wedge \forall v \forall w ((Wv \wedge Ww) \rightarrow \neg E_{\text{ex}}vw) \end{aligned}$$

describes the winning condition imposed on the graph (W, F) .

We are mainly interested in the subset of target vertices that are visited by a winning strategy $\mathcal{S} = (W, F)$. More formally, \mathcal{S} induces $\mathcal{T}(\mathcal{S}) := W \cap T$, which we also call the *target* of \mathcal{S} . This allows us to associate with every inclusion-exclusion game \mathcal{G} the set of targets of winning strategies: $\mathcal{T}(\mathcal{G}) := \{\mathcal{T}(\mathcal{S}) : \mathcal{S} \text{ is a winning strategy for player 0 in } \mathcal{G}\}$.

Intuitively, as already pointed out, games of this kind will be the model-checking games for Σ_1^1 -formulae $\varphi(X)$ that have a free relational variable X . Given a structure \mathfrak{A} and such a formula, we are interested in the possible relations Y that satisfy the formula, in symbols $(\mathfrak{A}, Y) \models \varphi(X)$. We will construct the game such that Y satisfies φ if and only if there is a strategy of player 0 winning for the set $Y \subseteq T$, thus $\mathcal{T}(\mathcal{G}) = \{Y : (\mathfrak{A}, Y) \models \varphi\}$. It will be more convenient for our purposes that the target vertices of an inclusion-exclusion game are not required to be terminal positions. However it would be no restriction as it is easy to transform any given game into one that agrees on the (possible) targets, in which all target vertices are terminal.

² E_{ex} can always be replaced by the symmetric closure of E_{ex} without altering its semantics.

25:6 On the Union Closed Fragment

One can reduce the satisfiability problem of propositional logic to deciding whether player 0 has a winning strategy in an inclusion-exclusion game.

► **Theorem 4.** *The problem of deciding whether $X \in \mathcal{T}(\mathcal{G})$ for a finite inclusion-exclusion game \mathcal{G} is NP-COMplete.*

3.1 Model-Checking Games for Existential Second-Order Logic

In this section we define model-checking games for formulae $\varphi(X) \in \Sigma_1^1$ with a free relational variable. These games are inclusion-exclusion games whose target sets are precisely the sets of relations that satisfy $\varphi(X)$.

► **Definition 5.** *Let \mathfrak{A} be a τ -structure and $\varphi(X) = \exists \bar{R} \varphi'(X, \bar{R}) \in \Sigma_1^1$ (in negation-normal form) where $\varphi'(X, \bar{R}) \in \text{FO}(\tau \cup \{X, \bar{R}\})$ using a free relation symbol X of arity $r := \text{ar}(X)$. The game $\mathcal{G}_X(\mathfrak{A}, \varphi) := (V, V_0, V_1, E, I, T, E_{\text{ex}})$ consists of the following components:*

- $V := \{(\vartheta, s) : \vartheta \in \text{subf}(\varphi'), s : \text{free}(\vartheta) \rightarrow A\} \cup A^r, I := \{(\varphi', \emptyset)\}, T := A^r,$
- $V_1 := \{(\vartheta, s) : \vartheta = \forall y \gamma \text{ or } \vartheta = \gamma_1 \wedge \gamma_2\} \cup \{(\gamma, s) : \gamma \text{ is a } \tau\text{-literal and } \mathfrak{A} \models_s \gamma\} \cup \{(\gamma, s) : \gamma = \neg X \bar{x} \text{ or } \gamma \text{ is a } \{\bar{R}\}\text{-literal}\} \cup T, V_0 := V \setminus V_1,$
- $E := \{((\gamma \circ \vartheta, s), (\delta, s \upharpoonright_{\text{free}(\delta)})) : \circ \in \{\wedge, \vee\}, \delta \in \{\gamma, \vartheta\}\} \cup \{((X \bar{x}, s), s(\bar{x})) : X \bar{x} \in \text{subf}(\varphi')\} \cup \{((Qx\gamma, s), (\gamma, s')) : Q \in \{\exists, \forall\}, s' = s[x \mapsto a], a \in A\},$
- $E_{\text{ex}} := \{((R_i \bar{x}, s), (\neg R_i \bar{y}, s')) : s(\bar{x}) = s'(\bar{y})\} \cup \{((\neg X \bar{x}, s), \bar{a}) : s(\bar{x}) = \bar{a}\}.$

These games capture the behaviour of existential second-order formulae which provides us with the following theorem.

► **Theorem 6.** $(\mathfrak{A}, X) \models \varphi(X) \iff$ *Player 0 has a winning strategy \mathcal{S} in $\mathcal{G}_X(\mathfrak{A}, \varphi)$ with $\mathcal{T}(\mathcal{S}) = X$. Or, in other words: $\mathcal{T}(\mathcal{G}_X(\mathfrak{A}, \varphi)) = \{X \subseteq A^r : (\mathfrak{A}, X) \models \varphi(X)\}.$*

4 Characterising the Union Closed Formulae within Existential Second-Order Logic

In this section we investigate formulae $\varphi(X)$ of existential second-order logic that are closed under unions with respect to their free relational variable X . Union closure, being a semantical property of formulae, is undecidable. However, we present a *syntactical* characterisation of all such formulae via the following normal form.

► **Definition 7.** *A formula $\varphi(X) \in \Sigma_1^1$ is called myopic if $\varphi(X) = \forall \bar{x} (X \bar{x} \rightarrow \exists \bar{R} \varphi'(X, \bar{R}))$, where $\varphi' \in \text{FO}$ and X occurs only positively³ in φ' .*

Variants of myopic formulae have already been considered for first-order logic [2, Definition 19] and for greatest fixed-point logics [3, Theorem 24 and Theorem 26], but to our knowledge myopic Σ_1^1 -formulae have not been studied so far.

Let \mathcal{U} denote the set of all union closed Σ_1^1 -formulae. To establish the claim that myopic formulae are a normal form of \mathcal{U} we need to show that all myopic formulae are indeed closed under unions and, more importantly, that every union closed formula can be translated into an equivalent myopic formula. This translation is in particular constructive.

³ That is under an even number of negations.

► **Theorem 8.** $\varphi(X) \in \Sigma_1^1$ is union closed if and only if $\varphi(X)$ is equivalent to some myopic Σ_1^1 -formula.

We split the proof into two parts, the direction from right to left is handled in Proposition 9 and from left to right in Theorem 11.

► **Proposition 9.** *Every myopic formula is union closed.*

Proof. Let $\varphi = \forall \bar{x}(X\bar{x} \rightarrow \exists \bar{R}\varphi'(X, \bar{R}))$ and $(\mathfrak{A}, X_i) \models \varphi$ for all $i \in I$. We claim that $(\mathfrak{A}, X) \models \varphi$ for $X = \bigcup_{i \in I} X_i$. Let $\bar{a} \in X_i \subseteq X$. By assumption $(\mathfrak{A}, X_i) \models_{\bar{x} \mapsto \bar{a}} \exists \bar{R}\varphi'(X, \bar{R})$. A fortiori (X occurs only positively in φ'), we obtain $(\mathfrak{A}, X) \models_{\bar{x} \mapsto \bar{a}} \exists \bar{R}\varphi'(X, \bar{R})$. Since \bar{a} was chosen arbitrarily, this property holds for all $\bar{a} \in X$, hence the claim follows. ◀

For a fixed formula $\varphi(X)$ the corresponding game \mathcal{G}_X can be constructed by a first-order interpretation depending of course on the current structure.

► **Lemma 10.** *Let $\varphi(X) = \exists \bar{R}\varphi'(X, \bar{R}) \in \Sigma_1^1$ where $\varphi' \in \text{FO}(\tau \cup \{X, \bar{R}\})$ and $r := \text{ar}(X)$. Then there exists a quantifier-free interpretation \mathcal{I} such that $\mathcal{G}_X(\mathfrak{A}, \varphi) \cong \mathcal{I}(\mathfrak{A})$ for every structure \mathfrak{A} (with at least two elements).*

► **Theorem 11.** *For every union closed formula $\varphi(X) \in \Sigma_1^1$ there is an equivalent myopic formula $\mu(X) \in \Sigma_1^1$.*

Proof. Let $\varphi(X) = \exists \bar{R}\varphi'(X, \bar{R}) \in \Sigma_1^1(\tau)$ be closed under unions, \mathfrak{A} be a τ -structure and $\mathcal{G} := \mathcal{G}_X(\mathfrak{A}, \varphi)$ be the corresponding game. W.l.o.g. \mathfrak{A} has at least two elements. By Theorem 6, we have that $\mathcal{T}(\mathcal{G}) = \{Y \subseteq A^r : \mathfrak{A} \models \varphi(Y)\}$ where $r := \text{ar}(X)$. Since $\varphi(X)$ is union closed, it follows that $\mathcal{T}(\mathcal{G})$ is closed under unions as well. Now we observe that $\mathcal{T}(\mathcal{G})$ can be defined in the game \mathcal{G} by the following myopic formula:

$$\begin{aligned} \varphi_{\mathcal{T}}(X) &:= \forall x(Xx \rightarrow \psi_{\mathcal{T}}(X, x)) \text{ where} \\ \psi_{\mathcal{T}}(X, x) &:= \exists W \exists F(\varphi_{\text{win}}(W, F) \wedge Wx \wedge \forall y(Wy \wedge Ty \rightarrow Xy)) \end{aligned}$$

Here φ_{win} is the first-order formula verifying winning strategies. Please note that $\varphi_{\mathcal{T}}$ is indeed a myopic formula, since X occurs only positively in $\psi_{\mathcal{T}}$.

▷ **Claim 12.** For every $X \subseteq A^r$, $(\mathcal{G}, X) \models \varphi_{\mathcal{T}}(X) \iff X \in \mathcal{T}(\mathcal{G})$.

Proof. Assume that $(\mathcal{G}, X) \models \varphi_{\mathcal{T}}(X)$. By construction of $\varphi_{\mathcal{T}}$, for every $\bar{a} \in X$ there exists a winning strategy $\mathcal{S}_{\bar{a}} = (W_{\bar{a}}, F_{\bar{a}})$ with $\bar{a} \in W_{\bar{a}}$ and $\mathcal{T}(\mathcal{S}_{\bar{a}}) = W_{\bar{a}} \cap T \subseteq X$. It follows that $X = \bigcup_{\bar{a} \in X} \mathcal{T}(\mathcal{S}_{\bar{a}})$. Since $\mathcal{T}(\mathcal{G})$ is closed under unions, we also obtain that $X \in \mathcal{T}(\mathcal{G})$.

We want to remark that at this point the semantical property is translated into a syntactical one, as the formula only describes the correct winning strategy because the initial formula was closed under unions.

To conclude the proof of Claim 12, assume that $X \in \mathcal{T}(\mathcal{G})$. Then there exists a winning strategy $\mathcal{S} = (W, F)$ for player 0 with $\mathcal{T}(\mathcal{S}) = X$. Thus, for the quantifiers $\exists W \exists F$ we can (for all $\bar{a} \in X$) choose \mathcal{S} , which, obviously, satisfies the formula. ◀

There is a first-order interpretation \mathcal{I} (of arity $n+m$) with $\mathcal{I}(\mathfrak{A}) \cong \mathcal{G}$ for some coordinate map $h: \delta^{\mathfrak{A}} \rightarrow V(\mathcal{G})$ and for every $\bar{a} \in T(\mathcal{G})$, $h^{-1}(\bar{a}) = \{(\bar{u}, \bar{a}, \bar{b}) \in A^{n+m} : \mathfrak{A} \models e_T(\bar{u}), \bar{b} \in A^{m-r}\}$ where $e_T(x_1, \dots, x_n)$ is some quantifier-free first-order formula (for more details, we refer to the full version). By the interpretation lemma for Σ_1^1 (Lemma 2), for every $X \subseteq T(\mathcal{G})$,

$$(\mathfrak{A}, X^*) \models \varphi_{\mathcal{T}}^{\mathcal{I}}(X^*) \iff (\mathcal{G}, X) \models \varphi_{\mathcal{T}}(X) \quad (1)$$

25:8 On the Union Closed Fragment

where $X^* := h^{-1}(X)$ is a relation of arity $(n + m)$. Recall that every variable x occurring in $\varphi_{\mathcal{T}}$ is replaced by a tuple \bar{x} of length $(n + m)$. Let $\bar{x} = (\bar{u}, \bar{v}, \bar{w})$ where $|\bar{u}| = n$, $|\bar{v}| = r$ and $|\bar{w}| = m - r$ and let

$$\mu(X) := \forall \bar{v}(X\bar{v} \rightarrow \forall \bar{u}\forall \bar{w}(e_T(\bar{u}) \rightarrow \psi^*(X, \bar{u}, \bar{v}, \bar{w})))$$

where ψ^* is the formula that results from $\psi_{\mathcal{T}}^{\mathcal{I}}$ by replacing every occurrence of $X^*\bar{u}'\bar{v}'\bar{w}'$ (where $|\bar{u}'| = n$, $|\bar{v}'| = r$ and $|\bar{w}'| = m - r$) by the formula $e_T(\bar{u}') \wedge X\bar{v}'$. By construction, this is a myopic formula, because X occurred only positively in $\psi^{\mathcal{I}}$ and, hence, X^* (resp. X) occurs only positively in $\psi_{\mathcal{T}}^{\mathcal{I}}$ (resp. ψ^*).

Recall that, in the game $\mathcal{G} \cong \mathcal{I}(\mathfrak{A})$, every $X \subseteq T(\mathcal{G})$ is a unary relation over \mathcal{G} , while the elements of $T(\mathcal{G})$ themselves are r -tuples of A . Furthermore, we have that $h^{-1}(X) := \{(\bar{a}, \bar{b}, \bar{c}) \in A^n \times A^r \times A^{m-r} : \mathfrak{A} \models e_T(\bar{a}) \text{ and } \bar{b} \in X\}$. Because of this and $X^* = h^{-1}(X)$, it follows that for every $s: \{\bar{u}', \bar{v}', \bar{w}'\} \rightarrow A$ holds

$$(\mathfrak{A}, X^*) \models_s X^*\bar{u}'\bar{v}'\bar{w}' \iff \mathfrak{A} \models e_T(s(\bar{u}')) \text{ and } s(\bar{v}') \in X \iff (\mathfrak{A}, X) \models_s e_T(\bar{u}') \wedge X\bar{v}'. \quad (2)$$

By construction of ψ^* , these are the only subformulae in which $\psi_{\mathcal{T}}^{\mathcal{I}}$ and ψ^* differ from each other. As a result, the following claim is true:

▷ **Claim 13.** For every $X \subseteq A^r$ and every assignment $s: \text{free}(\psi_{\mathcal{T}}^{\mathcal{I}}) \rightarrow A$, holds

$$(\mathfrak{A}, X^*) \models_s \psi_{\mathcal{T}}^{\mathcal{I}}(X^*, \bar{x}) \iff (\mathfrak{A}, X) \models_s \psi^*(X, \bar{x}).$$

Recall that $\bar{x} = (\bar{u}, \bar{v}, \bar{w})$ where $|\bar{u}| = n$, $|\bar{v}| = r$ and $|\bar{w}| = (m - r)$. Now we can see that

$$\begin{aligned} (\mathfrak{A}, X^*) \models \varphi_{\mathcal{T}}^{\mathcal{I}} &= \forall \bar{x}(X^*\bar{x} \rightarrow \psi_{\mathcal{T}}^{\mathcal{I}}(X^*, \bar{x})) \\ &\iff (\mathfrak{A}, X^*) \models_s \psi_{\mathcal{T}}^{\mathcal{I}}(X^*, \bar{x}) \text{ for every } s \text{ with } s(\bar{x}) \in X^* \\ &\iff (\mathfrak{A}, X) \models_s \psi^*(X, \bar{x}) \text{ for every } s \text{ with } s(\bar{x}) \in X^* \quad (\text{Claim 13}) \\ &\iff (\mathfrak{A}, X) \models_s \psi^*(X, \bar{x}) \text{ for every } s \text{ with } (\mathfrak{A}, X) \models_s e_T(\bar{u}) \wedge X\bar{v} \quad (\text{due to (2)}) \\ &\iff (\mathfrak{A}, X) \models \forall \bar{u}\forall \bar{v}\forall \bar{w}((e_T(\bar{u}) \wedge X\bar{v}) \rightarrow \psi^*(X, \bar{u}, \bar{v}, \bar{w})) \equiv \mu. \end{aligned}$$

As a result, we have that $(\mathfrak{A}, X) \models \mu(X) \iff (\mathfrak{A}, X^*) \models \varphi_{\mathcal{T}}^{\mathcal{I}}$. Putting everything together yields:

$$(\mathfrak{A}, X) \models \mu \iff (\mathfrak{A}, X^*) \models \varphi_{\mathcal{T}}^{\mathcal{I}} \stackrel{(1)}{\iff} (\mathcal{G}, X) \models \varphi_{\mathcal{T}} \stackrel{(\text{Claim 12})}{\iff} X \in \mathcal{T}(\mathcal{G}) \stackrel{(\text{Theorem 6})}{\iff} (\mathfrak{A}, X) \models \varphi$$

Thus, the constructed myopic formula $\mu(X)$ is indeed equivalent to $\varphi(X)$. ◀

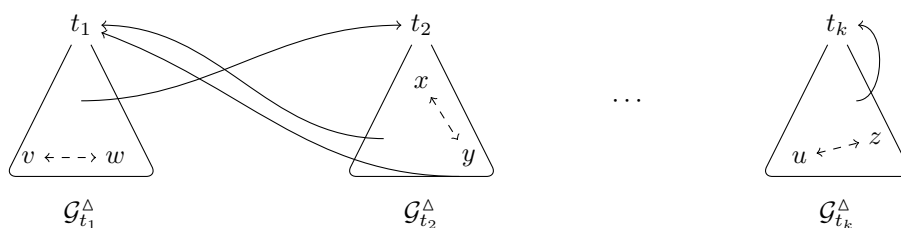
This construction can be applied to non union closed formulae as well, in which case the statement becomes $(\mathfrak{A}, \bigcup_{i \in I} X_i) \models \mu \iff (\mathfrak{A}, X_i) \models \varphi$ for all $i \in I$. To see this replace Claim 12 by “For every $X \subseteq A^r$, $(\mathcal{G}, X) \models \varphi_{\mathcal{T}}(X) \iff X = \bigcup_i X_i$ for some $X_i \in \mathcal{T}(\mathcal{G})$ ”.

5 Union Games

In the previous section we have characterised the union closed fragment of Σ_1^1 by means of a syntactic normal form. Now we aim at a game theoretic description, which leads to the following restriction of inclusion-exclusion games that reveals *how* union closed properties are assembled.

► **Definition 14.** A union game is an inclusion-exclusion game $\mathcal{G} = (V, V_0, V_1, E, I, T, E_{\text{ex}})$ obeying the following restrictions. For every $t \in T$ the subgraph reachable from t via the edges $E \setminus E_{\text{in}}$, that are the edges of E that do not go back into T , is denoted by \mathcal{G}_t^{Δ} .⁴ These

⁴ Recall that $E_{\text{in}} := E \cap (V \times T)$.



■ **Figure 1** A drawing of a union game. The target positions $T = \{t_1, \dots, t_k\}$ are at the top of the components \mathcal{G}_t^Δ , which are depicted by triangles. Recall that the inclusion edges, that are the edges going into target vertices, do not account for the reachability of the components \mathcal{G}_t^Δ . The exclusion edges E_{ex} are drawn as dashed arrows and, as seen here, are allowed only inside a component.

components must be disjoint, that is $V(\mathcal{G}_t^\Delta) \cap V(\mathcal{G}_{t'}^\Delta) = \emptyset$ for all $t \neq t' \in T$. Furthermore, exclusion edges are only allowed between vertices of the same component, that is $E_{\text{ex}} \subseteq \bigcup_{t \in T} V(\mathcal{G}_t^\Delta) \times V(\mathcal{G}_t^\Delta)$. The set of initial positions is empty, i.e. $I = \emptyset$.

See Figure 1 for a graphical representation of a union game. Since the exclusion edges are only inside a component we can in a way combine different strategies into one, which is the reason the target set of a union game is closed under unions.

► **Theorem 15.** *Let \mathcal{G} be a union game and $(\mathcal{S}_i)_{i \in J}$ be a family of winning strategies for player 0. Then there is a winning strategy \mathcal{S} for player 0 such that $\mathcal{T}(\mathcal{S}) = \bigcup_{i \in J} \mathcal{T}(\mathcal{S}_i)$. In other words, the set $\mathcal{T}(\mathcal{G})$ is closed under unions.*

Proof. Let $\mathcal{S}_i = (W_i, F_i)$ for $i \in J$. Let $U := \bigcup_{i \in J} \mathcal{T}(\mathcal{S}_i)$ and $f: U \rightarrow J$ be a function such that $t \in \mathcal{T}(\mathcal{S}_{f(t)})$ for all $t \in U$. Define $\mathcal{S} := \bigcup_{t \in U} (\mathcal{S}_{f(t)} \upharpoonright_{V(\mathcal{G}_t^\Delta)} + (E(\mathcal{S}_{f(t)}) \cap (V(\mathcal{G}_t^\Delta) \times T)))$. In words, \mathcal{S} is defined on every component \mathcal{G}_t^Δ with $t \in U$ as an arbitrary strategy \mathcal{S}_t that is defined on \mathcal{G}_t^Δ , including the inclusion edges leaving this component. By definition $\mathcal{T}(\mathcal{S}) = U$ and, furthermore, \mathcal{S} is indeed a winning strategy since it behaves on every component \mathcal{G}_t^Δ like $\mathcal{S}_{f(t)}$ and there are no exclusion edges between different components. ◀

► **Definition 16.** *Let $\mu(X) = \forall \bar{x}(X\bar{x} \rightarrow \exists \bar{R}\varphi(X, \bar{R}, \bar{x}))$ be a myopic τ -formula where φ is in negation-normal form and \mathfrak{A} be a τ -structure. The union game $\mathcal{G}(\mathfrak{A}, \mu) := (V, V_0, V_1, E, I = \emptyset, T = A^{\text{ar}(X)}, E_{\text{ex}})$ is defined similarly to Definition 5 with the difference being that for each $\bar{a} \in A^{\text{ar}(\bar{x})}$ we have to play on a copy of the game, so positions are now of the form (ϑ, s, \bar{a}) instead of (ϑ, s) , where $\vartheta \in \text{subf}(\varphi)$. The target vertices are the roots of these components, which is reflected by edges from \bar{a} to $(\varphi, \bar{x} \mapsto \bar{a}, \bar{a})$. Because of this construction exclusion edges can only occur inside a component.*

Notice that there are still edges from $(X\bar{x}, s, \bar{a})$ to $s(\bar{x})$ – the inclusion edges. It is also worth mentioning that the empty set is always included in $\mathcal{T}(\mathcal{G}(\mathfrak{A}, \mu))$ for all myopic μ because (\emptyset, \emptyset) is a (trivial) winning strategy for player 0. This mimics the behaviour that in case $X = \emptyset$, the formula $\forall \bar{x}(X\bar{x} \rightarrow \psi)$ is satisfied regardless of everything else. The analogue of Theorem 6 holds for union games and myopic formulae.

► **Proposition 17.** *Let \mathfrak{A} , μ and $\mathcal{G}(\mathfrak{A}, \mu)$ be as in Definition 16. Then $(\mathfrak{A}, X) \models \mu \iff X \in \mathcal{T}(\mathcal{G}(\mathfrak{A}, \mu))$.*

We want to end this section with the remark that for other fragments with certain closure properties natural restrictions of inclusion-exclusion games exist. Especially, forbidding exclusion edges at all leads to model-checking games for inclusion logic, while forbidding inclusion edges results in games suited for exclusion logic.

6 Myopic Fragment of Inclusion-Exclusion Logic

Similarly to the normal form of union closed Σ_1^1 -formulae (see Section 4) we present syntactic restrictions of inclusion-exclusion logic $\text{FO}(\subseteq, |)$ that correspond precisely to the union closed fragment \mathcal{U}^5 . Analogously to myopic Σ_1^1 -formulae we will also present a normal form for all union closed $\text{FO}(\subseteq, |)$ -formulae.

► **Definition 18.** *A formula $\varphi(\bar{x}) \in \text{FO}(\subseteq, |)$ is \bar{x} -myopic, if the following conditions are satisfied:*

- (a) *The variables from \bar{x} are never quantified in φ .*
- (b) *Every exclusion atom occurring in φ is of the form $\bar{x}\bar{y} | \bar{x}\bar{z}$.*
- (c) *Every inclusion atom occurring in φ is of the form $\bar{x}\bar{y} \subseteq \bar{x}\bar{z}$ or $\bar{y} \subseteq \bar{x}$, where the latter is only allowed if it is not in the scope of a disjunction.*

Please note that $\varphi(\bar{x})$ must not have any additional free variables besides \bar{x} . We call atoms of the form $\bar{x}\bar{y} \subseteq \bar{x}\bar{z}$ or $\bar{x}\bar{y} | \bar{x}\bar{z}$ (\bar{x} -)guarded and $\bar{y} \subseteq \bar{z}$, respectively $\bar{y} | \bar{z}$, the corresponding unguarded versions. Analogously, we call a formula ψ the unguarded version of φ , if ψ emerges from φ by replacing every dependency atom by the respective unguarded version.

The intuition behind this definition is that every \bar{x} -myopic formula can be evaluated componentwise on every team $X \upharpoonright_{\bar{x}=\bar{a}} = \{s \in X : s(\bar{x}) = \bar{a}\}$ for all $\bar{a} \in X(\bar{x})$. For a formula φ let T_φ denote its syntax tree⁶. A (team-)labelling of T_φ is a function λ mapping every node v_ψ to a team $\lambda(v_\psi)$ whose domain includes $\text{free}(\psi)$. In the following we write $\lambda(\psi)$ instead of $\lambda(v_\psi)$ if it is clear from the context which *occurrence* of the subformula ψ of φ is meant. We call λ a *witness* for $\mathfrak{A} \models_X \varphi$, if $\lambda(\varphi) = X$ and the semantical rules of Definition 1 are satisfied (e.g. $\lambda(\psi \vee \vartheta) = \lambda(\psi) \cup \lambda(\vartheta)$) and for every literal β of φ we have $\mathfrak{A} \models_{\lambda(\beta)} \beta$. By induction, if λ is a witness for $\mathfrak{A} \models_X \varphi$, then for every $\psi \in \text{subf}(\varphi)$ we have $\mathfrak{A} \models_{\lambda(\psi)} \psi$ and, moreover, $\mathfrak{A} \models_X \varphi$ if and only if there is a witness λ for $\mathfrak{A} \models_X \varphi$.

► **Proposition 19.** *Let X be team over \mathfrak{A} with $\text{dom}(X) \supseteq \{\bar{x}, \bar{v}, \bar{w}\}$ and $\varphi(\bar{x})$ be \bar{x} -myopic.*

1. $\mathfrak{A} \models_X \bar{x}\bar{v} \subseteq \bar{x}\bar{w} \iff \mathfrak{A} \models_{X \upharpoonright_{\bar{x}=\bar{a}}} \bar{v} \subseteq \bar{w}$ for all $\bar{a} \in X(\bar{x})$
2. $\mathfrak{A} \models_X \bar{x}\bar{v} | \bar{x}\bar{w} \iff \mathfrak{A} \models_{X \upharpoonright_{\bar{x}=\bar{a}}} \bar{v} | \bar{w}$ for all $\bar{a} \in X(\bar{x})$
3. *For every subformula $\bar{v} \subseteq \bar{x}$ of φ and witness λ for $\mathfrak{A} \models_X \varphi$ we have $(\lambda(\bar{v} \subseteq \bar{x}))(\bar{x}) = X(\bar{x})$.*

Like union games an \bar{x} -myopic formula is evaluated componentwise, which leads to the union closure of this fragment.

► **Theorem 20.** *Let $\varphi(\bar{x}) \in \text{FO}(\subseteq, |)$ be \bar{x} -myopic and $\mathfrak{A} \models_{X_i} \varphi$ for all $i \in I$. Then $\mathfrak{A} \models_X \varphi$ for $X = \bigcup_{i \in I} X_i$.*

It remains to prove that indeed every union closed formula φ of $\text{FO}(\subseteq, |)$ is equivalent to some \bar{x} -myopic formula. As we have already seen in Theorem 8, every union closed formula of existential second-order logic is equivalent to some myopic Σ_1^1 -formula. Moreover, it is well known that every $\text{FO}(\subseteq, |)$ -formula can be translated into an equivalent Σ_1^1 -formula [1]. Such a formula can be expressed as an \bar{x} -myopic one of the form $\exists \bar{s}(\bar{s} \subseteq \bar{x} \wedge \psi)$ where ψ uses only \bar{x} -guarded atoms.

⁵ We have defined \mathcal{U} to be the set of all union closed Σ_1^1 -formulae, by slight abuse of notation we use the same symbol here to denote the set of all $\text{FO}(\subseteq, |)$ -formulae that are closed under unions.

⁶ Since we consider a tree instead of a DAG, identical subformulae may occur at different nodes.

► **Lemma 21.** *Let $\varphi(\bar{x}) \in \text{FO}(\subseteq, |)$ be an \bar{x} -myopic formula of the form $\exists \bar{s}(\bar{s} \subseteq \bar{x} \wedge \psi)$, where in ψ no inclusion atoms of the form $\bar{y} \subseteq \bar{x}$ occur. Then $\mathfrak{A} \models_X \varphi$ if and only if there exists $F: X \rightarrow \mathcal{P}^+(A^{|\bar{x}|})$ such that $F(s) \subseteq X(\bar{x})$ for every $s \in X$ and $\mathfrak{A} \models_{X[\bar{s} \mapsto F]} \psi'$ for all $\bar{a} \in X(\bar{x})$, where ψ' is the unguarded version of ψ .*

Proof. By induction on ψ and applying Proposition 19. ◀

We present two different proofs for the next theorem, which bring a myopic Σ_1^1 -formula into this normal form. The following proof is based on methods of Galliani, Kontinen and Väänänen [1, 8] while the other one resembles the proof of Theorem 11 and can be found in the full version.

► **Theorem 22.** *Let $\varphi(X)$ be a myopic Σ_1^1 -formula. There is an equivalent \bar{x} -myopic formula of $\text{FO}(\subseteq, |)$ where $|\bar{x}| = \text{ar}(X)$.*

Proof. First of all let us introduce a normal form of myopic Σ_1^1 -formulae. Since in myopic formulae the variable X may occur only positively in the subformula φ' , we can transform every $\forall \bar{x}(X\bar{x} \rightarrow \exists \bar{R}\varphi'(\bar{R}, X, \bar{x}))$ into the equivalent formula $\forall \bar{x}(X\bar{x} \rightarrow \exists S(S \subseteq X \wedge \exists \bar{R}\varphi'(\bar{R}, S, \bar{x})))$, where $S \subseteq X$ is a shorthand for $\forall \bar{y}(S\bar{y} \rightarrow X\bar{y})$. We now apply the Skolem-normal form of Σ_1^1 -formulae to $\exists \bar{R}\varphi'(\bar{R}, S, \bar{x})$, which yields the formula $\sigma(S, \bar{x}) := \exists \bar{f}\forall \bar{y}((f_1(\bar{w}) = f_2(\bar{w}) \leftrightarrow S\bar{w}) \wedge \psi(\bar{f}, \bar{x}, \bar{y}))$, where ψ is a quantifier-free first-order formula and \bar{w} is a subtuple of \bar{y} and, moreover, every f_i occurs in σ only with a unique tuple \bar{w}_i (consisting of pairwise different variables) as argument, that is $f_i(\bar{w}_i)$ (see [8] where an analogous construction is made). The original formula can thus be transformed into $\forall \bar{x}(X\bar{x} \rightarrow \exists S(S \subseteq X \wedge \sigma(S, \bar{x}))$. Similarly to [1] we embed $\sigma(S, \bar{x})$ into inclusion-exclusion logic as $\vartheta(\bar{s}, \bar{x}) := \forall \bar{y}\exists \bar{z}(\bigwedge_i (\bar{x}\bar{w}_i, z_i) \wedge ((\bar{x}\bar{w} \subseteq \bar{x}\bar{s} \wedge z_1 = z_2) \vee (\bar{x}\bar{w} | \bar{x}\bar{s} \wedge z_1 \neq z_2))) \wedge \psi'(\bar{x}, \bar{y}, \bar{z}))$. Here ψ' is obtained from ψ by simply replacing every occurrence of $f_i(\bar{w}_i) = f_j(\bar{w}_j)$ by $z_i = z_j$. The only difference in our case is that every dependency atom is \bar{x} -guarded due to the fact that the subformula at hand is inside the scope of the universally quantified variables \bar{x} in $\forall \bar{x}(X\bar{x} \rightarrow \dots)$. Notice that dependence atoms of the form $(\bar{x}\bar{w}_i, z_i)$ can also be regarded as \bar{x} -myopic. Formally, we can embed such an atom into exclusion logic via the formula $\forall v(\bar{x}\bar{w}_i v | \bar{x}\bar{w}_i z_i \vee z_i = v)$, which has the intended shape [1]. The whole formula $\varphi(X)$ thus translates into $\mu(\bar{x}) := \exists \bar{s}(\bar{s} \subseteq \bar{x} \wedge \vartheta(\bar{s}, \bar{x}))$. Let $\vartheta'(\bar{s}, \bar{x})$ be the unguarded version of $\vartheta(\bar{s}, \bar{x})$. Analogously to the argumentation of Galliani [1] by additionally making use of Proposition 19, we see that $(\mathfrak{A}, Y |_{\bar{x}=\bar{a}}(\bar{s})) \models_{\bar{x} \mapsto \bar{a}} \sigma(S, \bar{x})$ if and only if $\mathfrak{A} \models_{Y |_{\bar{x}=\bar{a}}} \vartheta'(\bar{s})$ for $\bar{a} \in Y(\bar{x})$, where Y is a team with domain $\{\bar{s}, \bar{x}\}$ (here the variable S takes the role of the team). Using Lemma 21 we have $\mathfrak{A} \models_X \mu(\bar{x})$ if and only if there is a function $F: X \rightarrow \mathcal{P}^+(A^{\text{ar}(\bar{s})})$ such that $F(s) \subseteq X(\bar{x})$ for every $s \in X$ and $\mathfrak{A} \models_{X[\bar{s} \mapsto F]} \vartheta'(\bar{s}, \bar{x})$ for all $\bar{a} \in X(\bar{x})$, which again holds if and only if there exists such an F and $(\mathfrak{A}, F(s)) \models_t \sigma(S, \bar{x})$ for all $t \in X$, but this just means $(\mathfrak{A}, X(\bar{x})) \models \forall \bar{x}(X\bar{x} \rightarrow \exists S(S \subseteq X \wedge \sigma(S, \bar{x})))$. ◀

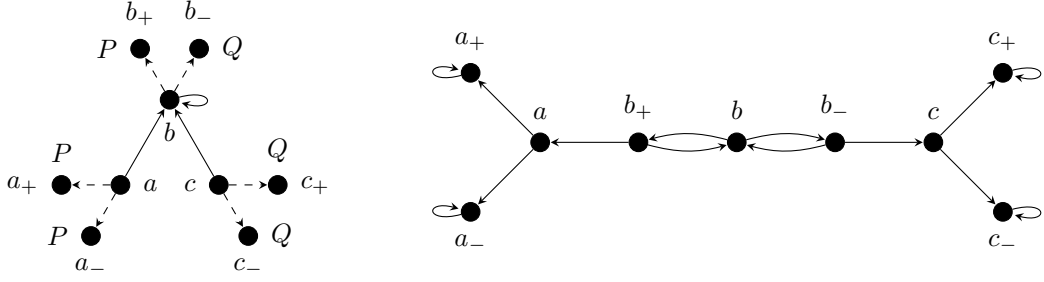
► **Corollary 23** (Normal form of myopic-FO($\subseteq, |$)). *Let $\varphi(\bar{x})$ be a union closed formula of $\text{FO}(\subseteq, |)$. There is a logically equivalent \bar{x} -myopic formula $\psi(\bar{x}) = \exists \bar{s}(\bar{s} \subseteq \bar{x} \wedge \vartheta)$ where in ϑ only \bar{x} -guarded dependency atoms occur.*

6.1 Optimality of the Myopic Fragment of Inclusion-Exclusion Logic

One might ask whether the restrictions of Definition 18 are actually imperative to capture the union closed fragment. In this section, we will show that neither condition can be dropped and that every single atom of Definition 18 is required to express all union closed properties.

We start by showing that neither condition can be dropped. First of all, it is pretty clear that exclusion atoms have to be \bar{x} -guarded, because $x_1 | x_2$ is not guarded and obviously

25:12 On the Union Closed Fragment



■ **Figure 2** The structures \mathfrak{A} and \mathfrak{B} . The structure $\mathfrak{A} = (V, E^{\mathfrak{A}}, F^{\mathfrak{A}}, P^{\mathfrak{A}}, Q^{\mathfrak{A}})$ on the left side uses two different kinds of edges: the dashed edges belong to F , while the other are E -edges. Furthermore, \mathfrak{A} exhibits two predicates P, Q . The structure $\mathfrak{B} = (V, E^{\mathfrak{B}})$ depicted on the right is just a directed graph. Please notice that both structures are using the same universe V .

not closed under unions. Furthermore, it is clear that the variables among \bar{x} must not be quantified. This points out the necessity of conditions (a) and (b) of Definition 18. In the next example we demonstrate that neither restriction of condition (c) can be dropped.

► **Example 24.** Consider the structures \mathfrak{A} and \mathfrak{B} drawn in Figure 2 and the following formulae:

$$\begin{aligned} \varphi(x) &:= \exists y \exists z (Fxy \wedge Fxz \wedge xy \mid xz \wedge [(Py \wedge \vartheta(x)) \vee (Qy \wedge \vartheta(x))]) \\ &\text{where } \vartheta(x) := \exists v (Exv \wedge v \subseteq x) \\ \psi(x) &:= \exists y \exists z (Exy \wedge Exz \wedge xy \mid xz \wedge \exists w (Eyw \wedge x \subseteq w)) \end{aligned}$$

Neither $\varphi(x)$ nor $\psi(x)$ is x -myopic, because the inclusion atom $v \subseteq x$ from ϑ occurs inside the scope of a disjunction (and it is not x -guarded), while the atom $x \subseteq w$ is neither x -guarded nor of the form that is allowed outside the scope of disjunctions, because x appears on the wrong side of the inclusion atom.

For every $v \in V$ let $s_v: \{x\} \rightarrow V$ be the assignment with $s_v(x) := v$. We define the teams $X_1 := \{s_a, s_b\}$, $X_2 := \{s_b, s_c\}$ and $X := X_1 \cup X_2 = \{s_a, s_b, s_c\}$. It is not difficult to verify that $\mathfrak{A} \models_{X_i} \varphi(x)$ and $\mathfrak{B} \models_{X_i} \psi(x)$ for $i = 1, 2$ but $\mathfrak{A} \not\models_X \varphi(x)$ and $\mathfrak{B} \not\models_X \psi(x)$. In particular, neither $\varphi(x)$ nor $\psi(x)$ is closed under unions. This shows that the restrictions of Definition 18 are indeed necessary.

Thus the atoms allowed in Definition 18 are sufficient to capture the union closed fragment of $\text{FO}(\subseteq, \mid)$. On the contrary, one may ask whether the set of atoms given in Definition 18 is necessary. Let us argue for all rules of Definition 18.

Assume that all exclusion atoms are forbidden. Then every formula is already in inclusion logic in which one cannot define every union closed property as was shown by Galliani and Hella [2, p. 16].

If inclusion atoms were only allowed in the form $\bar{x}\bar{y} \subseteq \bar{x}\bar{z}$, that means the atoms $\bar{y} \subseteq \bar{x}$ are forbidden, the formulae become flat, as can be seen by considering Proposition 19, but not all union closed properties are flat.

The case where inclusion atoms of form $\bar{x}\bar{y} \subseteq \bar{x}\bar{z}$ are forbidden is a bit more delicate. To prove that such a formula cannot express every union closed property consider the formula $\mu(x) = \exists z (z \subseteq x \wedge \forall y (Exy \rightarrow xy \subseteq xz))$, where $\tau = \{E\}$ for a binary predicate symbol E . This formula axiomatises the set of all teams X over a graph $G = (V, E)$ such that whenever $v \in X(x)$ and $(v, w) \in E$, then already $w \in X(x)$. The formula obviously describes a union

closed property. Consider the graph $G: b \leftarrow a \rightarrow c$. Here, $G \models_X \mu(x)$ for precisely those teams X that satisfy “ $a \in X(x)$ implies $b, c \in X(x)$ ”. For every $v \in V(G)$ let s_v be the assignment $x \mapsto v$ and let $X_v := \{s_v\}$. Furthermore, we define $X_{abc} := \{s_a, s_b, s_c\}$.

Let $\psi(x)$ be an x -myopic formula in which the construct $x\bar{y} \subseteq x\bar{z}$ does not appear. So the only inclusion atoms occurring in $\psi(x)$ are of the form $z \subseteq x$, which are not allowed in the scope of disjunctions. Notice that z cannot be universally quantified, as the team $X_b = \{s_b\}$ satisfies the described property, but not $\forall z(z \subseteq x)$. Thus we may assume without loss of generality that $\psi(x)$ has the form $\exists z(z \subseteq x \wedge \psi'(x, z))$, where in $\psi'(x, z)$ no atom of the kind $z' \subseteq x$ occurs. We want to remark that the following argumentation can be adapted to the slightly more general case that multiple atoms of form $z \subseteq x$ occur, but for sake of simplicity we only deal with one such atom. Let $\eta(x, z)$ be the unguarded version of $\psi'(x, z)$. By Lemma 21, there is a function $F: X_{abc} \rightarrow \mathcal{P}^+(V(G))$ such that $F(s) \subseteq X_{abc}(x) = V(G)$ for $s \in X_{abc}$ and $G \models_{X_{abc}[z \mapsto F]} \eta$ for every $v \in X_{abc}(x)$. Please notice that $X_{abc}[z \mapsto F] \upharpoonright_{x=v} = X_v[z \mapsto F]$. Moreover, because in $\eta(x, z)$ no inclusion atom occurs it is downwards closed. Assume $a \in F(s_a)$. By downwards closure of $\eta(x, z)$ we obtain $G \models_{X_a[z \mapsto a]} \eta$, which, by Lemma 21, implies that $G \models_{X_a} \psi$ contradicting our assumption that ψ describes the desired property. Otherwise, because of symmetry, b is in $F(s_a)$, and hence $G \models_{X_a[z \mapsto b]} \eta$. Additionally, since $G \models_{X_b} \psi$ we know, by Lemma 21, that $G \models_{X_b[z \mapsto b]} \eta$. Together this implies $G \models_{X_{ab}[z \mapsto b] \upharpoonright_{x=v}} \eta$ for $v = a, b$ and, due to Lemma 21, we get $G \models_{X_{ab}} \psi$ which is again in conflict with our assumption about ψ describing the desired property.

7 An Atom capturing the Union Closed Fragment

The present work was motivated by a question of Galliani and Hella in 2013 [2]. Galliani and Hella asked whether there is a union closed atomic dependency notion α that is definable in existential second-order logic such that $\text{FO}(\alpha)$ corresponds precisely to all union closed properties of $\text{FO}(\subseteq, |)$. In [2] they have already shown that inclusion logic does not suffice, as there are union closed properties not definable in it. Moreover, they have established a theorem stating that every union closed atomic property that is definable in first-order logic (where the formula has access to the team via a predicate) is expressible in inclusion logic. Thus, whatever atom characterises all union closed properties of $\text{FO}(\subseteq, |)$ must axiomatise an inherently second-order property.

Intuitively speaking, as we have seen in Section 5, solving union games is a complete problem for the class \mathcal{U} . Therefore, a canonical solution to this question is to propose an atomic formula that defines the winning regions in a union game. Towards this we must describe how a game can be encoded into a team. This is not as straightforward as one might think, because there is a technical pitfall we need to avoid. The union of two teams describing union games, each won by player 0, might encode a game won by player 1, but by union closure it must satisfy the atomic formula.

We encode union games in teams by using variable tuples for the respective components, where we also encode the complementary relations in order to ensure that the union of two different games cannot form a different game. For $k \in \mathbb{N}$ let \mathcal{V}_k be the set of distinct k -tuples of variables $\{\bar{u}, \bar{v}_0, \bar{v}_1, \bar{v}, \bar{w}, \bar{t}, \bar{v}_{\text{ex}}, \bar{w}_{\text{ex}}, \bar{\varepsilon}_1, \bar{\varepsilon}_2, \bar{u}^{\mathbb{C}}, \bar{v}^{\mathbb{C}}, \bar{w}^{\mathbb{C}}, \bar{t}^{\mathbb{C}}, \bar{v}_{\text{ex}}^{\mathbb{C}}, \bar{w}_{\text{ex}}^{\mathbb{C}}, \bar{\varepsilon}_1^{\mathbb{C}}, \bar{\varepsilon}_2^{\mathbb{C}}\}$.

► **Definition 25.** *Let X be a team with $\mathcal{V}_k \subseteq \text{dom}(X)$ and codomain A . We define $\sim := X(\bar{\varepsilon}_1, \bar{\varepsilon}_2)$ and $\mathfrak{A}^X := (V, V_0, V_1, E, I, T, E_{\text{ex}})$ with the following components.*

25:14 On the Union Closed Fragment

- $V := X(\bar{u})$ ■ $V_1 := X(\bar{v}_1)$ ■ $I := \emptyset$ ■ $E_{\text{ex}} := X(\bar{v}_{\text{ex}}, \bar{w}_{\text{ex}})$
- $V_0 := X(\bar{v}_0)$ ■ $E := X(\bar{v}, \bar{w})$ ■ $T := X(\bar{t})$

If the following consistency requirements are satisfied, then we define $\mathcal{G}_X^A := \mathfrak{A}_{/\sim}^X$.

1. $X(\bar{u}^{\mathbb{G}}) = A^k \setminus V$
2. $X(\bar{v}^{\mathbb{G}}, \bar{w}^{\mathbb{G}}) = (A^k \times A^k) \setminus E$
3. $X(\bar{t}^{\mathbb{G}}) = A^k \setminus T$
4. $X(\bar{v}_{\text{ex}}^{\mathbb{G}}, \bar{w}_{\text{ex}}^{\mathbb{G}}) = (A^k \times A^k) \setminus E_{\text{ex}}$
5. $X(\bar{\varepsilon}_1^{\mathbb{G}}, \bar{\varepsilon}_2^{\mathbb{G}}) = (A^k \times A^k) \setminus \sim$
6. $V_0 = V \setminus V_1$
7. \mathfrak{A}^X is a structure⁷.
8. \sim is a congruence on \mathfrak{A}^X .
9. $\mathfrak{A}_{/\sim}^X$ is a union game.

Otherwise, if any of these requirements is not fulfilled, we let \mathcal{G}_X^A be undefined.

We call X complete (w.r.t. A), if $X(\bar{y}) \cup X(\bar{y}^{\mathbb{G}})$ is A^k or $A^k \times A^k$ for every $\bar{y} \in \{(\bar{u}), (\bar{v}, \bar{w}), (\bar{t}), (\bar{v}_{\text{ex}}, \bar{w}_{\text{ex}}), (\bar{\varepsilon}_1, \bar{\varepsilon}_2)\}$ and $V = V_0 \cup V_1$, and incomplete otherwise. It is easy to observe that \mathcal{G}_X^A is undefined for every incomplete team X . Furthermore complete subteams of teams describing a game actually describe the same game and the same congruence relation.

► **Lemma 26.** *Let X, Y be teams with codomain A and $\mathcal{V}_k \subseteq \text{dom}(X) = \text{dom}(Y)$. If X is complete, $X \subseteq Y$ and \mathcal{G}_Y^A is defined, then $\mathcal{G}_X^A = \mathcal{G}_Y^A$ and $\sim_X := X(\bar{\varepsilon}_1, \bar{\varepsilon}_2) = Y(\bar{\varepsilon}_1, \bar{\varepsilon}_2) =: \sim_Y$.*

Now let us show that union games are definable in plain first-order logic with team semantics in the sense of Definition 25.

► **Lemma 27.** *Let $\varphi(X) = \forall \bar{x}(X\bar{x} \rightarrow \exists \bar{R}\varphi'(X, \bar{R}, \bar{x}))$ be a myopic Σ_1^1 -formula and $\psi(\mathcal{V}_k, \bar{x})$ be a formula with team semantics (where k is large enough such that the game $\mathcal{G}(\mathfrak{A}, \varphi)$ can be encoded). There is a formula $\vartheta_\varphi^\psi(\bar{x})$ such that $\mathfrak{A} \models_X \vartheta_\varphi^\psi \iff \mathfrak{A} \models_Y \psi$ for some team Y extending X with $\mathcal{G}_Y^A \cong \mathcal{G}(\mathfrak{A}, \varphi)$ and $X(\bar{x}) = Y(\bar{x})$, for every τ -structure \mathfrak{A} .*

Proof. Similar to Lemma 10, it is easy to construct a (quantifier-free) first-order interpretation $\mathcal{I} := (\delta, \varepsilon, \psi_V, \psi_{V_0}, \psi_{V_1}, \psi_E, \psi_I, \psi_T, \psi_{E_{\text{ex}}})$ with $\mathcal{I}(\mathfrak{A}) \cong \mathcal{G}(\mathfrak{A}, \varphi)$. Now let $\vartheta_\varphi^\psi(\bar{x}) := \forall \mathcal{V}_k(\gamma(\mathcal{V}_k) \rightarrow \psi(\mathcal{V}_k, \bar{x}))$ where the formula

$$\begin{aligned} \gamma(\mathcal{V}_k) := & \delta(\bar{u}) \wedge \psi_{V_0}(\bar{v}_0) \wedge \psi_{V_1}(\bar{v}_1) \wedge \psi_E(\bar{v}, \bar{w}) \wedge \psi_T(\bar{t}) \wedge \psi_{E_{\text{ex}}}(\bar{v}_{\text{ex}}, \bar{w}_{\text{ex}}) \wedge \varepsilon(\bar{\varepsilon}_1, \bar{\varepsilon}_2) \wedge \\ & \neg\delta(\bar{u}^{\mathbb{G}}) \wedge \neg\psi_E(\bar{v}^{\mathbb{G}}, \bar{w}^{\mathbb{G}}) \wedge \neg\psi_T(\bar{t}^{\mathbb{G}}) \wedge \neg\psi_{E_{\text{ex}}}(\bar{v}_{\text{ex}}^{\mathbb{G}}, \bar{w}_{\text{ex}}^{\mathbb{G}}) \wedge \neg\varepsilon(\bar{\varepsilon}_1^{\mathbb{G}}, \bar{\varepsilon}_2^{\mathbb{G}}) \end{aligned}$$

enforces that the game $\mathcal{G}(\mathfrak{A}, \varphi)$ will be “loaded” into the team. As long as none of these conjuncts are unsatisfiable this construction is correct. This is safe to assume because one can easily transform a union game into an equivalent one w.r.t. the target set such that none of its components are empty. ◀

This knowledge enables us to finally define the atomic formula we sought after. For this we need to show that the atom is union closed and its first-order closure can express all of \mathcal{U} .

► **Definition 28.** *The atomic team formula $\cup\text{-game}(\mathcal{V}_k, \bar{x})$ for the respective tuples of variables has the following semantics. For non-empty teams X with $\mathcal{V}_k, \bar{x} \subseteq \text{dom}(X)$ we define*

$$\mathfrak{A} \models_X \cup\text{-game}(\mathcal{V}_k, \bar{x}) \iff X \text{ is complete and if } \mathcal{G}_X^A \text{ is defined, then } X(\bar{x})_{/X(\bar{\varepsilon}_1, \bar{\varepsilon}_2)} \in \mathcal{T}(\mathcal{G}_X^A)$$

and we set $\mathfrak{A} \models_\emptyset \cup\text{-game}(\mathcal{V}_k, \bar{x})$ to be always true (to ensure the empty team property).

⁷ This condition ensures that $V_0 \subseteq V$ and so forth.

Note that this atom can be defined in existential second-order logic.

► **Proposition 29.** *The atomic formula $\cup\text{-game}$ is union closed.*

Proof. Assume that $\mathfrak{A} \models_{X_i} \cup\text{-game}(\mathcal{V}_k, \bar{x})$ for $i \in I$. We prove that $\mathfrak{A} \models_X \cup\text{-game}(\mathcal{V}_k, \bar{x})$ holds for the union $X := \bigcup_{i \in I} X_i$. If $X = \emptyset$, there is nothing to prove. Otherwise at least one X_j is non-empty and, since $\mathfrak{A} \models_{X_j} \cup\text{-game}(\mathcal{V}_k, \bar{x})$, X_j must be complete implying that X is also complete (because $X \supseteq X_j$). For the remainder of this proof, we assume w.l.o.g. that all involved teams X_i (and X) are non-empty. If \mathcal{G}_X^A is undefined, then $\mathfrak{A} \models_X \cup\text{-game}(\mathcal{V}_k, \bar{x})$ follows from the definition of $\cup\text{-game}$. Otherwise, if \mathcal{G}_X^A is defined, then we can use Lemma 26 to obtain that $\mathcal{G}_X^A = \mathcal{G}_{X_i}^A$ and $\sim := X(\bar{\varepsilon}_1, \bar{\varepsilon}_2) = X_i(\bar{\varepsilon}_1, \bar{\varepsilon}_2)$ for every $i \in I$. Since $\mathfrak{A} \models_{X_i} \cup\text{-game}(\mathcal{V}_k, \bar{x})$, we can conclude that $X_i(\bar{x})_{/\sim} \in \mathcal{T}(\mathcal{G}_{X_i}^A) = \mathcal{T}(\mathcal{G}_X^A)$ for each $i \in I$. By Theorem 15, $X(\bar{x})_{/\sim} = \bigcup_{i \in I} X_i(\bar{x})_{/\sim} \in \mathcal{T}(\mathcal{G}_X^A)$ and, hence, $\mathfrak{A} \models_X \cup\text{-game}(\mathcal{V}_k, \bar{x})$. ◀

► **Theorem 30.** *Let $\varphi \in \text{FO}(\subseteq, |)$ be a union closed formula. There is a logically equivalent formula $\zeta \in \text{FO}(\cup\text{-game})$. In other words, $\text{FO}(\cup\text{-game})$ corresponds precisely to the union closed fragment of $\text{FO}(\subseteq, |)$.*

Proof. Let \mathfrak{A} be an arbitrary structure. Due to [1, Theorem 6.1] there exists a formula $\varphi'(X) \in \Sigma_1^1$ which is logically equivalent to $\varphi(\bar{x})$ in the sense that $\mathfrak{A} \models_X \varphi(\bar{x}) \iff (\mathfrak{A}, X(\bar{x})) \models \varphi'(X)$ for every team X with $\bar{x} \subseteq \text{dom}(X)$. By Theorem 8, there is a myopic formula $\mu \equiv \varphi'$. So, we have $(\mathfrak{A}, X(\bar{x})) \models \mu(X) \iff \mathfrak{A} \models_X \varphi(\bar{x})$.

The game $\mathcal{G}(\mathfrak{A}, \mu)$ from Definition 16 is a union game and Lemma 27 allows us to load this game into a team. Please notice, that Lemma 27 is using a similar first-order interpretation \mathcal{I} as Lemma 10, which encodes a target vertex $\bar{a} \in T(\mathcal{G}(\mathfrak{A}, \mu))$ by tuples of the form $(\bar{u}, \bar{a}, \bar{w})$ of length $k = n + m$ where the n -tuple \bar{u} has the equality type e_T while \bar{w} is an arbitrary tuple of length $m - |\bar{a}|$. Let $\psi(\mathcal{V}_k, \bar{x}) := \forall \bar{u} \forall \bar{w} (e_T(\bar{u}) \rightarrow \cup\text{-game}(\mathcal{V}_k, \bar{u}\bar{x}\bar{w}))$ and $\zeta(\bar{x}) := \vartheta_\mu^\psi$ be as in Lemma 27, that is $\forall \mathcal{V}_k (\gamma(\mathcal{V}_k) \rightarrow \psi(\mathcal{V}_k, \bar{x}))$. So $\mathfrak{A} \models_X \zeta(\bar{x}) \iff \mathfrak{A} \models_Y \psi(\mathcal{V}_k, \bar{x})$ where $Y = X[\mathcal{V}_k \mapsto A] \upharpoonright_\gamma$. As in Lemma 27, we have $\mathcal{G}_Y^A \cong \mathcal{I}(\mathfrak{A}) \cong \mathcal{G}(\mathfrak{A}, \mu)$ and $X(\bar{x}) = Y(\bar{x})$. Furthermore, we have defined $\mathcal{G}_Y^A = \mathfrak{A}_{/\sim}^Y$ where $\sim := Y(\bar{\varepsilon}_1, \bar{\varepsilon}_2)$.

Because of the construction of ψ , we have $\mathfrak{A} \models_Y \psi(\mathcal{V}_k, \bar{x}) \iff \mathfrak{A} \models_Z \cup\text{-game}(\mathcal{V}_k, \bar{u}\bar{x}\bar{w})$ where $Z := Y[\bar{u} \mapsto e_T^{\mathfrak{A}}, \bar{w} \mapsto A^{m-|\bar{x}|}]$. Since $\mathcal{G}_Z^A = \mathcal{G}_Y^A \cong \mathcal{G}(\mathfrak{A}, \mu)$ is a well-defined union game, this is equivalent to $Z(\bar{u}\bar{x}\bar{w})_{/\sim} \in \mathcal{T}(\mathcal{G}_Z^A)$. Let $h: \delta_T^{\mathfrak{A}} \rightarrow V(\mathcal{G}(\mathfrak{A}, \mu))$ be the coordinate map for $\mathcal{G}(\mathfrak{A}, \mu) \cong \mathcal{I}(\mathfrak{A})$. By construction, h induces an isomorphism between $\mathfrak{A}_{/\sim}^Y$ and $\mathcal{G}(\mathfrak{A}, \mu)$. In particular each element of any equivalence class $[(\bar{u}', \bar{a}, \bar{w}')]_{/\sim} \in Z(\bar{u}\bar{x}\bar{w})_{/\sim}$ is mapped by h to \bar{a} . Therefore, $Z(\bar{u}\bar{x}\bar{w})_{/\sim} \in \mathcal{T}(\mathcal{G}_Z^A) \iff Z(\bar{x}) = X(\bar{x}) \in \mathcal{T}(\mathcal{G}(\mathfrak{A}, \mu))$. Thus we have $\mathfrak{A} \models_X \zeta(\bar{x}) \iff X(\bar{x}) \in \mathcal{T}(\mathcal{G}(\mathfrak{A}, \mu))$. Putting everything together, we have $\mathfrak{A} \models_X \zeta(\bar{x}) \iff X(\bar{x}) \in \mathcal{T}(\mathcal{G}(\mathfrak{A}, \mu)) \iff (\mathfrak{A}, X(\bar{x})) \models \mu \iff \mathfrak{A} \models_X \varphi(\bar{x})$ as desired. ◀

8 Concluding Remarks

Let us remark on the “naturalness” of the atom $\cup\text{-game}$. Certainly inclusion, exclusion and the notions alike can be regarded as natural atomic dependency formulae, whereas the just introduced atom has to be classified differently. Nevertheless, it is a canonical candidate since it solves a complete problem of the desired class. Of course, a more natural – and more usable – atom might be found, but it will not be as simplistic as e.g. inclusion for Galliani and Hella have shown that every first-order definable union closed property is already expressible in inclusion logic. Hence, whatever atom one proposes, it must make use of some inherently second-order concepts. For concretely expressing properties, the introduced myopic fragments of Σ_1^1 and $\text{FO}(\subseteq, |)$ are more practical.

The various syntactical characterisations of the union closed fragments presented in this work now enables their further investigation. This could result in a complexity theoretical analysis or a more detailed classification of Σ_1^1 .

References

- 1 Pietro Galliani. Inclusion and Exclusion Dependencies in Team Semantics—on Some Logics of Imperfect Information. *Annals of Pure and Applied Logic*, 163(1):68–84, 2012.
- 2 Pietro Galliani and Lauri Hella. Inclusion Logic and Fixed Point Logic. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013)*, volume 23 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 281–295, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2013.281.
- 3 Erich Grädel. Games for Inclusion Logic and Fixed-Point Logic. In Samson Abramsky and Others, editors, *Dependence Logic: Theory and Applications*, pages 73–98. Birkhäuser, 2016.
- 4 Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein. *Finite Model Theory and Its Applications*. Springer-Verlag, Berlin Heidelberg, 2007.
- 5 Erich Grädel and Jouko Väänänen. Dependence and Independence. *Studia Logica*, 101(2):399–410, 2013. doi:10.1007/s11225-013-9479-2.
- 6 Wilfrid Hodges. *A Shorter Model Theory*. Cambridge University Press, New York, NY, USA, 1997.
- 7 Wilfrid Hodges. Compositional Semantics for a Language of Imperfect Information. *Logic Journal of the IGPL*, 5(4):539–563, 1997. doi:10.1093/jigpal/5.4.539.
- 8 Juha Kontinen and Jouko Väänänen. On Definability in Dependence Logic. *Journal of Logic, Language and Information*, 18(3):317–332, 2009.
- 9 Jouko Väänänen. *Dependence Logic: A New Approach to Independence Friendly Logic (London Mathematical Society Student Texts)*. Cambridge University Press, New York, NY, USA, 2007.
- 10 Johan van Benthem. *Modal Correspondence Theory*. PhD dissertation, University of Amsterdam, 1976.

Expressive Logics for Coinductive Predicates

Clemens Kupke

Strathclyde University, United Kingdom
clemens.kupke@strath.ac.uk

Jurriaan Rot

University College London, United Kingdom and Radboud University, The Netherlands
jrot@cs.ru.nl

Abstract

The classical Hennessy-Milner theorem says that two states of an image-finite transition system are bisimilar if and only if they satisfy the same formulas in a certain modal logic. In this paper we study this type of result in a general context, moving from transition systems to coalgebras and from bisimilarity to coinductive predicates. We formulate when a logic fully characterises a coinductive predicate on coalgebras, by providing suitable notions of adequacy and expressivity, and give sufficient conditions on the semantics. The approach is illustrated with logics characterising similarity, divergence and a behavioural metric on automata.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Categorical semantics

Keywords and phrases Coalgebra, Fibration, Modal Logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.26

Funding This work was partially supported by a Marie Curie Fellowship (grant code 795119).

Acknowledgements We would like to thank Bart Jacobs for useful discussions and the anonymous referees for their very constructive and helpful feedback.

1 Introduction

A prominent example of the deep connection between bisimilarity and modal logic is the *Hennessy-Milner theorem*: two states of an image-finite labelled transition system (LTS) are behaviourally equivalent iff they satisfy the same formulas in a certain modal logic [13]. From left to right, this equivalence is sometimes referred to as *adequacy* of the logic w.r.t. bisimilarity, and from right to left as *expressivity*. By proving both adequacy and expressivity, the Hennessy-Milner theorem thus gives a logical characterisation of behavioural equivalence.

There are numerous variants and generalisations of this kind of result. For instance, a state x of an LTS *simulates* a state y if every formula satisfied by x is also satisfied by y , where the logic only has conjunction and diamond modalities; see [36] for this and many other related results. Another class of examples is logical characterisations of quantitative notions of equivalence, such as probabilistic bisimilarity and behavioural distances (e.g., [27, 8, 35, 19, 24, 37, 7]). In many such cases, including bisimilarity, the comparison between states is *coinductive*, and the problem is thus to characterise a coinductively defined relation (or distance) with a suitable modal logic.

Both coinduction and modal logic can be naturally and generally studied within the theory of *coalgebra*, which provides an abstract, uniform study of state-based systems [32, 18]. Indeed, in the area of *coalgebraic modal logic* [26] there is a rich literature on deriving expressive logics for behavioural equivalence between state-based systems, thus going well beyond labelled transition systems [29, 33, 22]. However, such results focus almost exclusively on behavioural equivalence or bisimilarity – a coalgebraic theory of logics for characterising coinductive predicates other than bisimilarity is still missing. The aim of this paper is to



© Clemens Kupke and Jurriaan Rot;
licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 26; pp. 26:1–26:18



Leibniz International Proceedings in Informatics

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

accommodate the study of logical characterisation of coinductive predicates in a general manner, and provide tools to prove adequacy and expressivity.

Our approach is based on universal coalgebra, to achieve results that apply generally to state-based systems. Central to the approach are the following two ingredients.

1. *Coinductive predicates in a fibration.* To characterise coinductive predicates, we make use of fibrations – this approach originates from the seminal work of Hermida and Jacobs [14]. The fibration is used to speak about predicates and relations on states. In this context, liftings of the type functor of coalgebras uniformly determine coinductive predicates and relations on such coalgebras. An important feature of this approach, advocated in [12], is that it covers not only bisimilarity, but also other coinductive predicates including, e.g., similarity of labelled transition systems and other coalgebras [16], behavioural metrics [2, 4, 34], unary predicates such as divergence [5, 12], and many more.
2. *Coalgebraic modal logic via dual adjunctions.* We use an abstract formulation of coalgebraic logic, which originated in [30, 22], building on a tradition of logics via duality (e.g., [25, 6]). This framework is formulated in terms of a contravariant adjunction, which captures the basic connection between states and theories, and a distributive law, which captures the one-step semantics of the logic. It covers classical modal logics of course, but also easily accommodates multi-valued logics, and, e.g., logics without propositional connectives, where formulas can be thought of as basic tests on state-based systems. This makes the framework suitable for an abstract formulation of Hennessy-Milner type theorems, where formulas play the role of tests on state-based systems.

To formulate adequacy and expressivity with respect to general coinductive predicates, we need to know how to compare collections of formulas. For instance, if the coinductive predicate is similarity of LTSs, the associated logical theories of one state should be *included* in the other, not necessarily equal. This amounts to stipulating a *relation* on truth values, that extends to a relation between theories. In the quantitative case, we need a *logical distance* between collections of formulas; this typically arises from a distance between truth values (which, in this case, will typically be an interval in the real numbers). The fibrational setting provides a convenient means for defining such an object for comparing theories.

With this in hand, we arrive at the main contributions of this paper: the formulation of adequacy and expressivity of a coalgebraic modal logic with respect to a coinductive predicate in a fibration, and sufficient conditions on the semantics of the logic that guarantee adequacy and expressivity. We exemplify the approach through a range of examples, including logical characterisations of a simple behavioural distance on deterministic automata, similarity of labelled transition systems, and a logical characterisation of a unary predicate: divergence, the set of states of an LTS which have an infinite path of outgoing τ -steps. The latter is characterised, on image-finite LTSs, by a quantitative logic with only diamond formulas, i.e., the set of formulas is simply the set of words.

Related work

As mentioned above, there are numerous specific results on Hennessy-Milner theorems, which – e.g., in the probabilistic setting as in [7] – can be highly non-trivial. A comprehensive historical treatment is beyond the scope of this paper, which is, instead, broad: it aims at studying these kinds of results in a general, coalgebraic setting.

The case of capturing bisimilarity and behavioural equivalence of coalgebras by modal logics has been very well studied, see [26] for an overview. Expressiveness w.r.t. similarity has been studied in [20], which is close in spirit to our approach, but focuses on the poset case. On a detailed level, the logic for similarity is based on distributive lattices, hence it

uses disjunction; this differs from our example, which only uses conjunction and diamond modalities. Expressiveness of multi-valued coalgebraic logics w.r.t. behavioural equivalence is studied in [3]. In [1], notions of equivalence are extracted from a logic through a variant of Λ -bisimulation [11]. To the best of our knowledge, the current work is the first in the area that connects general coinductive predicates in a fibration to coalgebraic logics.

In the recent [9], the authors prove Hennessy-Milner type theorems for coalgebras including, but going significantly beyond bisimilarity. The logics are related to a semantics obtained from graded monads, and the focus is exclusively on semantic equivalence of different types. In that sense, the scope differs substantially from the current paper, which relates logic to coinductive predicates and where it is essential to relate theories in different ways than equivalence (to cover, e.g., similarity, divergence or logical distance). On the one hand, it appears that none of our examples can be covered immediately in *loc. cit.*; on the other hand, trace equivalence of various kinds can be covered in [9] but not in the current paper.

In [37] a characterisation theorem is shown for fuzzy modal logic, and in [24] for a wide class of behavioural metrics. These papers are not aimed at other kinds of coinductive predicates, and they do not cover the examples in Section 4 (including the behavioural metric for deterministic automata, as we use a much simpler logic than in [24]). Conversely, the question whether the logical characterisation results of [24] can be covered in the current framework is left open. These papers also treat game-based characterisations of bisimilarity, which are studied in a general setting in the recent [23]. The latter paper, however, does not yet feature modal logic explicitly; in fact, the connection is posed there as future work.

2 Preliminaries

The category of sets and functions is denoted by \mathbf{Set} . The powerset functor is denoted by $\mathcal{P}: \mathbf{Set} \rightarrow \mathbf{Set}$, and the finite powerset functor by \mathcal{P}_ω . The diagonal relation on a set X is denoted by $\Delta_X = \{(x, x) \mid x \in X\}$.

Let \mathcal{C} be a category, and $B: \mathcal{C} \rightarrow \mathcal{C}$ a functor. A (B) -coalgebra is a pair (X, γ) where X is an object in \mathcal{C} and $\gamma: X \rightarrow BX$ a morphism. A homomorphism from a coalgebra (X, γ) to a coalgebra (Y, θ) is a morphism $h: X \rightarrow Y$ such that $\theta \circ h = Bh \circ \gamma$. An algebra for a functor $L: \mathcal{D} \rightarrow \mathcal{D}$ on a category \mathcal{D} is a pair (A, α) of an object A in \mathcal{D} and an arrow $\alpha: LA \rightarrow A$.

► **Example 1.** A labelled transition system (LTS) over a set of labels A is a coalgebra (X, γ) for the functor $B: \mathbf{Set} \rightarrow \mathbf{Set}$, $BX = (\mathcal{P}X)^A$. For states $x, x' \in X$ and a label $a \in A$, we sometimes write $x \xrightarrow{a} x'$ for $x' \in \gamma(x)(a)$. Image-finite labelled transition systems are coalgebras for the functor $BX = (\mathcal{P}_\omega X)^A$. A deterministic automaton over an alphabet A is a coalgebra for the functor $B: \mathbf{Set} \rightarrow \mathbf{Set}$, $BX = 2 \times X^A$. For many other examples of state-based systems modelled as coalgebras, see, e.g., [18, 32].

2.1 Coinductive Predicates in a Fibration

We recall the general approach to coinductive predicates in a fibration, starting by briefly presenting how bisimilarity of \mathbf{Set} coalgebras arises in this setting (see [12, 14, 18] for details). Let \mathbf{Rel} be the category where an object is a pair (X, R) consisting of a set X and a relation $R \subseteq X \times X$ on it, and a morphism from (X, R) to (Y, S) is a map $f: X \rightarrow Y$ such that $x R y$ implies $f(x) S f(y)$, for all $x, y \in X$. Below, we sometimes refer to an object (X, R) only by the relation $R \subseteq X \times X$. Any set functor $B: \mathbf{Set} \rightarrow \mathbf{Set}$ gives rise to a functor $\mathbf{Rel}(B): \mathbf{Rel} \rightarrow \mathbf{Rel}$, defined by *relation lifting*:

$$\mathbf{Rel}(B)(R \subseteq X \times X) = \{((B\pi_1)(z), (B\pi_2)(z)) \in BX \times BX \mid z \in BR\}. \quad (1)$$

Given a B -coalgebra (X, γ) , a *bisimulation* is a relation $R \subseteq X \times X$ such that $R \subseteq (\gamma \times \gamma)^{-1}(\text{Rel}(B)(R))$, i.e., if $x R y$ then $\gamma(x) \text{Rel}(B)(R) \gamma(y)$. *Bisimilarity* is the greatest such relation, and equivalently, the greatest fixed point of the monotone map $R \mapsto (\gamma \times \gamma)^{-1}(\text{Rel}(B)(R))$ on the complete lattice of relations on X , ordered by inclusion.

The functor $\text{Rel}(B)$ is a *lifting* of B : it maps a relation on X to a relation on BX . A first step towards generalisation beyond bisimilarity is obtained by replacing $\text{Rel}(B)$ by an arbitrary lifting $\bar{B}: \text{Rel} \rightarrow \text{Rel}$ of B . For instance, for $BX = (\mathcal{P}_\omega X)^A$ one may take

$$\bar{B}(R) = \{(t_1, t_2) \mid \forall a \in A. \forall x \in t_1(a). \exists y \in t_2(a). (x, y) \in R\}. \quad (2)$$

Then, for an LTS $\gamma: X \rightarrow (\mathcal{P}_\omega X)^A$, the greatest fixed point of the monotone map $R \mapsto (\gamma \times \gamma)^{-1} \circ \bar{B}(R)$ is *similarity*. In the same way, by varying the lifting \bar{B} , one can define many different coinductive relations on Set coalgebras.

Yet a further generalisation is obtained by replacing Set by a general category \mathcal{C} , and Rel by a category of “predicates” on \mathcal{C} . A suitable categorical infrastructure for such predicates on \mathcal{C} is given by the notion of *fibration*. This allows us, for instance, to move beyond (Boolean, binary) relations to quantitative relations (e.g., behavioural metrics) or unary predicates. Such examples follow in Section 4; also see, e.g., [12, 5].

To define fibrations, it will be useful to fix some associated terminology first. Let $p: \mathcal{E} \rightarrow \mathcal{C}$ be a functor. If $p(R) = X$ then we say R is *above* X , and similarly for morphisms. The collection of all objects R above a given object X and arrows above the identity id_X form a category, called the *fibre above* X and denoted by \mathcal{E}_X .

- **Definition 2.** A functor $p: \mathcal{E} \rightarrow \mathcal{C}$ is a (poset) fibration if
- each fibre \mathcal{E}_X is a poset category (that is, at most one arrow between every two objects); the corresponding order on objects is denoted by \leq ;
 - for every $f: X \rightarrow Y$ in \mathcal{C} and object S above Y there is a Cartesian morphism $\tilde{f}_S: f^*(S) \rightarrow S$ above f , with the property that for every arrow $g: Z \rightarrow X$, every object R above Z and arrow $h: R \rightarrow S$ above $f \circ g$, there is a unique arrow $k: R \rightarrow f^*(S)$ above g such that $\tilde{f}_S \circ k = h$.

$$\begin{array}{ccc}
 R & \xrightarrow{h} & S \\
 \searrow k & \nearrow \tilde{f}_S & \\
 & f^*(S) & \\
 & \xrightarrow{f} & \\
 & & S
 \end{array}$$

$$\begin{array}{ccc}
 Z & \xrightarrow{f \circ g} & Y \\
 \searrow g & \nearrow f & \\
 & X & \\
 & \xrightarrow{f} & \\
 & & Y
 \end{array}$$

► **Remark 3.** In this paper we only consider poset fibrations, and refer to them simply as fibrations. The usual definition of fibration is more general (e.g., [17]): normally, fibres are not assumed to be posets. Poset fibrations have several good properties, mentioned below. In the application to coinductive predicates, it is customary to work with poset fibrations.

For a morphism $f: X \rightarrow Y$, the assignment $R \mapsto f^*(R)$ gives rise to a functor $f^*: \mathcal{E}_Y \rightarrow \mathcal{E}_X$, called *reindexing along* f . (Note that functors between poset categories are just monotone maps.) We use a strengthening of poset fibrations, following [34, 23].

► **Definition 4.** A poset fibration $p: \mathcal{E} \rightarrow \mathcal{C}$ is called a CLat_\wedge -fibration if (\mathcal{E}_X, \leq) is a complete lattice for every X , and reindexing preserves arbitrary meets.

Any poset fibration p is split: we have $(g \circ f)^* = f^* \circ g^*$ for any morphisms f, g that compose. Further, p is faithful. This captures the intuition that morphisms in \mathcal{E} are morphisms in \mathcal{C} with a certain property; e.g., relation-preserving, or non-expansive (Examples 5, 6). We note that CLat_\wedge -fibrations are instances of topological functors [15]. We use the former, in line with existing related work [12, 23]. This also has the advantage of keeping our results amenable to possible future extensions to a wider class of examples.

► **Example 5.** Consider the *relation fibration* $p: \text{Rel} \rightarrow \text{Set}$, where $p(R \subseteq X \times X) = X$. Reindexing is given by inverse image: for a map $f: X \rightarrow Y$ and a relation $S \subseteq Y \times Y$, we have $f^*(S) = (f \times f)^{-1}(S)$. The functor p is a CLat_\wedge -fibration.

Closely related is the *predicate fibration* $p: \text{Pred} \rightarrow \text{Set}$. An object of Pred is a pair (X, Γ) consisting of a set X and a subset $\Gamma \subseteq X$, and an arrow from (X, Γ) to (Y, Θ) is a map $f: X \rightarrow Y$ such that $x \in \Gamma$ implies $f(x) \in \Theta$. The functor p is given by $p(X, \Gamma) = X$, reindexing is given by inverse image, and p is a CLat_\wedge -fibration as well.

In the relation fibration, we sometimes refer to an object $(X, R \subseteq X^2)$ simply by R , and similarly in the predicate fibration.

► **Example 6.** Let \mathcal{V} be a complete lattice. Define the category $\text{Rel}_\mathcal{V}$ as follows: an object is a pair (X, d) where X is a set and a function $d: X \times X \rightarrow \mathcal{V}$, and a morphism from (X, d) to (Y, e) is a map $f: X \rightarrow Y$ such that $d(x, y) \leq e(f(x), f(y))$. The forgetful functor $p: \text{Rel}_\mathcal{V} \rightarrow \text{Set}$ is a CLat_\wedge -fibration, where reindexing along $f: X \rightarrow Y$ is given by $f^*(Y, e) = (X, e \circ f \times f)$.

For $\mathcal{V} = 2 = \{0, 1\}$ with the usual order $0 \leq 1$, $\text{Rel}_\mathcal{V}$ coincides with Rel . Another example is given by the closed interval $\mathcal{V} = [0, 1]$, with the *reverse* order. Then, a morphism from (X, d) to (Y, e) is a *non-expansive map* $f: X \rightarrow Y$, that is, s.t. $e(f(x), f(y)) \leq d(x, y)$ (with \leq the usual order, i.e., where 0 is the smallest). This instance will be denoted by $\text{Rel}_{[0,1]}$.

Liftings and Coinductive Predicates

Let $p: \mathcal{E} \rightarrow \mathcal{C}$ be a fibration, and $B: \mathcal{C} \rightarrow \mathcal{C}$ a functor. A functor $\overline{B}: \mathcal{E} \rightarrow \mathcal{E}$ is called a *lifting* of B if $p \circ \overline{B} = B \circ p$. In that case, \overline{B} restricts to a functor $\overline{B}_X: \mathcal{E}_X \rightarrow \mathcal{E}_{BX}$, for any X in \mathcal{C} .

A lifting \overline{B} of B gives rise to an abstract notion of coinductive predicate, as follows. For any B -coalgebra (X, γ) there is the functor, i.e., monotone function defined by $\gamma^* \circ \overline{B}_X: \mathcal{E}_X \rightarrow \mathcal{E}_X$. We think of post-fixed points of $\gamma^* \circ \overline{B}_X$ as *invariants*, generalising *bisimulations*. If p is a CLat_\wedge -fibration, then $\gamma^* \circ \overline{B}_X$ has a greatest fixed point $\nu(\gamma^* \circ \overline{B}_X)$, which is also the greatest post-fixed point. It is referred to as the *coinductive predicate* defined by \overline{B} on γ .

► **Example 7.** First, for a Set functor $B: \text{Set} \rightarrow \text{Set}$, recall the lifting $\text{Rel}(B)$ of B defined in the beginning of this section. We refer to $\text{Rel}(B)$ as the *canonical relation lifting* of B . For a coalgebra (X, γ) , a post-fixed point of the operator $\gamma^* \circ \text{Rel}(B)_X$ is a bisimulation, as explained above. The coinductive predicate $\nu(\gamma^* \circ \text{Rel}(B)_X)$ defined by $\text{Rel}(B)$ is bisimilarity. Another example is given by the lifting \overline{B} for similarity defined in the beginning of this section, which we further study in Section 4. In that section we also define a unary predicate, divergence, making use of the predicate fibration. Coinductive predicates in the fibration $\text{Rel}_{[0,1]}$ can be thought of as *behavioural distances*, providing a quantitative analogue of bisimulations, measuring the distances between states. A simple example on deterministic automata is studied in Section 4.1.

► **Remark 8.** In the quantitative examples, such as $\text{Rel}_{[0,1]}$, one can replace the latter by a category with more structure, such as the category of pseudometrics and non-expansive maps.

Similarly, one can replace Rel by the category of equivalence relations. Defining liftings then requires slightly more work, and since we use fibrations to *define* coinductive predicates, this unnecessarily complicates matters. Therefore, we do not use such categories in our examples.

We sometimes need the notion of *fibration map*: if \bar{B} is a lifting of B , the pair (\bar{B}, B) is called a fibration map if $(Bf)^* \circ \bar{B}_Y = \bar{B}_X \circ f^*$ for any arrow $f: X \rightarrow Y$ in \mathcal{C} . If B preserves weak pullbacks, then $(\text{Rel}(B), B)$ is a fibration map [18] in the relation fibration (Example 5).

2.2 Coalgebraic Modal Logic

We recall a general approach to coalgebraic modal logic, in the context of a contravariant adjunction [30, 22, 19]. We assume the following setting, involving an adjunction $P \dashv Q$ and a natural transformation $\delta: BQ \Rightarrow QL$:

$$B \begin{array}{c} \circlearrowleft \\ \circlearrowright \end{array} \mathcal{C} \begin{array}{c} \xrightarrow{P} \\ \perp \\ \xleftarrow{Q} \end{array} \mathcal{D}^{\text{op}} \begin{array}{c} \circlearrowright \\ \circlearrowleft \end{array} L \quad \text{with} \quad BQ \xrightarrow{\delta} QL \quad (3)$$

In this context, a *logic* for B -coalgebras is a pair (L, δ) as above. The functor $L: \mathcal{D} \rightarrow \mathcal{D}$ represents the syntax of the modalities. It is assumed to have an initial algebra $\alpha: L\Phi \xrightarrow{\cong} \Phi$, which represents the set (or other structure) of formulas of the logic. The natural transformation δ gives the one-step semantics. It can equivalently be presented in terms of its *mate* $\hat{\delta}: LP \Rightarrow PB$, which is perhaps more common in the literature. However, we will formulate adequacy and expressiveness in terms of the current presentation of δ .

Let (X, γ) be a B -coalgebra. The semantics $\llbracket _ \rrbracket$ of a logic (L, δ) arises by initiality of α , making use of the mate $\hat{\delta}$, as the unique map making the diagram on the left below commute.

$$\begin{array}{ccc} L\Phi \xrightarrow{L\llbracket _ \rrbracket} LPX \xrightarrow{\hat{\delta}} PBX & & X \xrightarrow{\text{th}} Q\Phi \\ \alpha \downarrow & & \gamma \downarrow \\ \Phi \xrightarrow{\exists! \llbracket _ \rrbracket} PX & & BX \xrightarrow{B\text{th}} BQ\Phi \xrightarrow{\delta} QL\Phi \\ & & \downarrow Q\alpha \end{array}$$

The *theory map* $\text{th}: X \rightarrow Q\Phi$ is defined as the transpose of $\llbracket _ \rrbracket$. It is the unique map making the diagram on the right above commute.

► **Example 9.** Let $\mathcal{C} = \mathcal{D} = \text{Set}$, $P = Q = 2^-$ the contravariant powerset functor, and $BX = 2 \times X^A$. We define a simple logic for B -coalgebras, where formulas are just words over A . To this end, let $LX = A \times X + 1$. The initial algebra of L is the set A^* of words. Define $\delta: BQ \Rightarrow QL$ on a component X as follows:

$$\delta_X: 2 \times (2^X)^A \rightarrow 2^{A \times X + 1} \quad \delta_X(o, t)(u) = \begin{cases} o & \text{if } u = * \in 1 \\ t(a)(x) & \text{if } u = (a, x) \in A \times X \end{cases}$$

For a coalgebra $\langle o, t \rangle: X \rightarrow 2 \times X^A$, the associated theory map $\text{th}: X \rightarrow 2^{A^*}$ is given by $\text{th}(x)(\varepsilon) = o(x)$ and $\text{th}(x)(aw) = \text{th}(t(x)(a))(w)$ for all $x \in X$, $a \in A$, $w \in A^*$. This is, of course, the usual semantics of deterministic automata.

In the above example, the logic does not contain propositional connectives; this is reflected by the choice $\mathcal{D} = \text{Set}$. To add those, one chooses a category of algebras for \mathcal{D} . For instance, Boolean algebras are a standard choice for propositional logic, and in Section 4 we use the category of semilattices to represent conjunction. In fact, if one is only interested in defining the semantics of the logic, one can simply work with algebras for a signature; this is supported by the adjunctions presented in the next subsection. We outline in the next subsection how this can be used to represent the propositional part of a real-valued modal logic.

2.3 Contravariant Adjunctions

In this subsection we discuss several adjunctions that we use for presenting coalgebraic logic as above, and will allow us in Section 4 to demonstrate that a large variety of concrete examples is covered by our framework. In all cases, the adjunctions that we use for the logic are generated by an object Ω of “truth values”. In fact, we believe all of the dual adjunctions listed in this section are instances of the so-called concrete dualities from [31] where Ω is the dualising object inducing the adjunction.

For a simple but useful class of such adjunctions, let \mathcal{D} be a category with products, and Ω an object in \mathcal{D} . Then there is an adjunction

$$P \dashv Q: \mathbf{Set} \rightleftarrows \mathcal{D}^{\text{op}} \quad \text{where } PX = \Omega^X \text{ and } QX = \text{Hom}(X, \Omega), \quad (4)$$

where Ω^X is the X -fold product of Ω .

► **Example 10.** To illustrate the usefulness of this simple adjunction, consider the real-valued coalgebraic modal logics from [24]. The set Φ of formulas of these logics is given by the following definition that is indexed by a set \mathfrak{E} of modal operators:

$$\Phi ::= \top \mid [\mathfrak{e}]\varphi, \mathfrak{e} \in \mathfrak{E} \mid \min(\varphi_1, \varphi_2) \mid \neg\varphi \mid \varphi \ominus q, q \in \mathbb{Q} \cap [0, \top]$$

where \ominus is interpreted as truncated subtraction on $[0, \top]$ given by $p \ominus q := \max(p - q, 0)$, \min is interpreted as minimum and where negation on $[0, \top]$ is defined as $\neg q := \top - q$. Describing the category of L -algebras that precisely represents a given logic (i.e., where the initial algebra corresponds to the set of formulas modulo equivalence) is in general nontrivial. For studying expressivity, however, it is sufficient to consider formulas and their semantics, i.e., expressivity of a real-valued logic for B -coalgebras for some functor $B: \mathbf{Set} \rightarrow \mathbf{Set}$ can be studied by considering the dual adjunction

$$\begin{array}{ccc} B \circlearrowleft \mathbf{Set} & \begin{array}{c} \xrightarrow{P=[0, \top]^-} \\ \perp \\ \xleftarrow{Q=\text{Hom}(-, [0, \top])} \end{array} & \text{Alg}(\Sigma)^{\text{op}} \circlearrowright L \end{array}$$

where $\Sigma X = 1 + X^2 + X + X \times (\mathbb{Q} \cap [0, \top])$ and $L(A) = T_\Sigma(\{[\mathfrak{e}]a \mid a \in A, \mathfrak{e} \in \mathfrak{E}\})$ with $T_\Sigma(G)$ denoting the free Σ -algebra over a set G of generators.

Another class of adjunctions we use relates Rel to categories of algebras. To formulate it, we assume:

- \mathcal{V} is a complete lattice of distance values,
- Ω is a bounded poset of truth values,
- $\Sigma: \mathbf{Set} \rightarrow \mathbf{Set}$ is a functor,
- $a_\Omega: \Sigma\Omega \rightarrow \Omega$ is a Σ -algebra,
- $(\Omega, R_\Omega: \Omega \times \Omega \rightarrow \mathcal{V}) \in \text{Rel}_\mathcal{V}$, and
- Σ has a lifting $\bar{\Sigma}: \text{Rel}_\mathcal{V} \rightarrow \text{Rel}_\mathcal{V}$ such that
 1. there is a morphism $\bar{a}_\Omega: \bar{\Sigma}R_\Omega \rightarrow R_\Omega$ above a_Ω and
 2. for any $(X, R), (Y, S) \in \text{Rel}_\mathcal{V}$ there is a morphism $\bar{\text{st}}_{R, S}: R \times \bar{\Sigma}S \rightarrow \bar{\Sigma}(R \times S)$ above the strength map $\text{st}_{X, Y}: X \times \Sigma Y \rightarrow \Sigma(X \times Y)$ for the set functor Σ .

► **Proposition 11.** *Under the above assumptions there is a dual adjunction*

$$\begin{array}{ccc} \text{Rel}_\mathcal{V} & \begin{array}{c} \xrightarrow{\text{Hom}(_, R_\Omega)} \\ \perp \\ \xleftarrow{\text{Hom}(_, a_\Omega)} \end{array} & \text{Alg}(\Sigma)^{\text{op}} \end{array} \quad (5)$$

► **Corollary 12.** *In the above scenario, assume that Σ is a polynomial functor and $\bar{\Sigma}: \text{Rel}_V \rightarrow \text{Rel}_V$ is interpreted to be the canonical lifting of Σ that interprets products and coproducts occurring in Σ as products and coproducts in Rel_V , respectively. Then the condition on $\text{st}_{R,S}$ is always satisfied and the dual adjunction from (5) exists if there is a morphism $\bar{a}_\Omega: \bar{\Sigma}R_\Omega \rightarrow R_\Omega$ above a_Ω .*

The following remark is obvious, but at the same time useful for concrete examples.

► **Remark 13.** In the above cases, let \mathcal{C} be a full subcategory of Rel_V and \mathcal{D} a full subcategory of $\text{Alg}(\Sigma)$ such that $\text{Hom}(-, a_\Omega)$ and $\text{Hom}(-, R_\Omega)$ restrict to functors of type $\mathcal{D} \rightarrow \mathcal{C}$ and of type $\mathcal{C} \rightarrow \mathcal{D}$, respectively. Then the above dual adjunction restricts to a dual adjunction between \mathcal{C} and \mathcal{D} .

3 Abstract Framework: Adequacy & Expressivity

In this section, we define when a logic is adequate and expressive with respect to a coinductive predicate, and provide sufficient conditions on the logic. Coinductive predicates are expressed abstractly via fibrations and functor lifting, and logic via a contravariant adjunction. Therefore, we make the following assumptions.

► **Assumption 14.** Throughout this section, we assume:

1. (*Type of coalgebra*) An endofunctor $B: \mathcal{C} \rightarrow \mathcal{C}$ on a category \mathcal{C} ;
2. (*Coinductive predicate*) A CLat_\wedge -fibration $p: \mathcal{E} \rightarrow \mathcal{C}$ and a lifting $\bar{B}: \mathcal{E} \rightarrow \mathcal{E}$ of B ;
3. (*Coalgebraic logic*) An adjunction $P \dashv Q: \mathcal{C} \rightleftarrows \mathcal{D}^{\text{op}}$, a functor $L: \mathcal{D} \rightarrow \mathcal{D}$ with an initial algebra $\alpha: L(\Phi) \xrightarrow{\cong} \Phi$, and a natural transformation $\delta: BQ \Rightarrow QL$.

As explained in the introduction, to formulate adequacy and expressiveness, we need one more crucial ingredient: an object that stipulates how collections of formulas should be compared. In the abstract fibrational setting, we assume an object above $Q\Phi$; more systematically, a functor \bar{Q} above Q .

► **Definition 15 (Adequacy and Expressivity).** *Let $\bar{Q}: \mathcal{D}^{\text{op}} \rightarrow \mathcal{E}$ be a functor such that $p \circ \bar{Q} = Q$. We say the logic (L, δ) is*

- adequate if $\nu(\gamma^* \circ \bar{B}_X) \leq \text{th}^*(\bar{Q}\Phi)$ for every B -coalgebra (X, γ) ;
- expressive if $\nu(\gamma^* \circ \bar{B}_X) \geq \text{th}^*(\bar{Q}\Phi)$ for every B -coalgebra (X, γ) .

When we need to refer to the functors \bar{Q} or \bar{B} explicitly, we speak about adequacy and expressivity *via \bar{Q} w.r.t. \bar{B}* . Examples follow in Section 3.2, where classical expressivity and adequacy w.r.t. bisimilarity is recovered, and Section 4, where other instances are treated.

► **Remark 16.** Definition 15 can be generalised to arbitrary poset fibrations, not necessarily assuming complete lattice structure on the fibres, as follows. Adequacy means that for any B -coalgebra (X, γ) , if $R \leq \gamma^* \circ \bar{B}_X(R)$ then $R \leq \text{th}^*(\bar{Q}\Phi)$. Expressivity means that for any B -coalgebra (X, γ) , we have $\text{th}^*(\bar{Q}\Phi) \leq R$ for some R with $R \leq \gamma^* \circ \bar{B}_X(R)$. In fact, with these definitions, if (L, δ) is both adequate and expressive then $\gamma^* \circ \bar{B}_X$ has a greatest fixed point, given by $\text{th}^*(\bar{Q}\Phi)$. We prefer to work with CLat_\wedge -fibrations, since the definition is slightly simpler, and it covers all our examples.

3.1 Sufficient conditions for expressivity and adequacy

The results below give conditions on \bar{B} , \bar{Q} and primarily the one-step semantics δ that guarantee expressivity (Theorem 19) and adequacy (Theorem 18). For simplicity we fix the functor \bar{Q} .

► **Assumption 17.** In the remainder of this section we assume a functor $\overline{Q}: \mathcal{D}^{\text{op}} \rightarrow \mathcal{E}$ such that $p \circ \overline{Q} = Q$.

For adequacy, the main idea is to require sufficient conditions to lift δ to a logic for \overline{B} .

► **Theorem 18.** *Suppose that*

1. $\overline{B}\overline{Q}X \leq \delta_X^*(\overline{Q}LX)$ for every object X in \mathcal{D} , and
2. the functor \overline{Q} has a left adjoint.

Then (L, δ) is adequate.

Proof. The first assumption yields a natural transformation $\overline{\delta}: \overline{B}\overline{Q} \Rightarrow \overline{Q}L$, defined on a component X by

$$\overline{\delta}_X = \left(\overline{B}\overline{Q}X \longrightarrow \delta_X^*(\overline{Q}LX) \xrightarrow{\tilde{\delta}} \overline{Q}LX \right)$$

where the left arrow is the inclusion $\overline{B}\overline{Q}X \leq \delta_X^*(\overline{Q}LX)$, and the right arrow $\tilde{\delta}$ is the Cartesian morphism to $\overline{Q}LX$ above δ_X . It follows that $\overline{\delta}_X$ is above δ_X . Further, naturality follows from p being faithful (as it is a poset fibration, see Section 2.1) and naturality of δ . Observe that we have thus established $(L, \overline{\delta})$ as a logic for \overline{B} -coalgebras, via the adjunction $\overline{P} \dashv \overline{Q}$.

Now let (X, γ) be a B -coalgebra, and $R = \nu(\gamma^* \circ \overline{B}_X)$. Then, in particular, $R \leq \gamma^* \circ \overline{B}_X(R)$, which is equivalent to a coalgebra $\overline{\gamma}: R \rightarrow \overline{B}R$ above $\gamma: X \rightarrow BX$. The logic $(L, \overline{\delta})$ gives us a theory map \overline{th} of $(R, \overline{\gamma})$ as the unique map making the following diagram commute.

$$\begin{array}{ccc} R & \xrightarrow{\overline{th}} & \overline{Q}\Phi \\ \overline{\gamma} \downarrow & & \downarrow \overline{Q}\alpha \\ \overline{B}R & \xrightarrow{\overline{B}\overline{th}} \overline{B}\overline{Q}\Phi \xrightarrow{\overline{\delta}} & \overline{Q}L\Phi \end{array}$$

Since $p \circ \overline{Q} = Q$ and $p(\overline{\delta}_\Phi) = \delta_\Phi$, it follows that $p(\overline{th})$ equals the theory map th of (X, γ) . Hence $R \leq th^*(\overline{Q}\Phi)$ as required. ◀

Expressivity requires the converse inequality of the one in Theorem 18, but only on one component: the carrier Φ of the initial algebra. Further, the conditions include that (\overline{B}, B) is a fibration map. In particular, for the canonical relation lifting $\text{Rel}(B)$ this means that B should preserve weak pullbacks; this case is explained in more detail in Section 3.2.

► **Theorem 19.** *Suppose (\overline{B}, B) is a fibration map. If $\delta_\Phi^*(\overline{Q}L\Phi) \leq \overline{B}\overline{Q}\Phi$ then (L, δ) is expressive.*

Proof. Let (X, γ) be a B -coalgebra, with th the associated theory map. We show that $th^*(\overline{Q}\Phi)$ is a post-fixed point of $\gamma^* \circ \overline{B}_X$:

$$\begin{aligned} th^*(\overline{Q}\Phi) &= (Q(\alpha^{-1}) \circ \delta_\Phi \circ Bth \circ \gamma)^*(\overline{Q}\Phi) \\ &= \gamma^* \circ (Bth)^* \circ \delta_\Phi^* \circ Q(\alpha^{-1})^*(\overline{Q}\Phi) \\ &= \gamma^* \circ (Bth)^* \circ \delta_\Phi^*(\overline{Q}L\Phi) && \text{(follows from } \alpha^{-1} \text{ being an iso)} \\ &\leq \gamma^* \circ (Bth)^*(\overline{B}\overline{Q}\Phi) && \text{(assumption)} \\ &= \gamma^* \circ \overline{B}_X \circ th^*(\overline{Q}\Phi) && \text{((}\overline{B}, B\text{) fibration map)} \end{aligned}$$

Expressivity follows since $\nu(\gamma^* \circ \overline{B}_X)$ is the greatest post-fixed point. ◀

3.2 Adequacy and Expressivity w.r.t. Bisimilarity

In the setting of coalgebraic modal logic recalled in Section 2.2, Klin [22] proved that

1. the theory map th of a coalgebra (X, γ) factors through coalgebra morphisms from (X, γ) ;
2. if δ has monic components, then th factors as a coalgebra morphism followed by a mono.

The first item can be seen as adequacy w.r.t. behavioural equivalence (i.e., identification by a coalgebra morphism), and the second as expressivity.

In the current section we revisit this result for **Set** functors, as a sanity check of Definition 15. To this end, we focus on the canonical lifting $\text{Rel}(B): \text{Rel} \rightarrow \text{Rel}$ of a **Set** functor B in the relation fibration, so that, for a coalgebra (X, γ) , $\nu(\gamma^* \circ \text{Rel}(B)_X)$ is coalgebraic bisimilarity. We have to restrict to weak pullback preserving functors B . The reason is that expressive logics typically capture behavioural equivalence rather than bisimilarity. As is well-known, for weak pullback preserving functors, the two coincide [32].

To obtain the appropriate notion of adequacy and expressivity, we need to compare collections of formulas for equality. Therefore, the functor \overline{Q} in Definition 15 will be instantiated with $\overline{Q}X = (QX, \Delta_{QX})$ where Δ_{QX} denotes the diagonal. Then, for a coalgebra (X, γ) , $th^*(\overline{Q}\Phi)$ is the set of all pairs of states (x, y) such that $th(x) = th(y)$. Adequacy then means that for every coalgebra (X, γ) , bisimilarity is contained in $th^*(\overline{Q}\Phi)$, i.e., if x is bisimilar to y then $th(x) = th(y)$. Expressivity is the converse implication.

To state and prove the result, let $\text{Eq}: \text{Set} \rightarrow \text{Rel}$ be the functor given by $\text{Eq}(X) = \Delta_X$. This functor has a left adjoint $\text{Quot}: \text{Rel} \rightarrow \text{Set}$, which maps a relation $R \subseteq X \times X$ to the quotient of X by the least equivalence relation containing R (cf. [14]).

► **Proposition 20** (Adequacy and expressivity w.r.t. bisimilarity). *Consider the relation fibration $p: \text{Rel} \rightarrow \text{Set}$, let $B: \text{Set} \rightarrow \text{Set}$ be a weak pullback preserving functor, let $P \dashv Q: \text{Set} \rightleftarrows \mathcal{D}^{\text{op}}$ for some category \mathcal{D} , $L: \mathcal{D} \rightarrow \mathcal{D}$ a functor with an initial algebra and $\delta: BQ \Rightarrow QL$. Then*

1. (L, δ) is adequate w.r.t. $\text{Rel}(B)$;
2. if δ is componentwise injective, then (L, δ) is expressive w.r.t. $\text{Rel}(B)$, via $\overline{Q} = \text{Eq} \circ Q$.

Proof. For adequacy, we use Theorem 18. By composition of adjoints, $P \circ \text{Quot}$ is a left adjoint to $\text{Eq} \circ Q$. It will be useful to simplify $\text{Rel}(B) \circ \text{Eq} \circ QX$ and $\delta_X^*(\text{Eq} \circ Q \circ LX)$:

$$\text{Rel}(B) \circ \text{Eq} \circ QX = \text{Rel}(B)(\Delta_{QX}) = \Delta_{BQX}, \quad (6)$$

$$\delta_X^*(\text{Eq} \circ Q \circ LX) = (\delta_X \times \delta_X)^{-1}(\Delta_{QLX}), \quad (7)$$

using that $\text{Rel}(B) \circ \text{Eq} = \text{Eq} \circ B$ in the first equality (e.g., [18]). The remaining hypothesis of Theorem 18 is that $\text{Rel}(B) \circ \text{Eq} \circ QX \leq \delta_X^*(\text{Eq} \circ Q \circ LX)$ for all X , i.e., $\Delta_{BQX} \subseteq (\delta_X \times \delta_X)^{-1}(\Delta_{QLX})$, which is trivial.

For expressivity, we use Theorem 19. Since B preserves weak pullbacks, $(\text{Rel}(B), B)$ is a fibration map. We need to prove that $\delta_\Phi^*(\text{Eq} \circ Q \circ L\Phi) \leq \text{Rel}(B) \circ \text{Eq} \circ Q\Phi$, which amounts to the inclusion

$$(\delta_\Phi \times \delta_\Phi)^{-1}(\Delta_{QL\Phi}) \subseteq \Delta_{BQ\Phi}$$

But this is equivalent to injectivity of δ_Φ . ◀

4 Examples

In this section we instantiate the abstract framework to three concrete examples: a behavioural metric on deterministic automata (Section 4.1), captured by $[0, 1]$ -valued tests; a unary predicate on transition systems (Section 4.2); and similarity of transition systems, captured by a logic with conjunction and diamond modalities (Section 4.3).

4.1 Shortest distinguishing word distance

We study a simple behavioural distance on deterministic automata: for two states x, y and a fixed constant c with $0 < c < 1$, the distance is given by c^n , where n is the length of the smallest word accepted from one state but not the other. Following [4], this is referred to as the *shortest distinguishing word distance*, and, for an automaton with state space X , denoted by $d_{sdw}: X \times X \rightarrow [0, 1]$.

Formally, fix a finite alphabet A , and consider the functor $B: \mathbf{Set} \rightarrow \mathbf{Set}$, $BX = 2 \times X^A$ of deterministic automata. We make use of the fibration $p: \mathbf{Rel}_{[0,1]} \rightarrow \mathbf{Set}$, and define the lifting $\bar{B}: \mathbf{Rel}_{[0,1]} \rightarrow \mathbf{Rel}_{[0,1]}$ by

$$\bar{B}(X, d) = \left(BX, ((o_1, t_1), (o_2, t_2)) \mapsto \begin{cases} 1 & \text{if } o_1 \neq o_2 \\ c \cdot \max_{a \in A} \{d(t_1(a), t_2(a))\} & \text{otherwise} \end{cases} \right)$$

The shortest distinguishing word distance d_{sdw} on a deterministic automaton $\gamma: X \rightarrow 2 \times X^A$ is the greatest fixed point $\nu(\gamma^* \circ \bar{B}_X)$.

For an associated logic, we simply use words over A as formulas, and define a satisfaction relation which is weighted in $[0, 1]$. Consider the following setting.

$$B=2 \times \text{Id}^A \curvearrowright \mathbf{Set} \begin{array}{c} \xrightarrow{P=[0,1]^-} \\ \perp \\ \xleftarrow{Q=[0,1]^-} \end{array} \mathbf{Set}^{\text{op}} \curvearrowleft L=A \times \text{Id}+1 \quad \text{with} \quad B([0, 1]^-) \xrightarrow{\delta} [0, 1]^{L^-}$$

The initial algebra of L is the set of words A^* . The natural transformation δ is given by $\delta_X: 2 \times ([0, 1]^X)^A \rightarrow [0, 1]^{A \times X+1}$,

$$\delta_X(o, t)(u) = \begin{cases} o & \text{if } u = * \in 1 \\ c \cdot t(a)(x) & \text{if } u = (a, x) \in A \times X \end{cases}$$

which is a quantitative, discounted version of the Boolean-valued logic in Example 9. The logic (L, δ) defines, for any deterministic automaton $\langle o, t \rangle: X \rightarrow 2 \times X^A$, a theory map $th: X \rightarrow [0, 1]^{A^*}$, given by

$$th(x)(\varepsilon) = o(x) \quad \text{and} \quad th(x)(aw) = c \cdot th(t(x)(a))(w),$$

for all $x \in X$, $a \in A$, $w \in A^*$.

We characterise the shortest distinguishing word distance with the above logic, by instantiating and proving adequacy and expressivity. Define

$$\bar{Q}: \mathbf{Set}^{\text{op}} \rightarrow \mathbf{Rel}_{[0,1]}, \quad \bar{Q}(X) = \left([0, 1]^X, (\phi_1, \phi_2) \mapsto \sup_{x \in X} |\phi_1(x) - \phi_2(x)| \right).$$

Technically, this functor is given by mapping a set X to the X -fold product of the object $\overline{[0, 1]} = ([0, 1], (r, s) \mapsto |r - s|)$. It follows immediately that \bar{Q} has a left adjoint, mapping (X, d) to $\mathbf{Hom}((X, d), \overline{[0, 1]})$, see Equation 4. This will be useful for proving adequacy below.

The functor \bar{Q} yields a “logical distance” between states $x, y \in X$, given by $th^*(\bar{Q}\Phi)$. We abbreviate it by $d_{log}: X \times X \rightarrow [0, 1]$. Explicitly, we have

$$d_{log}(x, y) = \sup_{w \in A^*} |th(x)(w) - th(y)(w)|. \quad (8)$$

Instantiating Definition 15, the logic (L, δ) is

26:12 Expressive Logics for Coinductive Predicates

- *adequate* if $d_{sdw} \geq d_{log}$, and
- *expressive* if $d_{sdw} \leq d_{log}$.

Here \leq is the usual order on $[0, 1]$, with 0 the least element (the order in $\text{Rel}_{[0,1]}$ is reversed).

To prove adequacy and expressivity, we use Theorem 18 and Theorem 19. The functor \overline{Q} has a left adjoint, as explained above. Further, (\overline{B}, B) is a fibration map [4]. We prove the remaining hypotheses of both propositions by showing the equality $\overline{B}\overline{Q}X = \delta_X^*(\overline{Q}LX)$ for every object X in \mathcal{D} . To this end, we compute (suppressing the carrier set BQX):

$$\begin{aligned}
& \delta_X^*(\overline{Q}LX) \\
= & \left((o_1, t_1), (o_2, t_2) \right) \mapsto \sup_{u \in A \times X + 1} |\delta_X(o_1, t_1)(u) - \delta_X(o_2, t_2)(u)| \\
= & \left((o_1, t_1), (o_2, t_2) \right) \mapsto \begin{cases} 1 & \text{if } o_1 \neq o_2 \\ \sup_{u \in A \times X} |\delta_X(o_1, t_1)(u) - \delta_X(o_2, t_2)(u)| & \text{otherwise} \end{cases} \\
= & \left((o_1, t_1), (o_2, t_2) \right) \mapsto \begin{cases} 1 & \text{if } o_1 \neq o_2 \\ \sup_{(a,x) \in A \times X} |c \cdot t_1(a)(x) - c \cdot t_2(a)(x)| & \text{otherwise} \end{cases} \\
= & \left((o_1, t_1), (o_2, t_2) \right) \mapsto \begin{cases} 1 & \text{if } o_1 \neq o_2 \\ c \cdot \max_{a \in A} \sup_{x \in X} |t_1(a)(x) - t_2(a)(x)| & \text{otherwise} \end{cases} \\
= & \overline{B}\overline{Q}X
\end{aligned}$$

Hence, the logic (L, δ) is adequate and expressive w.r.t. the shortest distinguishing word distance, i.e., d_{sdw} coincides with the logical distance d_{log} given in Equation 8.

4.2 Divergence of processes

A state of an LTS is said to be *diverging* if there exists an infinite path of τ -transitions starting at that state. To model this predicate, let $B: \text{Set} \rightarrow \text{Set}$, $BX = (\mathcal{P}_\omega X)^A$, where A is a set of labels containing the symbol $\tau \in A$. Consider the predicate fibration $p: \text{Pred} \rightarrow \text{Set}$, and define the lifting $\overline{B}: \text{Pred} \rightarrow \text{Pred}$ by

$$\overline{B}(X, \Gamma) = ((\mathcal{P}_\omega X)^A, \{t \mid \exists x \in \Gamma. x \in t(\tau)\}).$$

The coinductive predicate defined by \overline{B} on a B -coalgebra (X, γ) is the set of diverging states:

$$\nu(\gamma^* \circ \overline{B}_X) = (X, \{x \mid x \text{ is diverging}\}).$$

Now, we want to prove in our framework of adequacy and expressivity that x is diverging iff for every $n \in \mathbb{N}$ there is a finite path of τ -steps starting in x , i.e., $x \models \langle \tau \rangle^n \top$ for every n . The proof relies on two main observations:

- if x satisfies infinitely many formulas of $\langle \tau \rangle^n \top$, then one of its τ -successors does, too;
- if a state x satisfies $\langle \tau \rangle^n \top$ for some n then x satisfies $\langle \tau \rangle^m \top$ for all $0 \leq m \leq n$.

Combined, one can then give a coinductive proof, showing that if the current state satisfies all formulas of the form $\langle \tau \rangle^n \top$ then one of its τ -successors also satisfies all these formulas.

We make this argument precise by casting it into the abstract framework. First, for the logic, we have the following setting:

$$\begin{array}{ccc}
B = (\mathcal{P}_\omega -)^A & \begin{array}{c} \curvearrowright \text{Set} \\ \xrightarrow{P=2^-} \\ \perp \\ \xleftarrow{Q=\text{Hom}(-,2)} \end{array} & \text{Pos}^{\text{op}} \curvearrowright \\
& & L = \text{Id}_\top \quad \text{with} \quad B\text{Hom}(-, 2) \xrightarrow{\delta} \text{Hom}(L-, 2)
\end{array}$$

Here Pos is the category of posets and monotone maps, and $2 = \{0, 1\}$ is the poset given by the order $0 \leq 1$. For a poset S , $\text{Hom}(S, 2)$ is then the set of *upwards closed* subsets of S .

The functor $LS = S_{\top}$ is defined on a poset S by adjoining a new top element \top , i.e., the carrier is $S + \{\top\}$ and \top is strictly above all elements of S . The initial algebra Φ of L is the set of natural numbers, representing the formulas of the form $\langle \tau \rangle^n \top$, linearly ordered, with 0 the top element. The choice of \mathbf{Pos} means that the set $\text{Hom}(\Phi, 2)$ used to represent the theory of a state $x \in X$ consists of upwards closed sets (so closed under lower natural numbers in the usual ordering), corresponding to the second observation above concerning the set of formulas satisfied by x .

The natural transformation δ is given by $\delta_S: (\mathcal{P}_\omega \text{Hom}(S, 2))^A \rightarrow \text{Hom}(S_{\top}, 2)$,

$$\delta_S(t)(x) = \begin{cases} 1 & \text{if } x = \top \\ \bigvee_{\phi \in t(\tau)} \phi(x) & \text{otherwise} \end{cases}.$$

To show that this is well-defined, suppose $x, y \in S_{\top}$ with $x \leq y$, and suppose $\delta_S(t)(x) = 1$. If $x = \top$ then $y = \top$, so $\delta_S(t)(y) = 1$. Otherwise, there is $\phi \in \text{Hom}(S, 2)$ such that $\phi \in t(\tau)$ and $\phi(x) = 1$. Since ϕ is upwards closed, $\phi(y) = 1$ and consequently $\delta_S(t)(y) = 1$ as needed.

Now, the theory map $th: X \rightarrow \text{Hom}(\Phi, 2)$ is given by $th(x)(n) = 1$ iff there exists a path of τ -steps of length n from x . We define

$$\bar{Q}: \mathbf{Pos}^{\text{op}} \rightarrow \mathbf{Pred}, \quad \bar{Q}(S) = (\text{Hom}(S, 2), \{\phi \mid \forall x \in S. \phi(x) = 1\}).$$

Instantiating Definition 15, *adequacy* means that if x is diverging, then $x \models \langle \tau \rangle^n \top$ for all n ; and expressivity is the converse.

We start with proving adequacy, using Theorem 18. The left adjoint \bar{P} is given by $\bar{P}(X, \Gamma) = (\text{Hom}((X, \Gamma), (2, \{1\})), \{(\phi_1, \phi_2) \mid \forall x \in X. \phi_1(x) \leq \phi_2(x)\})$. It remains to prove that $\bar{B}\bar{Q}(S) \leq \delta_S^*(\bar{Q}LS)$ for all S . To this end, we observe $BQS = (\mathcal{P}_\omega(\text{Hom}(S, 2)))^A$ and compute:

$$\begin{aligned} \delta_S^*(\bar{Q}LS) &= \{t \mid \delta_S(t) \in \bar{Q}LS\} \\ &= \{t \mid \forall x \in S_{\top}. \delta_S(t)(x) = 1\} \\ &= \{t \mid \forall x \in S. \delta_S(t)(x) = 1\} \\ &= \{t \mid \forall x \in S. \bigvee_{\phi \in t(\tau)} \phi(x) = 1\} \end{aligned}$$

and $\bar{B}\bar{Q}(S) = \{t \mid (\lambda x.1) \in t(\tau)\}$. The needed inclusion is now trivial.

For expressivity we have to prove the reverse inclusion with $S = \Phi$, i.e.,

$$\{t \in (\mathcal{P}_\omega(\text{Hom}(\Phi, 2)))^A \mid \forall x \in \Phi. \bigvee_{\phi \in t(\tau)} \phi(x) = 1\} \subseteq \{t \in (\mathcal{P}_\omega(\text{Hom}(\Phi, 2)))^A \mid (\lambda x.1) \in t(\tau)\}.$$

To this end, let t be an element of the left-hand side, and suppose towards a contradiction that for all ϕ with $\phi \in t(\tau)$, there is an element $x_\phi \in \Phi$ with $\phi(x_\phi) = 0$. Choosing an assignment $\phi \mapsto x_\phi$ of such elements, we get a *finite* set $\{x_\phi \mid \phi \in t(\tau)\}$. Let x_ϕ be the smallest element of that set (w.r.t. the order of Φ , i.e., the largest natural number), and let $\psi \in \text{Hom}(\Phi, 2)$ be such that $\psi(x_\phi) = 1$; such a ψ exists by assumption on t . However, since $x_\phi \leq x_\psi$ and ψ is upwards closed we have $\psi(x_\phi) = 1$, which gives a contradiction. Hence, the inclusion holds as required. The lifting (\bar{B}, B) is a fibration map. We thus conclude from Theorem 19 that the logic is expressive.

4.3 Simulation of processes

Let $B: \mathbf{Set} \rightarrow \mathbf{Set}$, $BX = (\mathcal{P}_\omega X)^A$, and let $\gamma: X \rightarrow (\mathcal{P}_\omega X)^A$ be B -coalgebra, i.e., a labelled transition system. Denote *similarity* by $\lesssim \subseteq X \times X$, defined more precisely below. Consider

26:14 Expressive Logics for Coinductive Predicates

the logic with the following syntax:

$$\varphi, \psi ::= \langle a \rangle \varphi \mid \varphi \wedge \psi \mid \top \quad (9)$$

where a ranges over A , with the usual interpretation $x \models \varphi$ for states $x \in X$. A classical Hennessy-Milner theorem for similarity is:

$$x \lesssim y \text{ iff } \forall \varphi. x \models \varphi \rightarrow y \models \varphi. \quad (10)$$

We show how to formulate and prove this result within our abstract framework.

First, recall from Equation 2 in Section 2.1 the appropriate lifting $\overline{B}: \text{Rel} \rightarrow \text{Rel}$ in the relation fibration $p: \text{Rel} \rightarrow \text{Set}$. A simulation on a B -coalgebra (X, γ) is a relation R such that $R \leq \gamma^* \circ \overline{B}_X(R)$, and similarity \lesssim is the greatest fixed point of $\gamma^* \circ \overline{B}_X$.

For the logic, to incorporate finite conjunction, we instantiate \mathcal{D} with the category SL of bounded (meet)-semilattices, i.e., sets equipped with an associative, commutative and idempotent binary operator \wedge and a top element \top .

To add the modalities $\langle a \rangle$ for each $a \in A$, we proceed as follows. Let $U: \text{SL} \rightarrow \text{Set}$ be the forgetful functor. It has a left adjoint $\mathcal{F}: \text{Set} \rightarrow \text{SL}$, mapping a set X to the meet-semilattice $\mathcal{P}_\omega(X)$ with the top element given by \emptyset and the meet by union. The functor $L: \text{SL} \rightarrow \text{SL}$ is given by $LX = \mathcal{F}(A \times UX)$; its initial algebra Φ consists precisely of the logic presented in Equation 9, quotiented by the semilattice equations. For the adjunction, we use:

$$B=(\mathcal{P}_\omega-)^A \begin{array}{c} \hookrightarrow \\ \text{Set} \end{array} \begin{array}{c} \xrightarrow{P=2^-} \\ \perp \\ \xleftarrow{Q=\text{Hom}(-,2)} \end{array} \text{SL}^{\text{op}} \begin{array}{c} \hookrightarrow \\ \end{array} L=\mathcal{F}(A \times U-) \quad \text{with} \quad B\text{Hom}(-, 2) \xrightarrow{\delta} \text{Hom}(L-, 2)$$

which is an instance of Equation 4. Here $2 = \{0, 1\}$ is the meet-semilattice given by the order $0 \leq 1$. For a semilattice S , the set $\text{Hom}(S, 2)$ of semi-lattice morphisms is isomorphic to the set of *filters* on S : subsets $X \subseteq S$ such that $\top \in X$, and $x, y \in X$ iff $x \wedge y \in X$.

To define the natural transformation $\delta_S: (\mathcal{P}_\omega(\text{Hom}(S, 2)))^A \rightarrow \text{Hom}(\mathcal{F}(A \times US), 2)$ on a semilattice S , we use that for every map $f: A \times US \rightarrow 2$ there is a unique semilattice homomorphism $f^\sharp: \mathcal{F}(A \times US) \rightarrow 2$:

$$\delta_S(t) = ((a, x) \mapsto \bigvee_{\phi \in t(a)} \phi(x))^\sharp = \left(W \mapsto \bigwedge_{(a,x) \in W} \bigvee_{\phi \in t(a)} \phi(x) \right).$$

For an LTS (X, γ) , the associated theory map $th: X \rightarrow \text{Hom}(\Phi, 2)$ maps a state to the formulas in (9) that it accepts, with the usual semantics.

To recover (10), we need to relate logical theories appropriately. Define

$$\overline{Q}: \text{SL}^{\text{op}} \rightarrow \text{Rel}, \quad \overline{Q}S = (\text{Hom}(S, 2), \{(\phi_1, \phi_2) \mid \forall x \in S. \phi_1(x) \leq \phi_2(x)\}).$$

Then $th^*(\overline{Q}\Phi) = \{(x, y) \mid \forall \varphi \in \Phi. th(x)(\varphi) \leq th(y)(\varphi)\}$, i.e., it relates all (x, y) such that the set of formulas satisfied at x is included in the set of formulas satisfied at y . Thus, instantiating Definition 15, adequacy $\lesssim = \nu(\gamma^* \circ \overline{B}_X) \leq th^*(\overline{Q}\Phi)$ is the implication from left to right in Equation 10, and expressivity is the converse.

We prove adequacy and expressivity. The functor \overline{Q} has a left adjoint, given by $\overline{P}(X, R) = \text{Hom}((X, R), \overline{2})$, where $\overline{2} = (2, \{(x, y) \mid x \leq y\})$. This follows by a straightforward computation, or using Proposition 11 with Remark 13, with SL as a full subcategory of the category of all algebras for the corresponding signature.

Given a semilattice S , we compute $\delta_S^*(\overline{QLS}) \subseteq (BQS)^2 = ((\mathcal{P}_\omega(\text{Hom}(S, 2)))^A)^2$:

$$\begin{aligned} \delta_S^*(\overline{QLS}) &= \delta_S^*({}(\phi_1, \phi_2) \mid \forall W \in \mathcal{F}(A \times US). \phi_1(W) \leq \phi_2(W)) \\ &= \{(t_1, t_2) \mid \forall W \in \mathcal{F}(A \times US). \bigwedge_{(a,x) \in W} \bigvee_{\phi \in t_1(a)} \phi(x) \leq \bigwedge_{(a,x) \in W} \bigvee_{\phi \in t_2(a)} \phi(x)\}. \end{aligned}$$

Further, $\overline{BQS} = \{(t_1, t_2) \mid \forall a \in A. \forall \phi_1 \in t_1(a). \exists \phi_2 \in t_2(a). \forall x \in S. \phi_1(x) \leq \phi_2(x)\}$. For adequacy, we need to prove $\overline{BQS} \leq \delta_S^*(\overline{QLS})$; but this is trivial, given the above computations. For expressivity, let $(t_1, t_2) \in \delta_S^*(\overline{QLS})$. We need to show that $(t_1, t_2) \in \overline{BQS}$. Suppose, towards a contradiction, that $(t_1, t_2) \notin \overline{BQS}$, i.e., there exist $a \in A$ and $\phi_1 \in t_1(a)$ such that for all $\phi_2 \in t_2(a)$, there is $x \in S$ with $\phi_1(x) = 1$ and $\phi_2(x) = 0$. We choose such an element x_{ϕ_2} for every $\phi_2 \in t_2(a)$. Note that the collection $\{x_{\phi_2} \mid \phi_2 \in t_2(a)\}$ is *finite* – here we make use of the image-finiteness captured by the functor B . Now, consider the conjunction $\psi = \bigwedge_{\phi_2 \in t_2(a)} x_{\phi_2} \in S$. Using that ϕ_1 is a homomorphism, we have $\phi_1(\psi) = \phi_1(\bigwedge_{\phi_2 \in t_2(a)} x_{\phi_2}) = \bigwedge_{\phi_2 \in t_2(a)} \phi_1(x_{\phi_2}) = 1$, and consequently $\bigvee_{\phi \in t_1(a)} \phi(\psi) = 1$. We also have $\bigvee_{\phi \in t_2(a)} \phi(\psi) = \bigvee_{\phi_2 \in t_2(a)} \bigwedge_{\phi_2 \in t_2(a)} \phi(x_{\phi_2}) = 0$ since $\phi_2(x_{\phi_2}) = 0$ for every $\phi_2 \in t_2(a)$. Finally, to arrive at a contradiction, let $W = \{(a, \psi)\}$. Since $(t_1, t_2) \in \delta_S^*(\overline{QLS})$ this implies $\bigvee_{\phi \in t_1(a)} \phi(\psi) \leq \bigvee_{\phi \in t_2(a)} \phi(\psi)$, which is in contradiction with the above. It is easy to check that (\overline{B}, B) is a fibration map (cf. [16]). Hence, we conclude expressivity from Theorem 19.

5 Future work

We proposed suitable notions of expressivity and adequacy, connecting coinductive predicates in a fibration to coalgebraic modal logic in a contravariant adjunction. Further, we gave sufficient conditions on the one-step semantics that guarantee expressivity and adequacy, and showed how to put these methods to work in concrete examples.

There are several avenues for future work. First, an intriguing question is whether the characterisation of behavioural metrics in [24, 37] can be covered in the setting of this paper, as well as logics for other distances such as the (abstract, coalgebraic) Wasserstein distance. Those behavioural metrics are already framed in a fibrational setting [4, 34, 2, 23]. While all our examples are for coalgebras in \mathbf{Set} , the fibrational framework allows different base categories, which might be useful to treat, e.g., behavioural metrics for continuous probabilistic systems [35].

A further natural question is whether we can automatically *derive* logics for a given predicate. As mentioned in the introduction, there are various tools to find expressive logics for behavioural equivalence. But extending this to the current general setting is non-trivial. Finally, we note that our expressivity result requires the relevant lifting defining the coinductive predicate to be a fibration map, which in particular implies weak pullback preservation for the canonical relation lifting. This is natural, since the latter captures bisimilarity, while logics capture coalgebraic behavioural equivalence. However, it remains an interesting question whether we can use different liftings to obtain expressivity for behavioural equivalence; perhaps based on the lifting in [21], techniques related to Λ -bisimulations [11, 1, 10] or the lax relation lifting from [28].

References

- 1 Zeinab Bakhtiari and Helle Hvid Hansen. Bisimulation for Weakly Expressive Coalgebraic Modal Logics. In Filippo Bonchi and Barbara König, editors, *7th Conference on Algebra*

- and Coalgebra in Computer Science, *CALCO 2017, June 12-16, 2017, Ljubljana, Slovenia*, volume 72 of *LIPICs*, pages 4:1–4:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPICs.CALCO.2017.4.
- 2 Paolo Baldan, Filippo Bonchi, Henning Kerstan, and Barbara König. Coalgebraic Behavioral Metrics. *Logical Methods in Computer Science*, 14(3), 2018. doi:10.23638/LMCS-14(3:20)2018.
 - 3 Marta Bilková and Matej Dostál. Expressivity of Many-Valued Modal Logics, Coalgebraically. In Jouko A. Väänänen, Åsa Hirvonen, and Ruy J. G. B. de Queiroz, editors, *Logic, Language, Information, and Computation - 23rd International Workshop, WoLLIC 2016, Puebla, Mexico, August 16-19th, 2016. Proceedings*, volume 9803 of *Lecture Notes in Computer Science*, pages 109–124. Springer, 2016. doi:10.1007/978-3-662-52921-8_8.
 - 4 Filippo Bonchi, Barbara König, and Daniela Petrisan. Up-To Techniques for Behavioural Metrics via Fibrations. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, volume 118 of *LIPICs*, pages 17:1–17:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.CONCUR.2018.17.
 - 5 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.
 - 6 Marcello M. Bonsangue and Alexander Kurz. Duality for Logics of Transition Systems. In Vladimiro Sassone, editor, *Foundations of Software Science and Computational Structures, 8th International Conference, FOSSACS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3441 of *LNCS*, pages 455–469. Springer, 2005. doi:10.1007/978-3-540-31982-5_29.
 - 7 Florence Clerc, Nathanaël Fijalkow, Bartek Klin, and Prakash Panangaden. Expressiveness of probabilistic modal logics: A gradual approach. *Inf. Comput.*, 267:145–163, 2019. doi:10.1016/j.ic.2019.04.002.
 - 8 Josee Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for Labelled Markov Processes. *Inf. Comput.*, 179(2):163–193, 2002. doi:10.1006/inco.2001.2962.
 - 9 Ulrich Dorsch, Stefan Milius, and Lutz Schröder. Graded Monads and Graded Logics for the Linear Time - Branching Time Spectrum. In *Proceedings of CONCUR 2019, to appear*, 2019.
 - 10 Sebastian Enqvist. Homomorphisms of Coalgebras from Predicate Liftings. In Reiko Heckel and Stefan Milius, editors, *Algebra and Coalgebra in Computer Science - 5th International Conference, CALCO 2013, Warsaw, Poland, September 3-6, 2013. Proceedings*, volume 8089 of *Lecture Notes in Computer Science*, pages 126–140. Springer, 2013. doi:10.1007/978-3-642-40206-7_11.
 - 11 Daniel Gorín and Lutz Schröder. Simulations and Bisimulations for Coalgebraic Modal Logics. In Reiko Heckel and Stefan Milius, editors, *Algebra and Coalgebra in Computer Science - 5th International Conference, CALCO 2013, Warsaw, Poland, September 3-6, 2013. Proceedings*, volume 8089 of *Lecture Notes in Computer Science*, pages 253–266. Springer, 2013. doi:10.1007/978-3-642-40206-7_19.
 - 12 Ichiro Hasuo, Toshiaki Kataoka, and Kenta Cho. Coinductive predicates and final sequences in a fibration. *Mathematical Structures in Computer Science*, 28(4):562–611, 2018. doi:10.1017/S0960129517000056.
 - 13 Matthew Hennessy and Robin Milner. Algebraic Laws for Nondeterminism and Concurrency. *J. ACM*, 32(1):137–161, 1985. doi:10.1145/2455.2460.
 - 14 Claudio Hermida and Bart Jacobs. Structural Induction and Coinduction in a Fibrational Setting. *Information and Computation*, 145(2):107–152, 1998. doi:10.1006/inco.1998.2725.
 - 15 Horst Herrlich. Topological functors. *General Topology and its Applications*, 4(2):125–142, 1974. doi:10.1016/0016-660X(74)90016-6.
 - 16 Jesse Hughes and Bart Jacobs. Simulations in coalgebra. *Theor. Comput. Sci.*, 327(1-2):71–108, 2004. doi:10.1016/j.tcs.2004.07.022.
 - 17 Bart Jacobs. *Categorical Logic and Type Theory*. Elsevier, 1999.

- 18 Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*, volume 59 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2016. doi:10.1017/CB09781316823187.
- 19 Bart Jacobs and Ana Sokolova. Exemplaric Expressivity of Modal Logics. *Journal of Logic and Computation*, 20(5):1041–1068, 2009.
- 20 Krzysztof Kapulkin, Alexander Kurz, and Jiri Velebil. Expressiveness of Positive Coalgebraic Logic. In Thomas Bolander, Torben Braüner, Silvio Ghilardi, and Lawrence S. Moss, editors, *Advances in Modal Logic 9, papers from the ninth conference on "Advances in Modal Logic," held in Copenhagen, Denmark, 22-25 August 2012*, pages 368–385. College Publications, 2012. URL: <http://www.aiml.net/volumes/volume9/Kapulkin-Kurz-Velebil.pdf>.
- 21 Bartek Klin. The Least Fibred Lifting and the Expressivity of Coalgebraic Modal Logic. In José Luiz Fiadeiro, Neil Harman, Markus Roggenbach, and Jan J. M. M. Rutten, editors, *Algebra and Coalgebra in Computer Science: First International Conference, CALCO 2005, Swansea, UK, September 3-6, 2005, Proceedings*, volume 3629 of *Lecture Notes in Computer Science*, pages 247–262. Springer, 2005. doi:10.1007/11548133_16.
- 22 Bartek Klin. Coalgebraic Modal Logic Beyond Sets. *Electr. Notes Theor. Comput. Sci.*, 173:177–201, 2007. doi:10.1016/j.entcs.2007.02.034.
- 23 Y. Komorida, S. Katsumata, N. Hu, B. Klin, and I. Hasuo. Codensity games for bisimilarity. In *Proceedings of LICS 2019, to appear*, 2019.
- 24 Barbara König and Christina Mika-Michalski. (Metric) Bisimulation Games and Real-Valued Modal Logics for Coalgebras. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, volume 118 of *LIPICs*, pages 37:1–37:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.CONCUR.2018.37.
- 25 Clemens Kupke, Alexander Kurz, and Dirk Pattinson. Algebraic Semantics for Coalgebraic Logics. *Electr. Notes Theor. Comput. Sci.*, 106:219–241, 2004. doi:10.1016/j.entcs.2004.02.037.
- 26 Clemens Kupke and Dirk Pattinson. Coalgebraic semantics of modal logics: An overview. *Theor. Comput. Sci.*, 412(38):5070–5094, 2011. doi:10.1016/j.tcs.2011.04.023.
- 27 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.
- 28 Johannes Marti and Yde Venema. Lax extensions of coalgebra functors and their logic. *Journal of Computer and System Sciences*, 81(5):880–900, 2015. CMCS 2012 (Selected Papers). doi:10.1016/j.jcss.2014.12.006.
- 29 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.
- 30 Dusko Pavlovic, Michael W. Mislove, and James Worrell. Testing Semantics: Connecting Processes and Process Logics. In Michael Johnson and Varmo Vene, editors, *Algebraic Methodology and Software Technology, 11th International Conference, AMAST 2006, Proceedings*, volume 4019 of *LNCS*, pages 308–322. Springer, 2006. doi:10.1007/11784180_24.
- 31 H. E. Porst and W. Tholen. Concrete Dualities. In H. E. Porst and W. Tholen, editors, *Categories at Work*, pages 111–136. Heldermann Verlag, 1991.
- 32 Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.
- 33 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.
- 34 David Sprunger, Shin-ya Katsumata, Jérémy Dubut, and Ichiro Hasuo. Fibrational Bisimulations and Quantitative Reasoning. In Corina Cirstea, editor, *Coalgebraic Methods in Computer Science - 14th IFIP WG 1.3 International Workshop, CMCS 2018, Colocated with ETAPS 2018, Thessaloniki, Greece, April 14-15, 2018, Revised Selected Papers*, volume 11202 of *Lecture Notes in Computer Science*, pages 190–213. Springer, 2018. doi:10.1007/978-3-030-00389-0_11.

- 35 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.
- 36 Rob J. van Glabbeek. The Linear Time-Branching Time Spectrum (Extended Abstract). In Jos C. M. Baeten and Jan Willem Klop, editors, *CONCUR '90, Theories of Concurrency: Unification and Extension, Amsterdam, The Netherlands, August 27-30, 1990, Proceedings*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990. doi:10.1007/BFb0039066.
- 37 Paul Wild, Lutz Schröder, Dirk Pattinson, and Barbara König. A van Benthem Theorem for Fuzzy Modal Logic. 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 909–918. ACM, 2018. doi:10.1145/3209108.3209180.

State Space Reduction For Parity Automata

Christof Löding

RWTH Aachen University, Germany
loeding@informatik.rwth-aachen.de

Andreas Tollkötter 

RWTH Aachen University, Germany
andreas.tollkoetter@rwth-aachen.de

Abstract

Exact minimization of ω -automata is a difficult problem and heuristic algorithms are a subject of current research. We propose several new approaches to reduce the state space of deterministic parity automata. These are based on extracting information from structures within the automaton, such as strongly connected components, coloring of the states, and equivalence classes of given relations, to determine states that can safely be merged. We also establish a framework to generalize the notion of quotient automata and uniformly describe such algorithms. The description of these procedures consists of a theoretical analysis as well as data collected from experiments.

2012 ACM Subject Classification Theory of computation \rightarrow Formal languages and automata theory; Theory of computation \rightarrow Automata over infinite objects; Theory of computation \rightarrow Regular languages

Keywords and phrases automata, ω -automata, parity, minimization, state space reduction, deterministic, simulation relations

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.27

Related Version <https://github.com/atollk/master-thesis/blob/master/tex/thesis.tex>

Supplement Material <https://github.com/atollk/master-thesis>

1 Introduction

Finite automata on ω -words (one sided infinite words) have been introduced in [2] as a formalism for a deciding a logical theory. Since then, such automata have turned out to be a useful tool in verification of finite state-based systems. In particular, nondeterministic Büchi automata (NBA) are a standard tool in model checking for expressing properties of non-terminating systems, see [1]. In some applications, there are algorithms that require the property to be represented by a deterministic automaton, like model checking of probabilistic systems (see, e.g., [1, Section 10.3]), or synthesis of finite state systems from ω -regular specifications (see [20] for an overview of the theory, and [12] for recent developments in practice). Deterministic ω -automata require a more expressive acceptance condition than nondeterministic Büchi automata in order to capture the same language class. One such condition that is widely used because of its compact representation and its good algorithmic properties is the parity condition that dates back to [13] (see the surveys [19, 21] on the theory of ω -automata). In a parity automaton, each state is assigned a priority, which is a natural number. We use here the convention that a run is accepting if the smallest priority that is seen infinitely often is even.



© Christof Löding and Andreas Tollkötter;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 27; pp. 27:1–27:16

Leibniz International Proceedings in Informatics



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

We consider here the problem of finding algorithms for reducing the state space of deterministic parity automata (DPA). Such reduction algorithms can be used as a post processing step that are applied after a determinization construction, and before the DPAs are used in further algorithms.

While deterministic finite automata on finite words can be minimized very efficiently [9] by merging language equivalent states, the problem becomes NP-hard even for deterministic Büchi automata [17], which are the special case of DPAs using only priorities 0 and 1. While language equivalence of states in DPAs can be computed in polynomial time by a simple adaptation of emptiness for Streett automata (see the “fair state problem” in [5]), merging language equivalent states of DPAs does, in general, not preserve the language. Heuristic approaches for reducing the state space of ω -automata, usually based on simulation relations, have up to now mainly focused on NBAs, e.g., [18, 6, 11], or even on alternating Büchi or parity automata [7, 8].

Because of the applications of DPAs in synthesis and probabilistic model checking, we think that it is worth studying the problem of state space reduction also for DPAs. Typically, state space reduction is done by identifying classes of equivalent states that can be merged, and then constructing the quotient automaton in case of a congruence relation, or redirecting all incoming transitions of a class to a representative of that class, and deleting all other states. The most basic merge for DPAs is obtained by interpreting a DPA as a Moore automaton with the priorities as output, which can then be minimized efficiently by merging states that produce the same output for every input sequence [9]. In this context, we call the equivalence relation which considers two states to be equivalent if they are merged by this algorithm the *Moore equivalence*.

While we also take the basic approach of computing states that can be merged, we sometimes need to be careful in the selection of representatives. For that reason, we introduce the notion of “merger templates”, which map sets of states in the original DPA, called the merge set, to other sets of states, called the candidate set. The easiest interpretation, which we refer to as representative merge, allows us to merge all states from the merge set into any single representative that is chosen from the candidate set.

We formulate some basic known reduction techniques for DPAs in this framework. Furthermore, we analyze the known notion of delayed simulation for DPAs. Delayed simulation has been introduced in [6] for nondeterministic Büchi automata. In [8] the notion has been extended to alternating parity automata, but it is shown there that quotienting alternating (and also nondeterministic) parity automata w.r.t. delayed simulation does not preserve the language. For this reason, [8] introduces variants of delayed simulation that can be used for merging. We revisit the definition of delayed simulation and show that for DPAs the corresponding quotient preserves the language.

As our main contribution, we propose three new equivalence relations that can be used for merging states in DPAs, which we call *path refinement*, *threshold Moore*, and *labeled SCC filter (LSF)*. All these techniques require a given equivalence relation \sim over the state space that implies language equivalence of states. This equivalence relation has to be computed separately (it can be the full language equivalence relation, or just a subset of it). In our experiments, we use a relation that is produced as a by-product of the determinization construction.

If \sim is a congruence (like full language equivalence), then path refinement refines one of the congruence classes up to a point such that the remaining blocks can be merged. In the threshold Moore technique, one computes the Moore equivalence of states, considering only the priorities less than or equal to some k , and intersects this with \sim . All states of priority k

that are equivalent in this intersection can be merged. Finally, the LSF merger template removes states also based on the Moore equivalence up to k , but it merges states that are in different SCCs of the DPA after removing all states up to priority k .

We illustrate all these new techniques on small examples, exhibit efficient algorithms for computing the corresponding relations, and provide some experimental data showing that they can achieve significant reductions on DPAs obtained from specifications from the competition SYNTCOMP [10].

The remainder of this paper is structured as follows. In Section 2 we give basic definitions and introduce the notion of merger template. In Section 3, we revisit the notion of delayed simulation. In Sections 4–6 we present our three new approaches. The experimental evaluation is given in Section 7, and in Section 8 we conclude.

2 Automata and Merger Templates

We consider deterministic parity automata (DPA), which are, syntactically, a specific type of Moore automaton. A Moore automaton is of the form $\mathcal{A} = (Q, \Sigma, \delta, f)$ with a finite set Q of states, the input alphabet Σ , a transition function $\delta : Q \times \Sigma \rightarrow Q$, and an output function $f : Q \rightarrow \Gamma$ for some output alphabet Γ . Note that we define the automaton without initial state because we are interested in reducing the number of states of automata by computing equivalence relations on states. In this context, the initial state does not play any role.

We use the standard notations Σ^* for all finite words $w = a_0a_1 \cdots a_n$ with all $a_i \in \Sigma$, and Σ^ω for the set of infinite words $\alpha = a_0a_1a_2 \cdots$ with all $a_i \in \Sigma$. We write $\alpha(i)$ for the i th letter a_i of α . When estimating the complexity of algorithms, we assume that $|\Sigma|$ is a constant.

A run of \mathcal{A} from state $q_0 \in Q$ on an infinite input word $\alpha \in \Sigma^\omega$ is an infinite state sequence $\rho = q_0q_1q_2 \cdots \in Q^\omega$ such that $\delta(q_i, \alpha(i)) = q_{i+1}$ for all i . The generated output of \mathcal{A} on α starting from q_0 is the sequence $f(\rho) = f(q_0)f(q_1) \cdots$ of outputs at the states in the run. Similarly, one defines runs and outputs for finite input words. As usual, we write $\delta^*(q, w)$ for the state that is reached by the run on w that starts in q .

A DPA is a special Moore automaton $\mathcal{A} = (Q, \Sigma, \delta, c)$, where the output function is of the form $c : Q \rightarrow \mathbb{N}$, and is called the priority function. For $q \in Q$, we refer to $c(q)$ as the priority of q , and for $P \subseteq Q$, we let $c(P) = \{c(q) \mid q \in P\}$. A run ρ of a DPA is called accepting if in $c(\rho)$ the smallest priority that occurs infinitely often is even. The word $\alpha \in \Sigma^\omega$ is accepted from $q \in Q$ if the run of \mathcal{A} on α from q is accepting. We write $L(\mathcal{A}, q)$ for the set of all words accepted by \mathcal{A} from q .

In the remainder of the paper, \mathcal{A} with the above components is always a DPA if not noted otherwise.

We consider several types of different relations, mostly over the state domain Q . A relation R is a preorder if it is reflexive and transitive. R is an equivalence relation if it is a symmetric preorder. R is a congruence relation if it is an equivalence relation that is compatible with δ , i.e., if $(p, q) \in R$, then also $(\delta(p, a), \delta(q, a)) \in R$ for all $a \in \Sigma$.

If \sim is an equivalence relation and \mathcal{A} is a DPA, we write $\mathfrak{E}(\sim) \subseteq 2^Q$ for the set of equivalence classes in \mathcal{A} . We define two basic equivalence relations that are used throughout the paper.

► **Definition 1.** *The language equivalence relation is defined by $p \equiv_L q$ iff $L(\mathcal{A}, p) = L(\mathcal{A}, q)$.*

The Moore equivalence relation is defined by $p \equiv_M q$ iff $c(\delta^(p, w)) = c(\delta^*(q, w))$ for all finite words $w \in \Sigma^*$ (that is, for every input word, the sequence of priorities when starting in p is the same as the one when starting in q).*

Both of these relations are actually congruence relations. It is well known that merging language equivalent states does not preserve the accepted language in general. Consider, for example, the DPA from Figure 4 on page 10. All three states are language equivalent, accepting the words with finitely many c and infinitely many a . But it is not possible to merge any of the states as that would change the languages of the remaining states.

In contrast, Moore equivalent states can be merged without changing the language. The main aim of this paper is to identify other conditions under which language equivalent states can be merged. We say that a relation \sim implies language equivalence if $p \sim q$ implies that $p \equiv_L q$.

2.1 Merger Templates

The merge operations that we use are more general than quotient automata. Consider, for example, the DPA in Figure 5 on page 12. As we explain in Section 6, it is possible to remove the states q_1, q_2 , and to redirect the incoming transitions of these states to q_3 or q_4 instead. We say that $M = \{q_1, q_2\}$ is a merge set, and that $C = \{q_3, q_4\}$ is the corresponding candidate set.

We define the notion of a merger template, which maps a collection of such merge sets to their corresponding candidate sets, and the notion of representative merge, which merges the states in the merge sets into a single candidate, respectively.

► **Definition 2.** Let $\mu : D \rightarrow (2^Q \setminus \{\emptyset\})$ be a function for some $D \subseteq 2^Q$. We call μ a merger template if all sets in D are pairwise disjoint and for all sets $M \in D$, $\mu(M) \cap (\bigcup D \setminus M) = \emptyset$. The latter condition means that the candidates $\mu(M)$ for M cannot be inside any other merge set (but they can be inside M).

A representative merge \mathcal{A}' of \mathcal{A} w.r.t. μ is constructed by choosing a representative $r_M \in \mu(M)$ for all $M \in D$ and then removing all states in $M \setminus \{r_M\}$. Transitions that originally lead to one of the removed states are redirected to the representative r_M instead.

The notion of quotient automaton w.r.t. a congruence relation is captured by a representative merge for the merger template that maps each congruence class to itself. We illustrate this on the example of Moore equivalence.

► **Definition 3.** The Moore merger template is defined as $\mu_M : \mathfrak{C}(\equiv_M) \rightarrow 2^Q$ with $\mu_M(\kappa) = \kappa$ for each $\kappa \in \mathfrak{C}(\equiv_M)$.

Then, the following is an easy consequence of the definitions.

► **Proposition 4.** A representative merge of a DPA w.r.t. μ_M is language equivalent to the original and isomorphic to the quotient automaton w.r.t. \equiv_M .

When we apply merge operations, we talk about language equivalence of the resulting automaton to the original one (as in the above proposition). We have defined our automata without initial states, so we need to fix our notion of language equivalence of two automata.

► **Definition 5.** Two DPAs \mathcal{A}_1 and \mathcal{A}_2 are called language equivalent if for each state p in one of the DPAs, there is a state q in the other DPA such that from p and q the same language is accepted (in the respective DPA).

2.2 Schewe Merge

The main focus of this paper lies on techniques to generate merger templates such that a representative merge produces a language equivalent DPA. However, in the remainder of this section, we want to discuss a more involved merge operation than the representative merge. This operation is based on [17], and we therefore call it Schewe merge. We do not use this merge operation in the other sections. The aim is rather to illustrate that representative merges are not the only option.

The Schewe merge works rather similar to the representative merge. In addition to merging states from the merge sets into the chosen representative, it also redirects some transitions to the candidate set. While this does not remove additional states on its own, it simplifies the structure of the automaton to potentially improve the reduction of further reduction algorithms that are applied after the Schewe merge.

► **Definition 6.** *Let μ be a merger template. A Schewe merge of a DPA \mathcal{A} w.r.t. μ is constructed by first building a representative merge. Then, for all merge sets M in μ and all transitions $\delta(p, a) = q$ in the original automaton, if $q \in \mu(M)$ and p is not reachable from q , then the transition is redirected to r_M instead.*

A Schewe merge differs from the representative merge if there is more than one state from the candidate set remaining, and the states are distributed over multiple SCCs. Whenever a transition would move the automaton to a candidate while changing SCC at the same time, that transition is instead redirected to the chosen representative state. One can imagine that, for example, this potentially enhances the reduction of a consecutive Moore merger, as more states now uniformly target the same representative.

It is not obvious if one can simply replace the representative merge with the Schewe merge and still keep the same properties such as preservation of language. We can identify a set of requirements that merger templates have to satisfy to be compatible with the Schewe merge.

► **Definition 7.** *For a representative merge \mathcal{A}' of \mathcal{A} w.r.t. μ , we define the candidate relation \sim_C^μ over the states of \mathcal{A}' by $p \sim_C^\mu q$ if and only if $p = q$ or there is a $C \in \mu(D)$ with $p, q \in C$.*

We call μ Schewe suitable if for all representative merges \mathcal{A}' , \sim_C^μ is a congruence relation, it implies language equivalence, and the reachability order restricted to any equivalence class of \sim_C^μ is symmetric (that is, if one state is reachable from another, they are in the same SCC).

An example for a Schewe suitable merger template is $\mu_{\text{LSF}}^{-1, \equiv L}$ from Section 6. This merger template intuitively expresses that for each class of language equivalent states, one only needs to keep those in a “latest” SCC. The Schewe merge then corresponds to Construction 12 of [17] (where in [17] a notion of “almost equivalence” is used instead of language equivalence because the operation is used in the context of automata on finite words).

► **Theorem 8.** *Let μ be a Schewe suitable merger template and let \mathcal{A} be a DPA. If a representative merge and a Schewe merge of \mathcal{A} are built with the same choices for the representative states, then these two merge DPAs are language equivalent.*

Proof. Let \mathcal{A}' be the representative merge and \mathcal{A}'' be the Schewe merge. Let q_0 be some starting state for the three runs ρ, ρ', ρ'' of the three automata on some word α . We claim that ρ' and ρ'' have the same acceptance status.

Let K be the set of positions where ρ'' uses a transition that does not exist in ρ' . We can observe that for every equivalence class κ of \sim_C^μ , there is at most one k_κ in K . If there would be two such positions k_κ and l_κ , then $\rho''(l_\kappa - 1)$ would be reachable from $\rho''(k_\kappa)$ which contradicts the requirement for the redirection of that edge in the Schewe merge.

As $K = \{k_1, \dots, k_n\}$ is finite, ρ'' eventually only uses transitions that are also present in ρ' . By induction on i , we can show that $\rho'(k_i + 1) \sim_C^\mu \rho''(k_i + 1)$, in particular for $i = n$. As \sim_C^μ implies language equivalence by assumption, that means ρ' and ρ'' must have the same acceptance status. ◀

3 Delayed Simulation

We adapt the notion of delayed simulation, which has been introduced for alternating parity automata in [8], to DPAs. In the special case of DPAs, the computation of delayed simulation becomes simpler, and it can directly be used for state space reduction, while alternating and nondeterministic automata require more restricted variants for this purpose [8].

► **Definition 9** (adapted from [8]). *The delayed simulation equivalence relation is defined as $p \equiv_{de} q$ if and only if the following property holds for all $w \in \Sigma^*$: Let $p' = \delta^*(p, w)$ and $q' = \delta^*(q, w)$. Every run in the automaton that starts in p' or q' eventually sees a priority less than or equal to $\min\{c(p'), c(q')\}$.*

It is easy to see that \equiv_{de} is a congruence relation that implies language equivalence. However, states that are \equiv_{de} -equivalent do in general not have the same priority. In order to correctly merge \equiv_{de} -equivalent states, one has to pick a representative of minimal priority from each class.

► **Definition 10.** *The delayed simulation merge template is $\mu_{de} : \mathfrak{C}(\equiv_{de}) \rightarrow 2^Q$ with $\mu_{de}(\kappa) = \{q \in \kappa \mid c(q) = \min c(\kappa)\}$.*

► **Theorem 11.** *A representative merge of a DPA \mathcal{A} w.r.t. μ_{de} is language equivalent to the original.*

Proof. Consider an input word α , a run $\rho = q_0 q_1 \dots$ of \mathcal{A} on α from some state q_0 , and the corresponding run ρ' starting in the \equiv_{de} -class of q_0 of the DPA \mathcal{A}' obtained by a representative merge. Since the merge picks from each class a representative with smallest priority, it is clear that the priorities of states in ρ' are at each position smaller than or equal to the priorities in ρ . If ρ' visits a state of priority k , then ρ visits a state of priority k now or later, by definition of delayed simulation. Hence, the smallest priority that occurs infinitely often is the same in both runs. ◀

In [8] it is shown that delayed simulation can be computed by solving a Büchi game. Since we consider the special case of deterministic automata, we instead obtain just a deterministic Büchi automaton for which one has to solve language universality in order to compute the delayed simulation equivalence. The automaton is obtained by a product construction for tracking two runs, and a third component that keeps track of the smallest priority that the second state still has to match (see Lemma 13 below).

► **Definition 12.** *Define the deterministic Büchi automaton $\mathcal{G}_{de} = (Q_{de}, \Sigma, \delta_{de}, F_{de})$ as*

- $Q_{de} = Q \times Q \times (c(Q) \cup \{\checkmark\})$
- $\delta_{de}((p, q, k), a) = (p', q', \gamma(c(p'), c(q'), k))$, where $p' = \delta(p, a)$ and $q' = \delta(q, a)$

with obligation function

$$\gamma(i, j, k) = \begin{cases} \checkmark & \text{if } j \leq i \text{ and } j \leq_{\checkmark} k \\ \min_{\leq_{\checkmark}}\{i, k\} & \text{otherwise} \end{cases}$$

where $0 \leq_{\checkmark} 1 \leq_{\checkmark} 2 \leq_{\checkmark} \dots \leq_{\checkmark} \checkmark$.

■ $F_{de} = Q \times Q \times \{\checkmark\}$.

Now using this automaton, we can relate delayed simulation to the question of universal language. A state is *language universal* if starting from it, every input word is accepted.

► **Lemma 13.** *For two states p and q , let $q_{de}^0(p, q) = (p, q, \gamma(c(p), c(q), \checkmark))$. Then $p \equiv_{de} q$ if and only if $q_{de}^0(p, q)$ and $q_{de}^0(q, p)$ are language universal states (all infinite words are accepted from these states in \mathcal{G}_{de}).*

Proof. The run from $q_{de}^0(p, q)$ in \mathcal{G}_{de} consists of the two runs from p and q in the original DPA \mathcal{A} , and the “obligations” in the third component. This obligation is the smallest number k such that the run from p has seen priority k , and the run from q has since then not seen a priority $\leq k$. The obligation is \checkmark if no such number k exists. If the obligation becomes \checkmark infinitely often, then for all priorities k seen in the run from p at some position i , the run from q visits a priority $\leq k$ at position i or later. With this observation, it follows that the condition from the lemma captures the definition of delayed simulation. ◀

► **Theorem 14.** μ_{de} can be computed in $\mathcal{O}(|Q|^2 \cdot |c(Q)|)$.

Proof. Assuming that we can compute \equiv_{de} in a suitable data structure in the described time, building μ_{de} from that is rather trivial. To see how we compute \equiv_{de} , observe that the size of \mathcal{G}_{de} is $\mathcal{O}(|Q|^2 \cdot |c(Q)|)$. The set of language universal states in a DBA can be computed in linear time: we are looking for loops in the subgraph that only consists of the non-accepting states. Then, every state from which such a loop is reachable is not language universal. These operations can all be done in linear time with classic graph algorithms such as depth first search. ◀

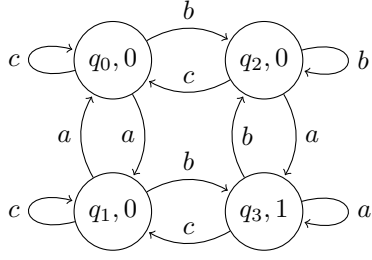
4 Path Refinement

In this section, we present our first new technique, which we call path refinement. It starts from a given congruence relation \sim on the state space that implies language equivalence. For path refinement, we pick one congruence class λ of \sim . We then define an equivalence relation only on the states of λ , and merge the states in the corresponding equivalence classes. For defining the equivalence relation of path refinement, we consider the set $L_{\lambda \leftarrow}$ of non-empty finite words that, starting in a state in λ lead the DPA back to λ without an intermediate visit to λ (note that the precise starting state inside λ does not matter because \sim is a congruence):

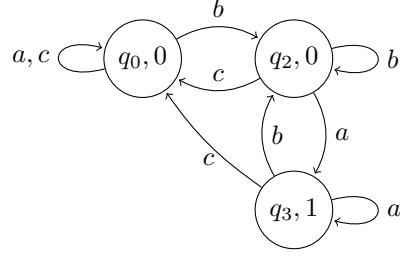
$$L_{\lambda \leftarrow} := \{w = a_1 \cdots a_n \in \Sigma^+ \mid \text{for all (or equivalently some) } q \in \lambda: \delta^*(q, w) \in \lambda, \text{ and } \delta^*(q, a_1 \cdots a_i) \notin \lambda \text{ for all } 1 \leq i < n\}.$$

Based on this language, we define the equivalence relation as follows.

► **Definition 15.** *Let \sim be a congruence relation that implies language equivalence, and let $\lambda \in \mathfrak{C}(\sim)$ be an equivalence class. We define a relation R_λ on λ as $(p, q) \in R_\lambda$ if and only if for all $w \in L_{\lambda \leftarrow}$, the smallest priority seen on the path induced by w is the same starting from p and from q .*



■ **Figure 1** Example automaton.



■ **Figure 2** Example automaton after merging with $\mu_{PR}^{\{q_0, q_1\}}$.

We define path refinement equivalence \equiv_{PR}^λ on λ as the largest subset of R_λ such that $p \equiv_{PR}^\lambda q$ if and only if for all $w \in L_{\lambda \leftrightarrow}$, $\delta^*(p, w) \equiv_{PR}^\lambda \delta^*(q, w)$.

As an example, consider the DPA shown in Figure 1. For the relation \sim , we use exact language equivalence between states. In this case, the language equivalent states are $\{q_0, q_1\}$ and $\{q_2, q_3\}$. They are separated, e.g., by the word a^ω . We choose $\lambda = \{q_0, q_1\}$.

The set $L_{\lambda \leftrightarrow}$ is described by the regular expression $a + c + b(a + b)^*c$; reading any of these words from either q_0 or q_1 will take the DPA back to λ again.

Since q_0, q_1 both have the lowest priority 0, on every path for a word in $L_{\lambda \leftrightarrow}$, the lowest priority that is seen is 0. Hence, $q_0 \equiv_{PR}^\lambda q_1$.

As for delayed simulation, the corresponding merger template defines for each class the states with smallest priority as candidates:

► **Definition 16.** The path refinement merger template is $\mu_{PR}^\lambda : \mathfrak{C}(\equiv_{PR}^\lambda) \rightarrow 2^Q$ with $\mu_{PR}^\lambda(\kappa) = \{q \in \kappa \mid c(q) = \min c(\kappa)\}$.

Going back to the example, the merger template would assign $\mu_{PR}^\lambda(\{q_0, q_1\}) = \{q_0, q_1\}$. The representative merge for the candidate q_0 is shown in Figure 2.

Path refinement thus is able to remove one state from the automaton. In contrast, no two different states are in the delayed simulation equivalence relation.

One can check that this automaton is equivalent to the original DPA. The fact that this is true in general, is captured by the following theorem.

► **Theorem 17.** A representative merge of a DPA \mathcal{A} w.r.t. μ_{PR}^λ is language equivalent to the original.

Proof. Let \mathcal{A}' be the representative merge. Assume there is a starting state $q_0 \in Q'$ and a word α such that the acceptance of the runs ρ (of \mathcal{A} starting in q_0) and ρ' (of \mathcal{A}' starting in q_0) differs. We will bring this assumption to a contradiction.

First, note that at every position i , $\rho(i)$ and $\rho'(i)$ must be \sim -equivalent, as \sim is a congruence relation. If in these runs, λ is visited only finitely often, there is a position j at which it is visited for the last time. Then from j on, ρ' only uses transitions that also exist in the original DPA \mathcal{A} . As $\rho(j) \sim \rho'(j)$, they must be language equivalent and therefore have the same acceptance status. This contradicts the assumption.

Otherwise, λ is visited infinitely often. However, for two consecutive positions k and k' at which λ is seen, we can show that the smallest priorities in $c(\rho(k)), \dots, c(\rho(k'))$ and $c'(\rho'(k)), \dots, c'(\rho'(k'))$ are the same. Then, it easily follows that the entire runs share the same smallest priority that is seen infinitely often.

To observe that the two run segments see the same minimal priority, first observe that $\rho(k) \equiv_{PR}^\lambda \rho'(k)$ by induction on k . If k is the first position at which λ is visited, then $\rho'(k)$

is the representative of the equivalence class of $\rho(k)$ and therefore $\equiv_{\text{PR}}^\lambda$ -equivalent to $\rho(k)$. Then, by definition of the path refinement equivalence, the same holds for k' and therefore all following positions.

Now that we have established $\rho(k) \equiv_{\text{PR}}^\lambda \rho'(k)$, it follows directly from the definition of R_λ that the smallest priorities in $c(\rho(k)), \dots, c(\rho(k'))$ and $c'(\rho'(k)), \dots, c'(\rho'(k'))$ are equal. ◀

We now turn to the question how to compute $\equiv_{\text{PR}}^\lambda$ efficiently. In the naive approach, one can build a product automaton similar to the one for delayed simulation, in which the third component tracks the smallest priority so far and the component it was seen in. Then, at every visit to λ , the tracked values need to coincide. An algorithm based on such a product would have a complexity that is at least quadratic in the state space.

Instead, we build a Moore automaton of size $|Q| \cdot |c(Q)|$ that tracks only for single states the smallest priority seen on paths from λ back to λ . Moore equivalence in this automaton then corresponds to $\equiv_{\text{PR}}^\lambda$.

► **Definition 18.** Define the Moore automaton $\mathcal{A}_{\text{visit}} = (Q_{\text{visit}}^\lambda, \Sigma, \delta_{\text{visit}}^\lambda, f_{\text{visit}}^\lambda)$ by

- $Q_{\text{visit}}^\lambda = Q \times (c(Q) \cup \{\perp\})$
- $\delta_{\text{visit}}^\lambda((q, k), a) = \begin{cases} (q', \min\{c(q), c(q')\}) & \text{if } q \in \lambda \\ (q', \min\{k, c(q')\}) & \text{if } q \notin \lambda \end{cases}$, where $q' = \delta(q, a)$
- $f_{\text{visit}}^\lambda((q, k)) = \begin{cases} k & \text{if } q \in \lambda \\ \perp & \text{if } q \notin \lambda \end{cases}$

► **Lemma 19.** For a state $q \in Q$, let $\iota_q = (q, \max c(Q)) \in Q_{\text{visit}}^\lambda$. Then, for all states p and q , it holds that $p \equiv_{\text{PR}}^\lambda q$ if and only if $\iota_p \equiv_M \iota_q$.

Proof. Our first observation is that for any state $p \in \lambda$, reading some $w \in L_{\lambda \leftarrow}$ from (p, k) ends in (q, k') , where k' is the smallest priority that occurs on the run segment.

If $p \not\equiv_{\text{PR}}^\lambda q$, then there is a $w \in L_{\lambda \leftarrow}$ such that either the smallest priority when reading w from p and q differs, or reading w moves to non-PR-equivalent states. If the former is true, then reading w from ι_p and ι_q brings the visit graph to states with different priorities and therefore $\iota_p \not\equiv_M \iota_q$. If the former is false and the latter is true, then one has to repeatedly apply this argument until at some point a state pair is reached at which the first case is violated. This must happen eventually, as $\equiv_{\text{PR}}^\lambda$ is defined as the *largest* subset satisfying its conditions.

For the other direction, if $\iota_p \not\equiv_M \iota_q$, there must be a $w \in \Sigma^*$ such that the priority differs when reading w from ι_p and ι_q . As all states not in λ have the same output \perp , we can split $w = v_1 \dots v_n$ such that all v_i are words in $L_{\lambda \leftarrow}$. Then, on the last segment, reading v_n sees different minimal priorities from the initial states, and therefore p and q cannot be PR-equivalent. ◀

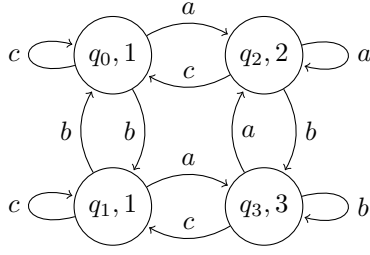
► **Theorem 20.** $\equiv_{\text{PR}}^\lambda$ can be computed in $\mathcal{O}(|Q| \cdot |c(Q)| \cdot \log |Q|)$.

Proof. Moore equivalence for automata with n states can be computed in time $\mathcal{O}(n \log n)$ [9]. The number of states of $\mathcal{A}_{\text{visit}}$ is in $\mathcal{O}(|Q| \cdot |c(Q)|)$. As $|c(Q)|$ is always at most $|Q|$, this gives us the desired complexity. ◀

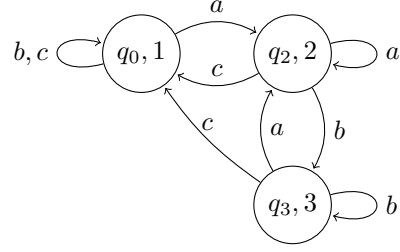
5 Threshold Moore

Similar to Section 4, we again start from an equivalence relation \sim on the state set Q of the DPA that implies language equivalence. In this section, \sim does not have to be a congruence relation. We then intersect \sim with a weakened version of Moore equivalence, and show that states that are equivalent in this intersection can be merged.

27:10 State Space Reduction For Parity Automata



■ **Figure 3** Example automaton.



■ **Figure 4** Example automaton after merging with $\mu_{\widetilde{\text{TM}}}$.

► **Definition 21.** For a priority k , we define the threshold Moore equivalence relation as $p \equiv_M^{\leq k} q$ if and only if for all finite words w , $\delta^*(p, w)$ and $\delta^*(q, w)$ have the same priority or both priorities are greater than k .

Let \sim be an equivalence relation that implies language equivalence. We define the TM equivalence relation as $p \equiv_{\widetilde{\text{TM}}} q$ if and only if $p \sim q$, $c(p) = c(q)$, and $p \equiv_M^{\leq c(p)} q$.

Note that for each k , the relation $\equiv_M^{\leq k}$ is a congruence but, in general, $\equiv_{\widetilde{\text{TM}}}$ is not a congruence, even if \sim is (as can be seen in the example below).

Figure 3 shows a DPA on which we want to illustrate the reduction process. For \sim , we use exact language equivalence again. In this example, all four states are equivalent, as all accept the language $(a + b + c)^*(b^*a)^\omega$.

The threshold Moore relation depends on the choice for parameter k . For $k = 0$, all four states are equivalent because all states have priority greater than 0. For $k = 1$, there are three equivalence classes, $\{q_0, q_1\}$, $\{q_2\}$, and $\{q_3\}$. For $k > 1$, the relation becomes the same as \equiv_M and all states are separated. These observations together imply that $q_0 \equiv_{\widetilde{\text{TM}}} q_1$, and these are the only states that are equivalent w.r.t. $\equiv_{\widetilde{\text{TM}}}$. Therefore, $\equiv_{\widetilde{\text{TM}}}$ is not a congruence because, for example, $\delta(q_0, a) = q_2$ and $\delta(q_1, a) = q_3$.

The merger template for TM relation simply merges classes of $\equiv_{\widetilde{\text{TM}}}$. Note that this is not, however, a quotient automaton, as $\equiv_{\widetilde{\text{TM}}}$ is in general not a congruence relation.

► **Definition 22.** We define the TM merger template $\mu_{\widetilde{\text{TM}}} : \mathfrak{C}(\equiv_{\widetilde{\text{TM}}}) \rightarrow 2^{\mathcal{Q}}$ as $\mu_{\widetilde{\text{TM}}}(\kappa) = \kappa$.

Continuing the example, the representative merge with the candidate q_0 for the class $\{q_0, q_1\}$, results in the automaton shown in Figure 4.

No distinct states are delayed simulation equivalent in this example. Furthermore, for the only \sim -class $\lambda = \{q_0, q_1, q_2, q_3\}$, one can check that no two distinct states are $\equiv_{\text{PR}}^\lambda$ -equivalent.

► **Lemma 23.** Let \mathcal{A} be a DPA and let \mathcal{A}' be a representative merge w.r.t. a single equivalence class $\kappa \in \mathfrak{C}(\mu_{\widetilde{\text{TM}}})$. Then $L(\mathcal{A}, q) = L(\mathcal{A}', q)$ for all states q of \mathcal{A}' . Furthermore, if k is the priority of the states in κ , then for all states p, q of \mathcal{A}' with $k \geq c(p), c(q)$, we have $p \equiv_M^{\leq k} q$ in \mathcal{A} if, and only if, $p \equiv_M^{\leq k} q$ in \mathcal{A}' .

Proof. We focus on the language equivalence first. Let ρ and ρ' be the runs of the two automata on some word α starting in q . We show that these two runs have the same acceptance status.

Note that all states in ρ' are also states of \mathcal{A} . Since \equiv_L is a congruence relation and only language equivalent states are merged, we have that $\rho(i) \equiv_L \rho'(i)$ in \mathcal{A} for all positions i . The same is true for $\equiv_M^{\leq k}$.

If ρ visits infinitely many states of priority at most k , then the two runs see the same smallest priority $l < k$ infinitely often, as $c(\rho(i)) = l$ if and only if $c'(\rho'(i)) = l$. Thus, they must have the same acceptance status.

If $c(\rho)$ only visits finitely many states of priority at most k , then from some point j on in ρ' , only transitions that also exist in \mathcal{A} are taken. As $\rho(j) \equiv_L \rho'(j)$ in \mathcal{A} , we obtain that the two runs have the same acceptance status.

Regarding the second claim of the lemma, let p, q be states with $k \geq c(p), c(q)$ such that p, q are $\equiv_M^{\leq k}$ -equivalent in \mathcal{A} . Let $\alpha \in \Sigma^\omega$, and consider the runs ρ, π of \mathcal{A} on α from p, q , as well as the run ρ', π' of \mathcal{A}' on α starting in p, q .

As $\equiv_M^{\leq k}$ is a congruence relation, $\rho(i) \equiv_M^{\leq k} \rho'(i)$ and $\pi(i) \equiv_M^{\leq k} \pi'(i)$ in \mathcal{A} for all positions i . Furthermore, since p and q are $\equiv_M^{\leq k}$ -equivalent in \mathcal{A} , also $\rho(i) \equiv_M^{\leq k} \pi(i)$ in \mathcal{A} for all i . This implies that $\rho'(i) \equiv_M^{\leq k} \pi'(i)$ in \mathcal{A} for all i .

Therefore, at the positions at which one of ρ' and π' visits a priority $\leq k$, the other run visits the same priority. Hence, p, q are $\equiv_M^{\leq k}$ -equivalent in \mathcal{A}' . ◀

► **Theorem 24.** *A representative merge of a DPA w.r.t. μ_{TM}^{\sim} is language equivalent to the original.*

Proof. Let $\kappa_1, \dots, \kappa_m$ be an enumeration of the equivalence classes in μ_{TM}^{\sim} sorted by descending priority. By Lemma 23, merging the states in κ_i will not change the equivalence classes $\kappa_{i+1}, \dots, \kappa_m$. It is therefore a language preserving operation to merge all equivalence classes in the given order. The resulting automaton is the same as a representative merge w.r.t. μ_{TM}^{\sim} . ◀

The computation of μ_{TM}^{\sim} is rather straightforward.

► **Theorem 25.** *For a given \sim in a suitable data structure, μ_{TM}^{\sim} can be computed in time $\mathcal{O}(|Q| \cdot |c(Q)| \cdot \log |Q|)$.*

Proof. Assuming that \equiv_{TM}^{\sim} is known, computing μ_{TM}^{\sim} is easy. For obtaining \equiv_{TM}^{\sim} , one needs the relations $\equiv_M^{\leq k}$. For each k , this can be computed with just a slight adaption of usual algorithms for Moore equivalence in time $\mathcal{O}(|Q| \cdot \log |Q|)$. This needs to be done for every k , so $|c(Q)|$ times. ◀

6 Labeled SCC Filter

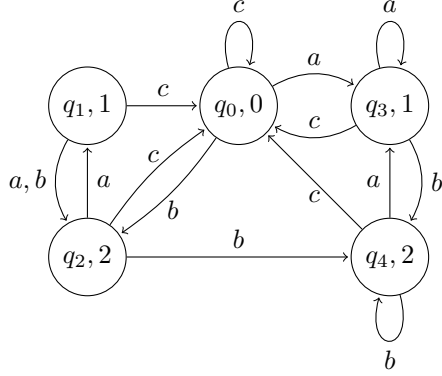
The labeled SCC filter technique (LSF) is also based on the threshold Moore equivalence from Definition 21. While in Section 5 only states of priority k could be merged based on $\equiv_M^{\leq k}$, we now consider states that are $\equiv_M^{\leq k}$ -equivalent, have priority greater than k , and are in different SCCs after removing all states with priority $\leq k$. We then keep from each equivalence class only those states that are in a “deepest” SCC in this restricted automaton, in the sense that no other SCCs are reachable from it.

For that purpose, let $\mathcal{A} \upharpoonright_{>k}^c$ be the restriction of \mathcal{A} to states with priority greater than k . Furthermore, we let \preceq_k be a total preorder on the states in $\mathcal{A} \upharpoonright_{>k}^c$ that extends the reachability preorder. More formally, if q is reachable from p in $\mathcal{A} \upharpoonright_{>k}^c$, then $p \preceq_k q$; on the other hand, if q is not reachable from p , then either $p \prec_k q$ or $q \prec_k p$.

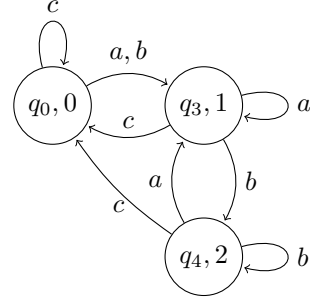
In other words, \preceq_k is a preorder whose equivalence classes are exactly the SCCs in $\mathcal{A} \upharpoonright_{>k}^c$ and which is compatible with a topological sorting of the states.

► **Definition 26.** *Let $k \geq -1$ and let \sim be an equivalence relation that implies language equivalence. We define the LSF equivalence relation $\equiv_{LSF}^{k, \sim}$ such that two states p and q are equivalent if and only if*

27:12 State Space Reduction For Parity Automata



■ **Figure 5** Example automaton.



■ **Figure 6** Example automaton after merging with $\mu_{LSF}^{0,\sim}$.

- $p = q$; or
- $c(p) > k, c(q) > k, p \equiv_M^{\leq k} q$, and $p \sim q$.

Consider the DPA shown in Figure 5. As \sim , we use language equivalence as in the previous examples. All five states are language equivalent. We choose $k = 0$. Since \sim has only one class, $\equiv_{LSF}^{0,\sim}$ is the same as the threshold Moore relation for $k = 0$, which consists of the two equivalence classes $\{q_0\}$ and $\{q_1, q_2, q_3, q_4\}$, separated by the empty word.

The LSF merger template selects for each equivalence class the maximal elements w.r.t. \preceq_k as candidates. Formally, we also need to treat the states with priority $\leq k$, each of which forms its own singleton equivalence class.

► **Definition 27.** For each equivalence class κ of $\equiv_{LSF}^{k,\sim}$: if $\kappa = \{q\}$ for $c(q) \leq k$, then $M_\kappa^k = C_\kappa^k = \{q\}$, and otherwise let

$$C_\kappa^k = \{r \in \kappa \mid p \preceq_k r \text{ for all } p \in \kappa\} \text{ and } M_\kappa^k = \kappa \setminus C_\kappa^k.$$

Define the LSF merger template by $\mu_{LSF}^{k,\sim}(M_\kappa^k) = C_\kappa^k$ for each equivalence class κ of $\equiv_{LSF}^{k,\sim}$.

We continue our example from before with the class $\kappa = \{q_1, q_2, q_3, q_4\}$. Keeping only the states with priority greater than 0, i.e. removing q_0 from the automaton, breaks it into the two SCCs $\{q_1, q_2\}$ and $\{q_3, q_4\}$. There is a transition from q_2 to q_4 , so the relation \preceq_0 is given by $\{q_1, q_2\} \prec_0 \{q_3, q_4\}$.

The merger template therefore assigns $\mu_{LSF}^{0,\sim}(\{q_1, q_2\}) = \{q_3, q_4\}$. Deciding on q_3 as the representative, the resulting automaton after the merge is displayed in Figure 6.

None of the previous reduction algorithms, that is, delayed simulation, path refinement based on \sim , or threshold Moore are able to remove a state from the original automaton.

► **Lemma 28.** Let \mathcal{A} be a DPA and let \mathcal{A}' be a representative merge w.r.t. a single equivalence class $\kappa \in \mathfrak{C}(\mu_{LSF}^{k,\sim})$. Then, $L(\mathcal{A}, q) = L(\mathcal{A}', q)$ for all states q of \mathcal{A}' . Furthermore, for all states p, q of \mathcal{A}' , we have $p \equiv_{LSF}^{k,\sim} q$ in \mathcal{A} if, and only if, $p \equiv_{LSF}^{k,\sim} q$ in \mathcal{A}' .

Proof. Let ρ and ρ' be the runs of the two automata on some word α starting in q . We show that these two runs have the same acceptance status. Since, by definition, $\equiv_{LSF}^{k,\sim} \subseteq \equiv_L$, and $\equiv_{LSF}^{k,\sim} \subseteq \equiv_M^{\leq k}$, we have that $\rho(i) \equiv_L \rho'(i)$ and $\rho(i) \equiv_M^{\leq k} \rho'(i)$ in \mathcal{A} for all positions i (because \equiv_L and $\equiv_M^{\leq k}$ are congruences). The definition of $\equiv_M^{\leq k}$ implies that both runs visit priorities $\leq k$ at the same positions. If infinitely many such priorities $\leq k$ are visited, this implies that both runs have the same acceptance status. If finitely many priorities $\leq k$ are visited,

note that between two transitions in \mathcal{A}' that are not in \mathcal{A} , there has to be a priority $\leq k$ by definition of the merger template. Hence, in this case, ρ' uses only finitely many transitions that are not in \mathcal{A} . Let j be some position such that ρ' after j only uses transitions that also exist in \mathcal{A} . Since $\rho(j) \equiv_L \rho'(j)$, as noted earlier, we obtain that ρ and ρ' have the same acceptance status.

Concerning the second statement, let p, q be states of \mathcal{A}' with $p \equiv_{\text{LSF}}^{k, \sim} q$ in \mathcal{A} . We show that $p \equiv_{\text{LSF}}^{k, \sim} q$ in \mathcal{A}' by proving $p \equiv_M^{\leq k} q$ and $p \equiv_L q$ in \mathcal{A}' . Note that \sim in \mathcal{A}' is just the restriction of \sim to the state set of \mathcal{A}' , so showing $p \equiv_L q$ in \mathcal{A}' also implies that $p \sim q$.

Let α be an infinite word, and let π, π' be the runs of $\mathcal{A}, \mathcal{A}'$ on α starting in p , and ρ, ρ' be the runs of $\mathcal{A}, \mathcal{A}'$ on α starting in q . As explained above, we have $\pi(i) \equiv_M^{\leq k} \pi'(i)$ and $\rho(i) \equiv_M^{\leq k} \rho'(i)$ in \mathcal{A} for all i . Furthermore, $p \equiv_{\text{LSF}}^{k, \sim} q$ in \mathcal{A} implies that also $\pi(i) \equiv_M^{\leq k} \rho(i)$ in \mathcal{A} for all i . By transitivity, we obtain that $\pi'(i) \equiv_M^{\leq k} \rho'(i)$ in \mathcal{A} . In particular, π' and ρ' are at the same time in states of priority $\leq k$. Since α was picked arbitrarily, we conclude that $p \equiv_M^{\leq k} q$ in \mathcal{A}' .

For showing $p \equiv_L q$ in \mathcal{A}' , note that we have shown above (for the first claim of the lemma) that from p the same words are accepted in \mathcal{A} and \mathcal{A}' , and from q the same words are accepted in \mathcal{A} and \mathcal{A}' . Since $p \equiv_L q$ in \mathcal{A} , we can conclude that also $p \equiv_L q$ in \mathcal{A}' . ◀

► **Theorem 29.** *A representative merge of a DPA w.r.t. $\mu_{\text{LSF}}^{k, \sim}$ is language equivalent to the original.*

Proof. Let $\kappa_1, \dots, \kappa_m$ be an enumeration of the equivalence classes of $\equiv_{\text{LSF}}^{k, \sim}$. When merging $M_{\kappa_i}^k$ into $C_{\kappa_i}^k$, the language is preserved and the equivalence classes $\kappa_{i+1}, \dots, \kappa_m$ do not change by Lemma 28. Also the candidate sets $C_{\kappa_j}^k$ themselves do not change, so we can safely merge all $M_{\kappa_i}^k$ into $C_{\kappa_i}^k$. This is the same operation as performed by the merger template. ◀

Computation of the LSF merger consists of computing the threshold Moore equivalence and a reachability analysis in the restricted graph.

► **Theorem 30.** *For a given \sim in a suitable data structure, $\mu_{\text{LSF}}^{k, \sim}$ can be computed in time $\mathcal{O}(|Q| \cdot \log |Q|)$.*

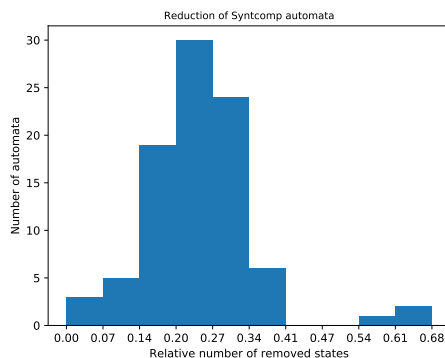
Proof. \sim is already given and $\equiv_M^{\leq k}$ can be computed in $\mathcal{O}(|Q| \cdot \log |Q|)$. Building $\equiv_{\text{LSF}}^{k, \sim}$ is an easy linear time intersection operation.

The second step to building $\mu_{\text{LSF}}^{k, \sim}$ is to compute C_{κ}^k for each κ . For that, it suffices to find the order \preceq_k and then select all the maximal elements from each equivalence class. This order can be computed by a topological sorting on the SCCs of $\mathcal{A} \upharpoonright_{>k}^c$ which one can construct in linear time. ◀

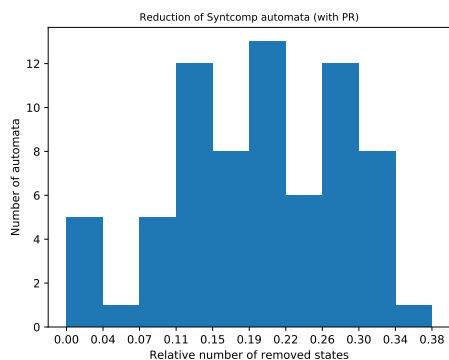
7 Experimental Data

All algorithms that have been presented in Sections 3–6 were also implemented by us in C++ in order to evaluate them on larger examples. The test data set consisted of roughly 100 automata that were constructed from LTL specifications of the Reactive Synthesis Competition (SYNTCOMP) [10].

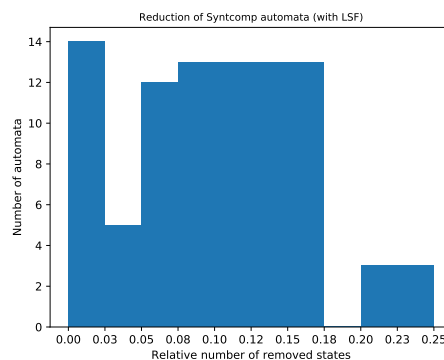
The automata were constructed by translating the given LTL formulas into nondeterministic Büchi automata using the Spot tool ([4]), followed by a conversion to a DPA using nbautils ([14]). During the generation of these DPAs, several techniques for state reduction are already applied, such as minimizing the number of priorities according to [3], and then minimizing the DPA as Moore automata.



■ **Figure 7** Reduction of SYNTCOMP automata. (all techniques)



■ **Figure 8** Reduction of SYNTCOMP automata. (only PR)



■ **Figure 9** Reduction of SYNTCOMP automata. (only LSF)

The sizes of the testing automata range from 9 to 3575 states with a median of 48 and an average of 202; the size of the alphabet Σ ranges from 4 to 2048 symbols with a median of 16. There are three or four different priorities in most of the automata.

The techniques presented in Sections 4–6 require a given equivalence relation \sim that implies language equivalence. Although language equivalence of states in DPAs can be computed in polynomial time, the space and time complexity of the algorithm is in $\mathcal{O}(|Q|^2|c(Q)|^2)$, which turns out to be too high for the larger examples. Instead, we use a relation \sim that can be produced as a side-effect of the determinization construction, as explained in the following.

Determinization constructions for Büchi automata, like the Safra construction [16, 15], are refinements of the standard subset construction for NFAs. The set of states that could have been reached in the NBA is tracked in combination with additional information on visits to accepting states. So there are, in general, many states of the constructed DPA that correspond to the same set S of Büchi states. All these states in the DPA that correspond to the same set S are language equivalent, because from all of them precisely those words are accepted that are accepted by the Büchi automaton from one of the states in S . Therefore, the relation \sim defined for states p, q of the DPA by $p \sim q$ iff p and q correspond to the same set S of Büchi states, is a congruence relation that implies language equivalence. It can therefore be used in the algorithms from Sections 4–6, and is obtained “for free” from the determinization construction.

Figure 7 shows a histogram of overall reduction that was achieved in our experiments. As the generated DPAs can reach sizes of more than 1000, the low complexity of the new techniques was very important. To obtain the histogram, all reduction techniques were applied in succession, with the exception of delayed simulation which proved to be too difficult for the largest automata with its quadratic complexity in both space and time.

The histogram shows a reduction between 14 and 34% of the states in most cases. Individually, the two approaches showing most reduction were path refinement (Section 4) and LSF (Section 6), both of which are analyzed in Figures 8 and 9.

Apart from these tests, we had our reduction also run on DPAs that were determinized from randomly constructed NBAs. The results are rather similar to those shown here and confirm our analysis. In addition to path refinement and LSF, also delayed simulation showed great potential on automata small enough. We consider automata from actual specifications to be of more relevance though, which is why we focus on the SYNTCOMP set here.

8 Conclusion

We have proposed three new ways of reducing the state space of DPAs, and analyzed the known technique of delayed simulation from [6, 8] in the context of DPAs. For obtaining a uniform way of describing the methods, we have introduced the notion of merger template, in order to capture different types of merge operations.

The equivalence relations on which our reduction techniques are based can all be computed very efficiently. Our experiments show that the new methods can further reduce the state space of DPAs that have been obtained by determinizing Büchi automata, and that have already been reduced with known techniques.

We therefore believe that the proposed methods provide interesting tools to be used as post-processing after determinization constructions that produce DPAs. Since other acceptance conditions, like Rabin or Streett automata, are also commonly used in algorithms, a possible topic for future research would be to see if and how our methods can be adapted to these conditions.

References

- 1 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
- 2 J. Richard Büchi. On a decision method in restricted second order arithmetic. In *International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.
- 3 Olivier Carton and Ramón Maceiras. Computing the Rabin index of a parity automaton. *RAIRO-Theoretical Informatics and Applications*, 33(6):495–505, 1999.
- 4 Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω -automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA '16)*, volume 9938 of *Lecture Notes in Computer Science*, pages 122–129. Springer, October 2016. doi:10.1007/978-3-319-46520-3_8.
- 5 E. Allen Emerson and Chin-Laung Lei. Modalities for Model Checking: Branching Time Logic Strikes Back. *Sci. Comput. Program.*, 8(3):275–306, 1987. doi:10.1016/0167-6423(87)90036-0.
- 6 Kousha Etessami, Thomas Wilke, and Rebecca A. Schuller. Fair Simulation Relations, Parity Games, and State Space Reduction for Büchi Automata. In *Automata, Languages and Programming*, pages 694–707, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

- 7 Carsten Fritz and Thomas Wilke. Simulation Relations for Alternating Büchi Automata. *Theor. Comput. Sci.*, 338(1-3):275–314, June 2005. doi:10.1016/j.tcs.2005.01.016.
- 8 Carsten Fritz and Thomas Wilke. Simulation Relations for Alternating Parity Automata and Parity Games. In *Developments in Language Theory, 10th International Conference, DLT 2006, Santa Barbara, CA, USA, June 26-29, 2006, Proceedings*, pages 59–70, 2006. doi:10.1007/11779148_7.
- 9 John E. Hopcroft. An N Log N Algorithm for Minimizing States in a Finite Automaton. Technical report, Stanford University, Stanford, CA, USA, 1971.
- 10 Swen Jacobs, Guillermo A. Pérez, and Roderick Bloem. The Reactive Synthesis Competition. URL: <http://www.syntcomp.org>.
- 11 Richard Mayr and Lorenzo Clemente. Advanced Automata Minimization. In *POPL 2013*, October 2012. URL: <http://arxiv.org/abs/1210.6624>.
- 12 Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit Reactive Synthesis Strikes Back! In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, pages 578–586, 2018. doi:10.1007/978-3-319-96145-3_31.
- 13 A.W. Mostowski. Hierarchies of weak automata and weak monadic formulas. *Theoretical Computer Science*, 83(2):323–335, 1991. doi:10.1016/0304-3975(91)90283-8.
- 14 Anton Pirogov. nbautils. <https://github.com/apirogov/nbautils>, 2018.
- 15 Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Logic in Computer Science, 2006 21st Annual IEEE Symposium on*, pages 255–264. IEEE, 2006.
- 16 Shmuel Safra. On the complexity of omega-automata. In *Foundations of Computer Science, 1988., 29th Annual Symposium on*, pages 319–327, 1988.
- 17 Sven Schewe. Beyond Hyper-Minimisation—Minimising DBAs and DPAs is NP-Complete. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 400–411, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2010.400.
- 18 Fabio Somenzi and Roderick Bloem. Efficient Büchi Automata from LTL Formulae. In *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, pages 248–263, 2000. doi:10.1007/10722167_21.
- 19 Wolfgang Thomas. Handbook of Formal Languages, Vol. 3. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Vol. 3*, chapter Languages, Automata, and Logic, pages 389–455. Springer-Verlag New York, Inc., New York, NY, USA, 1997. URL: <http://dl.acm.org/citation.cfm?id=267871.267878>.
- 20 Wolfgang Thomas. Church’s Problem and a Tour through Automata Theory. In *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, pages 635–655. Springer, 2008. doi:10.1007/978-3-540-78127-1.
- 21 Moshe Y. Vardi and Thomas Wilke. Automata: from logics to algorithms. In *Logic and automata - history and perspectives*, volume 2 of *Texts in Logic and Games*, pages 629–724. Amsterdam University Press, 2007.

Syntactic Interpolation for Tense Logics and Bi-Intuitionistic Logic via Nested Sequents

Tim Lyon 

Institut für Logic and Computation, Technische Universität Wien, Austria
<https://logic-cs.at/phd/students/timothy-lyon/>
lyon@logic.at

Alwen Tiu

Research School of Computer Science, The Australian National University, Australia

Rajeev Goré

Research School of Computer Science, The Australian National University, Australia

Ranald Clouston

Research School of Computer Science, The Australian National University, Australia

Abstract

We provide a direct method for proving Craig interpolation for a range of modal and intuitionistic logics, including those containing a “converse” modality. We demonstrate this method for classical tense logic, its extensions with path axioms, and for bi-intuitionistic logic. These logics do not have straightforward formalisations in the traditional Gentzen-style sequent calculus, but have all been shown to have cut-free nested sequent calculi. The proof of the interpolation theorem uses these calculi and is purely syntactic, without resorting to embeddings, semantic arguments, or interpreted connectives external to the underlying logical language. A novel feature of our proof includes an orthogonality condition for defining duality between interpolants.

2012 ACM Subject Classification Theory of computation → Proof theory; Theory of computation → Automated reasoning

Keywords and phrases Bi-intuitionistic logic, Interpolation, Nested calculi, Proof theory, Sequents, Tense logics

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.28

Related Version A full version of the paper is available at [16], <https://arxiv.org/abs/1910.05215>.

Funding *Tim Lyon*: Supported by the European Union Horizon 2020 Marie Skłodowska-Curie grant No 689176 and FWF projects I 2982, Y544-N2, and W1255-N23.

1 Introduction

The Craig interpolation property for a logic \mathbf{L} states that if $A \Rightarrow B \in \mathbf{L}$, then there exists a formula C in the language of \mathbf{L} such that $A \Rightarrow C \in \mathbf{L}$ and $C \Rightarrow B \in \mathbf{L}$, and every propositional variable appearing in C appears in A and B . This property has many useful applications: it can be used to prove Beth definability [11]; in computer-aided verification it can be used to split a large problem involving $A \Rightarrow B$ into smaller problems involving $A \Rightarrow C$ and $C \Rightarrow B$ [18]; and in knowledge representation (uniform) interpolation can be used to conceal or forget irrelevant or confidential information in ontology querying [15]. Therefore, demonstrating that a logic possesses the Craig interpolation property is of practical value.

Interpolation can be proved semantically or syntactically. In the semantic method, \mathbf{L} is the set of valid formulae, thereby requiring a semantics for \mathbf{L} . In the syntactic method, often known as Maehara’s method [17], \mathbf{L} is the set of theorems, thereby requiring a proof-calculus. The syntactic approach constructs the interpolant C by induction on the (usually cut-free) derivation of $A \Rightarrow B$, and usually also provides derivations witnessing $A \Rightarrow C$ and $C \Rightarrow B$.



© Tim Lyon, Alwen Tiu, Rajeev Goré, and Ranald Clouston;
licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 28; pp. 28:1–28:16

Leibniz International Proceedings in Informatics



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

Over the past forty years, Gentzen’s original sequent calculus has been extended in many different ways to handle a plethora of logics. The four main extensions are hypersequent calculi [2], display calculi [7], nested sequent calculi [1, 10, 21], and labelled calculi [19]. Various interpolation results have been found using these calculi but the only general methodology that we know of is the recent work of Kuznets [13] with Lellman [14]. Although they use extended sequent calculi, binary-relational Kripke semantical arguments are crucial for their methodology, and extending their method to other semantics is left as further work. They also construct the interpolants using a language containing (interpreted) meta-level connectives which are external to the logic at hand, and do not handle logics containing converse modalities such as tense logic. Finally, their method does not yield derivations witnessing $A \Rightarrow C$ and $C \Rightarrow B$.

We give a general, purely syntactic, methodology for proving Craig interpolation using nested sequent calculi for a variety of propositional, non-classical logics including normal tense logics, their extensions with path axioms, and bi-intuitionistic logic. Our methodology does not utilise semantics, does not embed one logic into another, and does not utilise logical connectives which are external to the underlying logical language.

The first novelty of our approach is a generalisation of the notion of interpolant from formulas to sets of sequents. The second is a notion of orthogonality which gives rise to a notion of duality via cut: if two interpolants are orthogonal, then the empty sequent is derivable from the sequents in the interpolants using only the cut and the contraction rules. This duality via cut allows us to relate our more general notion of interpolants (as sets of sequents) to the usual notion of interpolants (as formulas). Moreover, given a derivation of $A \Rightarrow B$, our orthogonality condition not only allows us to construct the interpolant C , but also the derivations witnessing $A \Rightarrow C$ and $C \Rightarrow B$. This fact shows that our approach possesses a distinct complexity-theoretic advantage over the semantic approach: to verify that C is indeed the interpolant of $A \Rightarrow B$, one need only check the derivations of $A \Rightarrow C$ and $C \Rightarrow B$, which is a PTIME process. In the semantic approach, to verify that $A \Rightarrow C$ and $C \Rightarrow B$ are indeed valid (and that C is in fact an interpolant of $A \Rightarrow B$) one must construct proofs of the implications, which is generally much harder (e.g., finding a proof of a validity in one of the tense logics presented in Sec. 3 is PSPACE complete).

Related work. Interpolation has been heavily investigated in the description logic community, where it is used to hide or forget information [23]. In this setting, the logic ALC is a syntactic variant of the multimodal normal modal logic Kn while its extension with inverse roles, ALCI , is a variant of the multimodal normal tense logic Ktn . Cate et al [23] utilise a complexity-optimal tableau algorithm to prove interpolation for ALC via Maehara’s method. They then embed ALCI into ALC and extend their interpolation result to ALCI .

By contrast, our methodology is direct: we obtain interpolation for the normal tense logic Kt , and can then extract interpolation for the normal modal logic K by simply observing that our nested sequent calculus obeys the separation property: if the end-sequent $\vdash A \rightarrow B$ contains no occurrences of the black (converse) modalities, then neither does the interpolant.

As mentioned earlier, the work of Kuznets et al. [6, 13, 14] on interpolation for modal logics in nested sequent calculi is closest to ours. Our construction of interpolants for tense logics shares some similarity with theirs. One crucial difference is that our interpolants are justified purely through syntactic and proof-theoretic means, whereas their interpolants are justified via semantic arguments. Another important difference is that our method extends to the bi-modal case and also (bi-)intuitionistic case, and it is straightforward to adapt our work to the multi-modal case, e.g., using nested sequent calculi as in [24]. Kowalski and Ono [12] showed interpolation for bi-intuitionistic logic using a sequent calculus with analytic cut. In contrast, our proof is based on a cut-free nested sequent calculus [8].

Outline of the paper. In Sec. 2 we give a brief overview of a typical interpolation proof using the traditional sequent calculus, and highlight some issues of extending it to nested sequent calculi, which motivates the generalisation of the interpolation theorem we adopt in this paper. In Sec. 3 we show how the generalised notion of interpolants can be used to prove the Craig interpolation theorem for classical tense logic and its extensions with path axioms [9], covering all logics in the modal cube and more. We then show how our approach can be extended to bi-intuitionistic logic in Sec. 4. In Sec. 5 we conclude and discuss future work. Proofs of the main lemmas and theorems can be found in an extended version of this paper [16].

2 Overview of our approach

We analyze a typical syntactic interpolation proof for Gentzen sequents, highlight the issues of extending it to nested sequents, and motivate our syntactic approach for interpolation.

Consider, for example, a two-sided sequent calculus for classical logic such as **G3c** [25]. Interpolation holds when we can prove that for all $\Gamma_1, \Gamma_2, \Delta_1, \Delta_2$, if $\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2$ is provable in **G3c**, then so are both $\Gamma_1 \vdash \Delta_1, C$ and $C, \Gamma_2 \vdash \Delta_2$, for some C containing only propositional variables common to both Γ_1, Δ_1 and Γ_2, Δ_2 .

The inductive construction of C can be encoded via inference rules over more expressive sequents that specify the splitting of the contexts and the interpolant constructed thus far. In **G3c**, we write $\Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid \Delta_2 \parallel C$ to denote the sequent $\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2$ with its context split into $\Gamma_1 \vdash \Delta_1$ and $\Gamma_2 \vdash \Delta_2$, and with C the interpolant. Inference rules for this extended sequent are similar to the usual ones, with variations encoding the different ways the contexts may be split. For example, the initial rule $\Gamma, p \vdash p, \Delta$ has the following four variants corresponding to the four splittings of where p can occur (with four different interpolants!):

$$\frac{}{\Gamma_1, p \mid \Gamma_2 \vdash p, \Delta_1 \mid \Delta_2 \parallel \perp} \quad \frac{}{\Gamma_1, p \mid \Gamma_2 \vdash \Delta_1 \mid p, \Delta_2 \parallel p}$$

$$\frac{}{\Gamma_1 \mid \Gamma_2, p \vdash p, \Delta_1 \mid \Delta_2 \parallel \neg p} \quad \frac{}{\Gamma_1 \mid \Gamma_2, p \vdash \Delta_1 \mid p, \Delta_2 \parallel \top}$$

Branching rules, such as the right-introduction rule for \wedge , split into two variants, depending on whether the principal formula is in the first or the second partition of the context:

$$\frac{\Gamma_1 \mid \Gamma_2 \vdash A, \Delta_1 \mid \Delta_2 \parallel C \quad \Gamma_1 \mid \Gamma_2 \vdash B, \Delta_1 \mid \Delta_2 \parallel D}{\Gamma_1 \mid \Gamma_2 \vdash A \wedge B, \Delta_1 \mid \Delta_2 \parallel C \vee D} \wedge_{R_1}$$

$$\frac{\Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid \Delta_2, A \parallel C \quad \Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid \Delta_2, B \parallel D}{\Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid \Delta_2, A \wedge B \parallel C \wedge D} \wedge_{R_2}$$

Observe that the interpolants of the conclusion sequents are composed from the interpolants of the premises, but with the main connectives dual to one another: a disjunction in the \wedge_{R_1} rule and a conjunction in the \wedge_{R_2} rule. These observations also apply for the other rules of **G3c**, with a slight subtlety for the implication-left rule: see [25]. Interpolation for **G3c** can then be proved by a straightforward induction on the height of proofs.

Below we discuss some issues with extending this approach to proving interpolation for modal/tense logics and bi-intuitionistic logic using nested sequent calculi, and how these issues lead to the generalisation of the intermediate lemmas we need to prove (which amounts to an interpolation theorem for sequents, rather than formulae).

Classical modal and tense logics

A nested sequent [10] can be seen as a tree of traditional Gentzen-style sequents. For classical modal logics, single-sided sequents suffice, so a nested sequent in this case can be seen as a nested multiset: i.e. a multiset whose elements can be formulae or multisets. Following the notation in [9], a sequent nested inside another sequent is prefixed with a \circ , which is the structural proxy for the \Box modal operator. For example, the nested sequent below first left, with two sub-sequents $\{c, d\}$ and $\{e, f\}$, represents the formula shown second left:

$$\vdash \{a, b, \circ\{c, d\}, \circ\{e, f\}\} \quad a \vee b \vee \Box(c \vee d) \vee \Box(e \vee f) \quad \frac{\vdash \Gamma, \circ\{A, \Delta\}}{\vdash \Gamma, \Diamond A, \circ\{\Delta\}} \quad \frac{\vdash \Diamond \neg p, \circ\{p, q\}}{\vdash \Diamond \neg p, \Diamond p, \circ\{q\}}$$

Nested sequent calculi for modal logics [1, 9, 10] typically contain the *propagation rule* for diamond shown third left above which “propagates” the A into the scope of \circ , when read upwards. Propagation rules complicate the adaptation of the interpolation proof from traditional Gentzen sequent calculi. In particular, it is not sufficient to partition a context into two disjoint multisets. That is, suppose a nested sequent $\vdash \Gamma, \Delta$ is provable, and we would like to construct an interpolant C such that $\vdash \Gamma, C$ and $\vdash \overline{C}, \Delta$ are provable, where \overline{C} is the negation normal form of $\neg C$. Suppose the proof of $\vdash \Gamma, \Delta$ ends with a propagation rule, e.g., when $\Gamma = \Diamond \neg p, \Diamond p$ and $\Delta = \circ\{q\}$ as shown above far right. In this case, by induction, we can construct an interpolant D such that the splittings $\vdash \Diamond \neg p, D$ and $\vdash \overline{D}, \circ\{p, q\}$ of the premiss are provable, but it is in general not obvious how to construct the desired interpolant C for the conclusion $\vdash \Diamond \neg p, \Diamond p, \circ\{q\}$ from D . For this example, D should be $\Box p$, and C should be $\Box \perp$, which does not mention p at all.

The above issue with propagation rules suggests that we need to strengthen the induction hypothesis to construct interpolants, i.e., by considering splitting the sequent context at every sub-sequent in the nested sequent. For example, the nested sequent $\vdash \Diamond \neg p, \circ\{p, q\}$ above should be split into $\vdash \Diamond \neg p, \circ\{p\}$ and $\vdash \circ\{q\}$ when applying the induction hypothesis. Then, $D = C = \Box \perp$ is indeed an interpolant: both $\vdash \Diamond \neg p, \circ\{p\}, \Box \perp$ and $\vdash \circ\top, \circ\{q\}$ are provable. Nevertheless, employing a formula interpolant is not enough to push through the inductive argument in general. Consider, for example, the nested sequent $\vdash \circ\{p, \neg p\}$, which is provable with an identity rule, and its partition $\vdash \circ\{p\}$ and $\vdash \circ\{\neg p\}$. There is no formula C such that both $\vdash \circ\{p\}, C$ and $\vdash \overline{C}, \circ\{\neg p\}$ are provable. One solution to this problem is to generalise the interpolation statement to consider a nested sequent as an interpolant: If a nested sequent $\vdash \Gamma$ is provable, then for every “partitioning” of $\vdash \Gamma$ into $\vdash \Gamma_1$ and $\vdash \Gamma_2$ (where the partitioning applies to every sub-sequent in a nested sequent; the precise definition will be given in subsequent sections), there exists $\vdash \Delta$ (the interpolant), $\vdash \Gamma'_1$ and $\vdash \Gamma'_2$ such that

1. The propositional variables occurring in $\vdash \Delta$ are in both $\vdash \Gamma_1$ and $\vdash \Gamma_2$,
2. $\vdash \Gamma'_1$ splits into $\vdash \Gamma_1$ and $\vdash \Delta$, and $\vdash \Gamma'_2$ splits into $\vdash \Gamma_2$ and $\vdash \overline{\Delta}$, where $\vdash \overline{\Delta}$ denotes the nested sequent $\vdash \Delta$ with all formula occurrences replaced with their negations, and
3. Both $\vdash \Gamma'_1$ and $\vdash \Gamma'_2$ are provable.

For example, the nested sequent $\vdash \circ\{p, \neg p\}$, with partitions $\vdash \circ\{p\}$ and $\vdash \circ\{\neg p\}$, has the interpolant $\vdash \Delta = \vdash \circ\{\neg p\}$ (hence $\vdash \overline{\Delta} = \vdash \circ\{p\}$), and $\vdash \Gamma'_1 = \vdash \Gamma'_2 = \vdash \circ\{p, \neg p\}$.

One remaining issue is that, since we now use a nested sequent as an interpolant, the composition of interpolants needs to be adjusted as well. Recall that in the construction of interpolants for G3c above, in the case involving the right-introduction for \wedge , we constructed either $C \vee D$ or $C \wedge D$ as the interpolant for the conclusion. If C and D are nested sequents, the expression $C \vee D$ or $C \wedge D$ would not be well-formed. To solve this remaining issue, we generalise the interpolant further to be a *set* of nested sequents.

Fitting and Kuznets [6] similarly generalise the notion of interpolants, but instead of generalising interpolants to a set of (nested) sequents, they introduce “meta” connectives for conjunction and disjunction, applicable only to interpolants, and justified semantically. Our notion of interpolants requires no new logical operators or semantical notions.

Propositional bi-intuitionistic logic

Bi-intuitionistic logic is obtained from intuitionistic logic by adding a subtraction (or exclusion) connective \prec that is dual to implication. Its introduction rules are the mirror images of those for implication; in the traditional sequent calculus, these take the form:

$$\frac{A \vdash B, \Delta}{A \prec B \vdash \Delta} \prec_L \quad \frac{\Gamma \vdash \Delta, A \quad \Gamma, B \vdash \Delta}{\Gamma \vdash \Delta, A \prec B} \prec_R$$

However, as shown in [20], the cut rule cannot be entirely eliminated in a sequent calculus featuring these rules, although they can be restricted to analytic cuts [12]. In [8], Postniece et al. show how bi-intuitionistic logic can be formalised in a nested sequent calculus. Although interpolation holds for intuitionistic logic, it does not generalise straightforwardly to bi-intuitionistic logic, and only very recently has interpolation for bi-intuitionistic logic been shown [12]. The proof for the interpolation theorem for intuitionistic logic is very similar to the proof of the same theorem for classical logic; one simply needs to restrict the partitioning of the sequent to the form $\Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid \Delta_2$ where Δ_1 is empty and Δ_2 contains at most one formula occurrence. Since the (nested) sequent calculus for bi-intuitionistic logic uses multiple-conclusion (nested) sequents, the proof for intuitionistic logic cannot be adapted to the bi-intuitionistic case. The problem already shows up in the very simple case involving the identity rule: suppose we have a proof of the initial sequent $p \vdash p$ and we want to partition the sequent as $\cdot \mid p \vdash p \mid \cdot$. It is not possible to find an interpolant C such that $\cdot \vdash p, C$ and $C, p \vdash \cdot$ (otherwise, one would be able to prove the excluded middle $p \vee (p \supset \perp)$, which is not valid in bi-intuitionistic logic, using the cut formula $p \vee C$). In general, the inductive construction of the interpolant for $A \supset B$ may involve finding an interpolant C for the problematic partition of the form $\cdot \mid \Gamma \vdash \Delta \mid \cdot$, where Δ is non-empty. This case does not arise in the interpolation proof for intuitionistic logic in [25], due to the restriction to single-conclusion sequents.

We show that the above issue with bi-intuitionistic logic can be solved using the same approach as in modal logic: simply extend the interpolant to a set of nested sequents. In particular, for $\cdot \mid p \vdash p \mid \cdot$, the generalised interpolation statement only requires finding an interpolating sequent $\Gamma \vdash \Delta$ and its “dual” $\Gamma' \vdash \Delta'$ (see below) such that both $\Gamma \vdash p, \Delta$ and $\Gamma', p \vdash \Delta'$ are provable, which is achieved by letting $\Gamma = \{p\}$, $\Delta = \{ \}$, $\Gamma' = \{ \}$ and $\Delta' = \{p\}$.

Interpolating sequents and orthogonality

In a simplified form (e.g., sequent calculus), the generalised interpolation result we show can be roughly summarised as follows: given a provable sequent $\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2$, there exist two sets of sequents \mathcal{I} and \mathcal{I}' such that

1. For every sequent $(\Sigma \vdash \Theta) \in \mathcal{I}$, the sequent $\Gamma_1, \Sigma \vdash \Delta_1, \Theta$ is provable,
2. For every sequent $(\Sigma' \vdash \Theta') \in \mathcal{I}'$, the sequent $\Gamma_2, \Sigma' \vdash \Delta_2, \Theta'$ is provable,
3. The propositional variables in \mathcal{I} and \mathcal{I}' occur in both $\Gamma_1 \vdash \Delta_1$ and $\Gamma_2 \vdash \Delta_2$, and
4. The sequents in \mathcal{I} and \mathcal{I}' are *orthogonal* to each other, that is, the empty sequent \vdash is derivable from all sequents in $\mathcal{I} \cup \mathcal{I}'$ using only the cut rule and possibly structural rules (contraction and/or weakening).

28:6 Syntactic Interpolation via Nested Sequents

The set \mathcal{I} is taken to be the (sequent) interpolant.

Last, the orthogonality condition, can be seen as a generalisation of duality. To see how this is the case, consider a degenerate case where $\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2$ is a classical sequent (e.g., in G3c). We show how one can convert a formula interpolant in the usual definition (i.e., formula C s.t. $\Gamma_1 \vdash C, \Delta_1$ and $\Gamma_2, C \vdash \Delta_2$ are provable) to a sequent interpolant satisfying the four conditions above, and vice-versa. For the forward direction, simply let $\mathcal{I} = \{ \vdash C \}$ and $\mathcal{I}' = \{ C \vdash \}$. It is easy to see that \mathcal{I} is orthogonal to \mathcal{I}' . For the converse direction, suppose we have a sequent interpolant \mathcal{I} and its orthogonal \mathcal{I}' . We illustrate how one can construct a formula interpolant C . To simplify the discussion, let us assume that

$$\mathcal{I} = \{ (p, q \vdash r, s) \} \quad \mathcal{I}' = \{ (\vdash p), (\vdash q), (r \vdash), (s \vdash) \}$$

and that the following sequents are provable:

$$(1) \Gamma_1, p, q \vdash r, s, \Delta_1 \quad (2) \Gamma_2 \vdash \Delta_2, p \quad (3) \Gamma_2 \vdash \Delta_2, q \quad (4) \Gamma_2, r \vdash \Delta_2 \quad (5) \Gamma_2, s \vdash \Delta_2$$

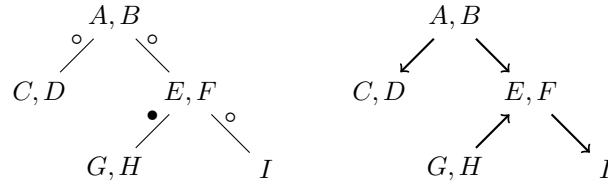
Let $C = (p \wedge q) \supset (r \vee s)$. Then it is easy to see that $\Gamma_1 \vdash \Delta_1, C$ is provable given (1), and $\Gamma_2, C \vdash \Delta_2$ is provable given (2) - (5). The formal statement and the proof of the generalised interpolation theorem will be discussed in detail in the next two sections.

A note on notation

In what follows, we adopt a representation of nested sequents using restricted labelled sequents where we use the labels and relational atoms to encode the tree structure of a nested sequent. To clarify what we mean, consider the following nested sequent for tense logic [9]:

$$A, B, \circ\{C, D\}, \circ\{E, F, \bullet\{G, H\}, \circ\{I\}\}$$

Graphically, the nested sequent can be represented as a tree (shown below left) with two types of edges $\overset{\circ}{\rightarrow}$ and $\overset{\bullet}{\rightarrow}$. Alternatively, the nested sequent can be represented as the polytree shown below right with a single type of edge \rightarrow and where the orientation of the edge encodes the two types of structures $\circ\{\}$ and $\bullet\{\}$ of the nested sequent (observe that the \bullet -edge from E, F to G, H in the left diagram has been reversed in the right diagram).¹



In the latter representation, the structure of the nested sequent can be encoded using a single binary relation: we label each node of the tree corresponding to the nested sequent (as shown above left) with unique labels x, y, z, \dots , encode each edge $x \overset{\circ}{\rightarrow} y$ from a label x to a label y with a relation Rxy , and encode each edge $x \overset{\bullet}{\rightarrow} y$ with a relation Ryx [3]. The above nested sequent can then be equivalently represented as a labelled sequent where $\mathcal{R} = \{Ruv, Ruw, Rvw, Rwy\}$ and R is a relational symbol:

$$\mathcal{R} \vdash u : A, u : B, v : C, v : D, w : E, w : F, x : G, x : H, y : I$$

¹ A polytree is a directed graph such that its underlying graph – the graph obtained by ignoring the orientation of the edges – is a tree.

$$\overline{A} \vee \square \blacklozenge A \quad \overline{A} \vee \blacksquare \blacklozenge A \quad \diamond(A \wedge \overline{B}) \vee \diamond \overline{A} \vee \square B \quad \blacklozenge(A \wedge \overline{B}) \vee \blacklozenge \overline{A} \vee \blacksquare B \quad \frac{A}{\square A} \square \quad \frac{A}{\blacksquare A} \blacksquare$$

■ **Figure 1** The minimal tense logic Kt consists of all classical propositional tautologies, modus ponens, and is additionally extended with the above axioms and inference rules.

Inference rules in a nested sequent calculus can be trivially encoded as rules in a restricted labelled calculus seen as a ‘data structure’ rather than a proper labelled sequent calculus.

We **stress** that our labelled notation to represent nested sequents is just a matter of presentation: the labelled representation is notationally simpler for presenting inference rules and composing nested sequents. For instance, the operation of merging two nested sequents with isomorphic shapes is simply the union of the multiset of labelled formulae.

3 Interpolation for Tense Logics

As usual, we interpret $\square A$ as saying that A holds at every point in the immediate future, and $\diamond A$ as saying that A holds at some point in the immediate future. Conversely, the \blacksquare and \blacklozenge modalities make reference to the past: $\blacksquare A$ says that A holds at every point in the immediate past, and $\blacklozenge A$ says that A holds at some point in the immediate past. Last, we take \overline{p} to be the negation of p , and use the notation $[?] \in \{\square, \blacksquare\}$ and $\langle ? \rangle \in \{\diamond, \blacklozenge\}$.

We consider tense formulae in negation normal form (nnf) as this simplifies our calculi while retaining the expressivity of the original language. The language for the tense logics we consider is given via the following BNF grammar:

$$A ::= p \mid \overline{p} \mid (A \wedge A) \mid (A \vee A) \mid (\square A) \mid (\diamond A) \mid (\blacksquare A) \mid (\blacklozenge A).$$

Since our language excludes an explicit connective for negation, we define it formally below (Def. 1). Using the definition, we may define an implication $A \rightarrow B$ to be $\overline{A} \vee B$.

► **Definition 1.** *For a formula A , we define the negation \overline{A} recursively on the structure of A : if $A = p$ then $\overline{A} := \overline{p}$ and if $A = \overline{p}$ then $\overline{A} := p$. The clauses concerning the connectives are as follows: (1) $\overline{B \wedge C} := \overline{B} \vee \overline{C}$, (2) $[\overline{?}]B := \langle ? \rangle \overline{B}$, (3) $\overline{B \vee C} := \overline{B} \wedge \overline{C}$, and (4) $\langle ? \rangle \overline{B} := [\overline{?}]B$.*

Path axioms are of the form $[\overline{?}]_1[\overline{?}]_2 \cdots [\overline{?}]_n \overline{p} \vee \langle ? \rangle p$ (or, equivalently, $\langle ? \rangle_1 \cdots \langle ? \rangle_n p \rightarrow \langle ? \rangle p$) with $n \in \mathbb{N}$. See [24] for an overview of path axioms.

The tense logics we consider are all extensions of the minimal tense logic Kt (Fig. 1) with path axioms. Thus, $\text{Kt}\Pi$ is the minimal extension of Kt with all axioms from the finite set Π of path axioms.

The calculus for Kt , extended with a set of path axioms Π , is given in Fig. 2. Labelled sequents are defined to be syntactic objects of the form $\mathcal{R} \vdash \Gamma$, where \mathcal{R} is a multiset of relational atoms of the form Rxy and Γ is a multiset of labelled formulae of the form $x : A$, with A a tense formula and labels from a countable set $\{x, y, z, \dots\}$.

Note that the side conditions $x\mathcal{R}^\Pi y$ and $y\mathcal{R}^\Pi x$ of the \diamond and \blacklozenge rules, respectively, depend on the set Π of path axioms added to Kt . The definition of the relation \mathcal{R}^Π is founded upon various auxiliary concepts that fall outside the main scope of this paper. We therefore refer the interested reader to App. A of [16] where the \mathcal{R}^Π relation as well as the concepts needed for its definition are explicitly provided. See also [9, 24] for details.

► **Lemma 2.** *The contraction rules ctr , the weakening rules wk and cut_1 are admissible, and all inference rules are invertible in $\text{Kt}\Pi$.*

can be derived from \mathcal{I}_1 and \mathcal{I}_2 using cut (possibly with contraction). For example, given $\mathcal{I}_1 = \{(\vdash x : A, y : B), (\vdash u : C), (\vdash v : D)\}$, there are several candidates for its dual:

$$\begin{aligned}\mathcal{I}_2 &= \{(\vdash x : \overline{A}), (\vdash y : \overline{B}, u : \overline{C}, v : \overline{D})\} \\ \mathcal{I}_3 &= \{(\vdash x : \overline{A}, u : \overline{C}, v : \overline{D}), (\vdash y : \overline{B})\} \\ \mathcal{I}_4 &= \{(\vdash x : \overline{A}, u : \overline{C}), (\vdash x : \overline{A}, v : \overline{D}), (\vdash y : \overline{B}, u : \overline{C}), (y : \overline{B}, v : \overline{D})\}\end{aligned}$$

The empty sequent can be derived from $\mathcal{I}_1 \cup \mathcal{I}_i$, for $i = 2, 3, 4$ using cut (and contraction, in the case of \mathcal{I}_4). In principle, any of the dual candidates to \mathcal{I}_1 can be used, but to make the construction of the interpolants deterministic, our definition below will always choose \mathcal{I}_4 , as it is relatively straightforward to define as a function of \mathcal{I}_1 .

► **Definition 5.** For an interpolant $\mathcal{I} = \{\vdash \Lambda_1, \dots, \vdash \Lambda_n\}$, the orthogonal $(\mathcal{I})^\perp$ is defined as

$$(\mathcal{I})^\perp = \{(\vdash x_1 : \overline{A_1}, \dots, x_n : \overline{A_n}) \mid \forall i \in \{1, \dots, n\}, x_i : A_i \in \Lambda_i\}.$$

For example, the orthogonal of $\mathcal{I} = \{(\vdash x : A, y : B), (\vdash x : C, z : D)\}$ is

$$(\mathcal{I})^\perp = \{(\vdash x : \overline{A}, x : \overline{C}), (\vdash x : \overline{A}, z : \overline{D}), (\vdash y : \overline{B}, x : \overline{C}), (\vdash y : \overline{B}, z : \overline{D})\}.$$

► **Definition 6.** Let \mathcal{I} be the interpolant

$$\{(\vdash \Delta_1, y : B_{1,1}, \dots, y : B_{1,k_1}), \dots, (\vdash \Delta_n, y : B_{n,1}, \dots, y : B_{n,k_n})\}$$

where y does not occur in $\Delta_1, \dots, \Delta_n$ and define

$$[?]\mathcal{I}_x^y := \{(\vdash \Delta_1, x : [?] \bigvee_{j=1}^{k_1} B_{1,j}), \dots, (\vdash \Delta_n, x : [?] \bigvee_{j=1}^{k_n} B_{n,j})\}$$

where an empty disjunction is \perp .

► **Definition 7.** We define an interpolation sequent to be a syntactic object of the form $\mathcal{R}, \Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid \Delta_2 \parallel \mathcal{I}$, where \mathcal{R} is a set of relational atoms, Γ_i and Δ_i are multisets of labelled formulae (for $i \in \{1, 2\}$), and \mathcal{I} is an interpolant. Note that in the interpolation calculus $\text{Kt}\Pi\text{LI}$, $\Gamma_1 = \Gamma_2 = \emptyset$ (see Fig. 4).

The vertical bar \mid in an interpolation sequent marks where the sequent will be partitioned, with the left partition serving as the antecedent and the right partition serving as the consequent in the interpolation statement. For example, the initial interpolation sequent shown below left splits into the two sequents shown below right

$$\overline{\mathcal{R} \vdash \Gamma \mid x : p, x : \overline{p}, \Delta \parallel \{(\vdash x : \top)\}} \text{ id} \quad (\mathcal{R} \vdash \Gamma, x : \top) \quad (\mathcal{R} \vdash x : \perp, x : p, x : \overline{p}, \Delta)$$

where the first member Γ of the split is placed in the left sequent and the second member $x : p, x : \overline{p}, \Delta$ is placed in the right sequent (note that the relational atoms \mathcal{R} are inherited by both sequents). We think of the interpolant $x : \top$ as being *implied by* the left sequent, and so, we place it in the left sequent, and we think of the interpolant as *implying* the right sequent, so we place its negation (viz. $x : \perp$) in the right sequent. Observe that an application of cut_1 between the two sequents, yields $\mathcal{R} \vdash \Gamma, x : \overline{p}, x : p, \Delta$ without the interpolant. Performing a cut_1 in this way *syntactically establishes* (without evoking the semantics) that the interpolant is indeed an interpolant (so long as the interpolant satisfies certain other properties; cf. Lem. 10 below).

The interpolation calculus $\text{Kt}\Pi\text{LI}$ (Fig. 4) uses interpolation sequents. More importantly, the calculus succinctly represents our algorithm for constructing interpolants. Most of the rules

$$\begin{array}{c}
 \overline{\mathcal{R} \vdash \Gamma, x : \bar{p} \mid x : p, \Delta \parallel \{(\vdash x : p)\}} \text{ id} \quad \overline{\mathcal{R} \vdash \Gamma \mid x : \bar{p}, x : p, \Delta \parallel \{(\vdash x : \top)\}} \text{ id} \quad \frac{\mathcal{R} \vdash \Gamma \mid \Delta \parallel \mathcal{I}}{\mathcal{R} \vdash \Delta \mid \Gamma \parallel (\mathcal{I})^\perp} \text{ orth} \\
 \\
 \frac{\mathcal{R} \vdash \Gamma \mid x : A, x : B, \Delta \parallel \mathcal{I}}{\mathcal{R} \vdash \Gamma \mid x : A \vee B, \Delta \parallel \mathcal{I}} \vee \quad \frac{\mathcal{R} \vdash \Gamma \mid x : A, \Delta \parallel \mathcal{I}_1 \quad \mathcal{R} \vdash \Gamma \mid x : B, \Delta \parallel \mathcal{I}_2}{\mathcal{R} \vdash \Gamma \mid x : A \wedge B, \Delta \parallel \mathcal{I}_1 \cup \mathcal{I}_2} \wedge \\
 \frac{\mathcal{R} \vdash \Gamma \mid x : \diamond A, y : A, \Delta \parallel \mathcal{I}}{\mathcal{R} \vdash \Gamma \mid x : \diamond A, \Delta \parallel \mathcal{I}} \diamond, x\mathcal{R}^\square y \quad \frac{\mathcal{R}, Rxy \vdash \Gamma \mid y : A, \Delta \parallel \mathcal{I}}{\mathcal{R} \vdash \Gamma \mid x : \square A, \Delta \parallel \square \mathcal{I}_x^y} \square, y \text{ fresh} \\
 \frac{\mathcal{R} \vdash \Gamma \mid x : \blacklozenge A, y : A, \Delta \parallel \mathcal{I}}{\mathcal{R} \vdash \Gamma \mid x : \blacklozenge A, \Delta \parallel \mathcal{I}} \blacklozenge, y\mathcal{R}^\square x \quad \frac{\mathcal{R}, Ryx \vdash \Gamma \mid y : A, \Delta \parallel \mathcal{I}}{\mathcal{R} \vdash \Gamma \mid x : \blacksquare A, \Delta \parallel \blacksquare \mathcal{I}_x^y} \blacksquare, y \text{ fresh}
 \end{array}$$

■ **Figure 4** Calculus KtΠLI for constructing interpolants for Kt extended with path axioms Π.

are straightforward counterparts of the proof system in Fig. 2, except for the orthogonality rule *orth*. The orthogonality rule is arguably the most novel aspect of our interpolation calculus, as it imposes a strong requirement on our generalised notion of interpolants, that it must respect the underlying duality in the logic. The key to the correctness of this rule is given in the Persistence Lemma below, which shows that double-orthogonal transformation always retains some sequents in the original interpolant.

► **Lemma 8** (Persistence). *If $\vdash \Lambda \in ((\mathcal{I})^\perp)^\perp$, then there exists a $\vdash \Lambda' \in \mathcal{I}$ such that $\Lambda' \subseteq \Lambda$.*

Proof. Suppose otherwise, i.e., there exists $\vdash \Lambda \in ((\mathcal{I})^\perp)^\perp$ such that for all $\vdash \Lambda' \in \mathcal{I}$, we have $\Lambda' \not\subseteq \Lambda$. Suppose $\mathcal{I} = \{\vdash \Lambda_1, \dots, \vdash \Lambda_n\}$. Then for each i , there must be a labelled formula $x_i : A_i \in \Lambda_i$ such that $x_i : A_i \notin \Lambda$. Let $\Theta = \{x_1 : A_1, \dots, x_n : A_n\}$. By construction, we must have that $\Theta \cap \Lambda = \emptyset$. However, by Def. 5, we have $\bar{\Theta} \in (\mathcal{I})^\perp$, and since $\Lambda \in ((\mathcal{I})^\perp)^\perp$, by Def. 5, $\Theta \cap \Lambda \neq \emptyset$. Contradiction. ◀

Given a formula A , we define the set of propositional variables $\text{var}(A)$ of A to be the set $\{p \mid p \text{ or } \bar{p} \text{ in } A\}$. This notation extends straightforwardly to sets of formulae and interpolants.

We write $\mathcal{R} \Vdash \Gamma, \Delta$ to denote that the sequent $\mathcal{R} \vdash \Gamma, \Delta$ is provable in KtΠL. Similarly, $\mathcal{R} \Vdash \Gamma \mid \Delta \parallel \mathcal{I}$ denotes that the sequent $\mathcal{R} \vdash \Gamma \mid \Delta \parallel \mathcal{I}$ is provable in KtΠLI.

► **Definition 9.** *A logic has the Craig interpolation property iff for every implication $A \Rightarrow B$ in the logic, there is a formula C such that (i) $\text{var}(C) \subseteq \text{var}(A) \cap \text{var}(B)$ and (ii) $A \Rightarrow C$ and $C \Rightarrow B$ are in the logic, where \Rightarrow is taken to be the implication connective of the logic.*

We now establish that each tense logic KtΠ possess the Craig interpolation property when the implication connective is taken to be \rightarrow . To achieve this, we begin by showing that an interpolant sequent can be constructed from any cut-free proof.

► **Lemma 10.** *If $\mathcal{R} \Vdash \Gamma, \Delta$, then there exists an \mathcal{I} such that $\mathcal{R} \Vdash \Gamma \mid \Delta \parallel \mathcal{I}$, $\text{var}(\mathcal{I}) \subseteq \text{var}(\Gamma) \cap \text{var}(\Delta)$, and all labels occurring in \mathcal{I} also occur in \mathcal{R}, Γ or Δ .*

Proof. Induction on the height of the proof of $\mathcal{R} \vdash \Gamma, \Delta$ and by using the rules of KtΠLI. ◀

The next lemma establishes the correctness of the interpolants constructed from our interpolation calculus in Fig. 4. Its proof can be found in [16].

► **Lemma 11.** *For all $\mathcal{R}, \Gamma, \Delta$ and \mathcal{I} , if $\mathcal{R} \Vdash \Gamma \mid \Delta \parallel \mathcal{I}$, then*

1. *For all $(\vdash \Lambda) \in \mathcal{I}$, we have $\mathcal{R} \Vdash \Gamma, \Lambda$ and*
2. *For all $(\vdash \Theta) \in (\mathcal{I})^\perp$, we have $\mathcal{R} \Vdash \Theta, \Delta$.*

To prove Craig interpolation, we need to construct formula interpolants. Lem. 11 provides sequent interpolants, so the next step is to show how one can derive a formula interpolant from a sequent interpolant. This is possible if the formulas in an interpolant are all prefixed with the same label. In that case, there is a straightforward interpretation of the interpolant as a formula. More precisely, let $\mathcal{I} = \{(\vdash \Lambda_1), \dots, (\vdash \Lambda_n)\}$, where $\Lambda_i = \{x : A_{i,1}, \dots, x : A_{i,k_i}\}$ for all $1 \leq i \leq n$. Then, its formula interpretation is given by $\bigwedge_{i=1}^n \bigvee_{j=1}^{k_i} A_{i,j}$. Given such an interpolant \mathcal{I} , we write $\bigwedge \bigvee \mathcal{I}$ to denote its formula interpretation. The following lemma is a straightforward consequence of this interpretation.

► **Lemma 12.** *Let $\mathcal{I} = \{\vdash \Lambda_1, \dots, \vdash \Lambda_n\}$ be an interpolant with $\Lambda_i = \{x : A_{i,1}, \dots, x : A_{i,k_i}\}$ for each $1 \leq i \leq n$. For any multiset of relational atoms \mathcal{R} and multiset of labelled formulae Γ , if $\mathcal{R} \Vdash \Gamma, \Lambda$ for all $\vdash \Lambda \in \mathcal{I}$, then $\mathcal{R} \Vdash \Gamma, x : \bigwedge \bigvee \mathcal{I}$.*

However, the formula-interpolant derived in Lem. 12 gives only one-half of the full picture, as one still needs to show that the orthogonal of a sequent interpolant admits a dual interpretation as a formula. A key to this is the following Duality Lemma that shows that orthogonality behaves like negation.

► **Lemma 13 (Duality).** *Given an interpolant \mathcal{I} , the empty sequent is derivable from $\mathcal{I} \cup (\mathcal{I})^\perp$ using the cut_1 rule and the contraction rule.*

An interesting consequence of Duality Lemma is that it translates into duality in the above formula interpretation as well, as made precise in the following lemma.

► **Lemma 14.** *Let $\mathcal{I} = \{\vdash \Lambda_1, \dots, \vdash \Lambda_n\}$ be an interpolant with $\Lambda_i = \{x : A_{i,1}, \dots, x : A_{i,k_i}\}$ for each $1 \leq i \leq n$. For any multiset of relational atoms \mathcal{R} and multiset of labelled formulae Δ , if $\mathcal{R} \Vdash \Theta, \Delta$ for all $\vdash \Theta \in (\mathcal{I})^\perp$, then $\mathcal{R} \Vdash x : \overline{\bigwedge \bigvee \mathcal{I}}, \Delta$.*

Proof. Suppose $(\mathcal{I})^\perp = \{(\vdash \Theta_1), \dots, (\vdash \Theta_k)\}$ for some k . By Lem. 13, we have a derivation Ξ_1 of the empty sequent from assumptions $\mathcal{I} \cup (\mathcal{I})^\perp$.

$$\begin{array}{ccccccc} \vdash \Lambda_1 & \cdots & \vdash \Lambda_n & & \vdash \Theta_1 & \cdots & \vdash \Theta_k \\ & & & & \vdots & & \\ & & & & \vdash & & \end{array}$$

Due to admissibility of weakening (Lem. 2), for each Λ_i , there is a proof Ψ_i of the sequent $\mathcal{R} \vdash \Lambda_i, x : \overline{\bigwedge \bigvee \mathcal{I}}$. Adding $x : \overline{\bigwedge \bigvee \mathcal{I}}$ to every leaf sequent in Ξ_1 belonging to \mathcal{I} gives us a derivation Ξ_2 :

$$\begin{array}{ccccccc} [\mathcal{R} \vdash x : F, \Lambda_1] & \cdots & [\mathcal{R} \vdash x : F, \Lambda_n] & & \mathcal{R} \vdash \Theta_1 & \cdots & \mathcal{R} \vdash \Theta_k \\ & & & & \vdots & & \\ & & & & \overline{\mathcal{R} \vdash (x : \overline{\bigwedge \bigvee \mathcal{I}})^*} & & \\ & & & & \mathcal{R} \vdash x : \overline{\bigwedge \bigvee \mathcal{I}} & & \text{ctr}^* \end{array}$$

where $F = \overline{\bigwedge \bigvee \mathcal{I}}$ and sequents in brackets are provable, and where $*$ denotes multiple copies of sequents or rules. By the assumption we know that each $\mathcal{R} \vdash \Theta_i, \Delta$ is provable, so by adding Δ to each premise sequent in Ξ_2 , we get the following proof:

$$\begin{array}{ccccccc} [\mathcal{R} \vdash x : F, \Lambda_1, \Delta] & \cdots & [\mathcal{R} \vdash x : F, \Lambda_n, \Delta] & & [\mathcal{R} \vdash \Theta_1, \Delta] & \cdots & [\mathcal{R} \vdash \Theta_k, \Delta] \\ & & & & \vdots & & \\ & & & & \overline{\mathcal{R} \vdash (x : \overline{\bigwedge \bigvee \mathcal{I}})^*, \Delta^*} & & \\ & & & & \mathcal{R} \vdash x : \overline{\bigwedge \bigvee \mathcal{I}}, \Delta & & \text{ctr}^* \end{array} \quad \blacktriangleleft$$

► **Theorem 15.** *If $\Vdash x : A \rightarrow B$, then there exists a C such that (i) $\text{var}(C) \subseteq \text{var}(A) \cap \text{var}(B)$ and (ii) $\Vdash x : A \rightarrow C$ and $\Vdash x : C \rightarrow B$.*

$$\begin{array}{c}
\frac{Rxy, Ryz, Rzw \vdash x : \diamond \Box q, w : q, z : q \mid y : \diamond \bar{p}, w : \bar{p}, y : \diamond \diamond p, z : \diamond p, w : p \parallel \{(\vdash w : \top)\}}{Rxy, Ryz, Rzw \vdash x : \diamond \Box q, w : q, z : q \mid y : \diamond \bar{p}, w : \bar{p}, y : \diamond \diamond p, z : \diamond p \parallel \{(\vdash w : \top)\}} \quad id \\
\frac{Rxy, Ryz, Rzw \vdash x : \diamond \Box q, w : q, z : q \mid y : \diamond \bar{p}, w : \bar{p}, y : \diamond \diamond p \parallel \{(\vdash w : \top)\}}{Rxy, Ryz, Rzw \vdash x : \diamond \Box q, w : q, z : q \mid y : \diamond \bar{p}, y : \diamond \diamond p \parallel \{(\vdash w : \top)\}} \quad \diamond \\
\frac{Rxy, Ryz, Rzw \vdash x : \diamond \Box q, w : q, z : q \mid y : \diamond \bar{p}, y : \diamond \diamond p \parallel \{(\vdash w : \top)\}}{Rxy, Ryz, Rzw \vdash y : \diamond \bar{p}, y : \diamond \diamond p \mid x : \diamond \Box q, w : q, z : q \parallel \{(\vdash w : \perp)\}} \quad orth \\
\frac{Rxy, Ryz \vdash y : \diamond \bar{p}, y : \diamond \diamond p \mid x : \diamond \Box q, z : \Box q, z : q \parallel \{(\vdash z : \Box \perp)\}}{Rxy, Ryz \vdash y : \diamond \bar{p}, y : \diamond \diamond p \mid x : \diamond \Box q, y : \Box q \parallel \{(\vdash y : \Box \Box \perp)\}} \quad \Box \\
\frac{Rxy \vdash y : \diamond \bar{p}, y : \diamond \diamond p \mid x : \diamond \Box q, y : \Box q \parallel \{(\vdash y : \Box \Box \perp)\}}{Rxy \vdash y : \diamond \bar{p}, y : \diamond \diamond p \mid x : \diamond \Box q \parallel \{(\vdash y : \Box \Box \perp)\}} \quad \diamond \\
\frac{Rxy \vdash x : \diamond \Box q \mid y : \diamond \bar{p}, y : \diamond \diamond p \parallel \{(\vdash y : \diamond \diamond \top)\}}{Rxy \vdash x : \diamond \Box q \mid y : \diamond \bar{p} \vee \diamond \diamond p \parallel \{(\vdash y : \diamond \diamond \top)\}} \quad orth \\
\frac{Rxy \vdash x : \diamond \Box q \mid y : \diamond \bar{p} \vee \diamond \diamond p \parallel \{(\vdash y : \diamond \diamond \top)\}}{\vdash x : \diamond \Box q \mid x : \Box (\diamond \bar{p} \vee \diamond \diamond p) \parallel \{(\vdash x : \Box \diamond \diamond \top)\}} \quad \vee
\end{array}$$

■ **Figure 5** An example of the construction of tense interpolants.

$$\begin{array}{c}
\frac{\mathcal{R}, \Gamma, \Gamma', \Gamma' \vdash \Delta}{\mathcal{R}, \Gamma, \Gamma' \vdash \Delta} \quad ctr \quad \frac{\mathcal{R}, \Gamma \vdash \Delta, \Delta', \Delta'}{\mathcal{R}, \Gamma \vdash \Delta, \Delta'} \quad ctr \quad \frac{\mathcal{R}, \mathcal{R}', \mathcal{R}', \Gamma \vdash \Delta}{\mathcal{R}, \mathcal{R}', \Gamma \vdash \Delta} \quad ctr \quad \frac{\mathcal{R}, \Gamma \vdash \Delta}{\mathcal{R}, \Gamma \vdash \Delta, \Delta'} \quad wk \quad \frac{\mathcal{R}, \Gamma \vdash \Delta}{\mathcal{R}, \Gamma, \Gamma' \vdash \Delta} \quad wk \\
\frac{\mathcal{R}, \Gamma \vdash \Delta}{\mathcal{R}, \mathcal{R}', \Gamma \vdash \Delta} \quad wk \quad \frac{\mathcal{R} \vdash \Gamma, x : A \quad \mathcal{R} \vdash \Gamma, x : \bar{A}}{\mathcal{R} \vdash \Gamma} \quad cut_1 \quad \frac{\mathcal{R}, \Gamma \vdash \Delta, x : A \quad \mathcal{R}, x : A, \Gamma \vdash \Delta}{\mathcal{R}, \Gamma \vdash \Delta} \quad cut_2
\end{array}$$

■ **Figure 6** Admissible rules.

► **Corollary 16.** *Every extension of the (minimal) tense logic \mathbf{Kt} with a set Π of path axioms has the Craig interpolation property.*

► **Example 17.** Consider the formula given in example 3. By making use of its derivation in Fig. 3, we can apply our interpolation algorithm as shown in Fig. 5 to construct an interpolant for the formula.

4 Interpolation for Bi-Intuitionistic Logic

The language for bi-intuitionistic logic \mathbf{Bilnt} is given via the following BNF grammar:

$$A ::= p \mid \top \mid \perp \mid (A \wedge A) \mid (A \vee A) \mid (A \supset A) \mid (A \prec A)$$

For an axiomatic definition of \mathbf{Bilnt} consult [22] and for a semantic definition see [8, 20].

The calculus \mathbf{BilntL} for \mathbf{Bilnt} is given in Fig. 7. The calculus makes use of sequents of the form $\mathcal{R}, \Gamma \vdash \Delta$ with \mathcal{R} a multiset of relational atoms of the form Rxy , Γ and Δ multisets of labelled formulae of the form $x : A$ (where A is a bi-intuitionistic formula), and all labels are among a countable set $\{x, y, z, \dots\}$. Note that we need not restrict the consequent of sequents to at most one formula on the right or left due to the eigenvariable condition imposed on the \supset_R and \prec_L rules. Moreover, for a multiset \mathcal{R} of relational atoms or a multiset Γ of labelled formulae, we use the notation $\mathcal{R}[x/y]$ and $\Gamma[x/y]$ to represent the multiset obtained by replacing each occurrence of the label y for the label x . The *monl* and *monr* rules are the natural way to capture monotonicity when nested sequents are represented using labels.

► **Lemma 18.** *The calculus \mathbf{BilntL} enjoys the following: (1) admissibility of *ctr*, *wk* and *cut₂* from Fig. 6; (2) invertibility of all inference rules from Fig. 7; (3) if $\mathcal{R}, Rxy, \Gamma \Vdash \Delta$, then $\mathcal{R}[x/y], \Gamma[x/y] \Vdash \Delta[x/y]$, and (4) If $\Gamma \Vdash \Delta$ where Γ and Δ only contain formulae solely labelled with y , then $\Gamma[x/y] \Vdash \Delta[x/y]$ for any label x .*

$$\begin{array}{c}
\frac{\overline{\mathcal{R}, \Gamma, x : p, \Gamma \vdash \Delta, x : p} \text{ id}}{\mathcal{R}, \Gamma, x : A \vdash \Delta} \quad \frac{\overline{\mathcal{R}, \Gamma \vdash x : \top, \Delta} \top}{\mathcal{R}, \Gamma \vdash x : A \vee B, \Delta} \quad \frac{\overline{\mathcal{R}, \Gamma, x : \perp \vdash \Delta} \perp}{\mathcal{R}, Rxy, x : A, y : A, \Gamma \vdash \Delta} \\
\frac{\mathcal{R}, \Gamma, x : A \vee B \vdash \Delta}{\mathcal{R}, \Gamma, x : A \vee B, \Delta} \vee_L \quad \frac{\mathcal{R}, \Gamma \vdash x : A, x : B, \Delta}{\mathcal{R}, \Gamma \vdash x : A \vee B, \Delta} \vee_R \quad \frac{\mathcal{R}, Rxy, x : A, y : A, \Gamma \vdash \Delta}{\mathcal{R}, Rxy, x : A, \Gamma \vdash \Delta} \text{ monl} \\
\frac{\mathcal{R}, \Gamma \vdash x : A, \Delta \quad \mathcal{R}, \Gamma \vdash x : B, \Delta}{\mathcal{R}, \Gamma \vdash x : A \wedge B, \Delta} \wedge_R \quad \frac{\mathcal{R}, \Gamma, x : A, x : B \vdash \Delta}{\mathcal{R}, \Gamma, x : A \wedge B \vdash \Delta} \wedge_L \quad \frac{\mathcal{R}, Rxy, \Gamma \vdash x : A, y : A, \Delta}{\mathcal{R}, Rxy, \Gamma \vdash y : A, \Delta} \text{ monr} \\
\frac{\mathcal{R}, Rxy, y : A, \Gamma \vdash y : B, \Delta}{\mathcal{R}, x : A < B, \Gamma \vdash \Delta} <_L, y \text{ fresh} \quad \frac{\mathcal{R}, x : A \supset B, \Gamma \vdash x : A, \Delta \quad \mathcal{R}, x : B, \Gamma \vdash \Delta}{\mathcal{R}, x : A \supset B, \Gamma \vdash \Delta} \supset_L \\
\frac{\mathcal{R}, \Gamma \vdash x : A, \Delta \quad \mathcal{R}, x : B, \Gamma \vdash x : A < B, \Delta}{\mathcal{R}, \Gamma \vdash x : A < B, \Delta} <_R \quad \frac{\mathcal{R}, Rxy, \Gamma, y : A \vdash y : B, \Delta}{\mathcal{R}, \Gamma \vdash x : A \supset B, \Delta} \supset_R, y \text{ fresh}
\end{array}$$

■ **Figure 7** The calculus BilntL for Bilnt [20].

Proof. For proofs of (1)-(3), see [20, Section 3]. Statement (4) follows from the others. ◀

As in the case with tense logics, we define a generalised interpolant to be a set of two-sided flat sequents. However, to ease the definition of orthogonal, we shall use an encoding of two-sided sequents into single-sided sequents by annotating the left-hand side occurrence of a formula with an L and the right-hand side occurrence with an R . In this way some results concerning intuitionistic interpolants can be easily adapted from the classical counterparts.

► **Definition 19.** A polarised formula is a formula annotated with L (left-polarised) or R (right-polarised). We write A^L (A^R) for the left-polarised (right-polarised) version of formula A . A labelled polarised formula is a polarised formula further annotated with a label. We write $x : A^L$ ($x : A^R$) to denote a left-polarised (a right-polarised) formula labelled with x . Given a polarised formula A^L (resp. A^R), its dual is defined as $\overline{A^L} = A^R$ and $\overline{A^R} = A^L$. That is, duality changes polarities (the side where the formula occurs), but not the actual formula.

► **Definition 20.** A polarised (flat) sequent is a single-sided (flat) sequent where all formulas in the sequent are polarised. Given a two-sided sequent $S = \mathcal{R}, x_1 : A_1, \dots, x_m : A_m \vdash y_1 : B_1, \dots, y_n : B_n$, its corresponding polarised sequent is the following

$$\mathcal{R} \vdash x_1 : A_1^L, \dots, x_m : A_m^L, y_1 : B_1^R, \dots, y_n : B_n^R.$$

Given a two-sided sequent S , we denote with $\mathsf{P}(S)$ its encoding as a polarised sequent. Conversely, given a polarised sequent S , we denote with $\mathsf{T}(S)$ its two-sided counterpart. This notation extends to sets of sequents by applying the encoding element-wise.

► **Definition 21.** An intuitionistic interpolant is a set of two-sided flat sequents. Given an intuitionistic interpolant \mathcal{I} , its orthogonal $(\mathcal{I})^\perp$ is defined as $\mathsf{T}((\mathsf{P}(\mathcal{I}))^\perp)$.

► **Example 22.** Let $\mathcal{I} = \{(x : A \vdash y : B), (\vdash u : C, v : D)\}$. Then, $(\mathcal{I})^\perp$ is the set:

$$\{(x : A \vdash u : C), (x : A \vdash v : D), (\vdash u : C, y : B), (\vdash v : D, y : B)\}$$

By defining orthogonality via the embedding into polarised sequents, the Persistence Lemma for the intuitionistic case comes for free, by appealing to Lem. 8. Note that for sequents $\Gamma_1 \vdash \Delta_1$ and $\Gamma_2 \vdash \Delta_2$, we write $\Gamma_1 \vdash \Delta_1 \subseteq \Gamma_2 \vdash \Delta_2$ iff $\Gamma_1 \subseteq \Gamma_2$ and $\Delta_1 \subseteq \Delta_2$.

► **Lemma 23** (Persistence). *If $\Lambda \in ((\mathcal{I})^\perp)^\perp$, then there exists a $\Lambda' \in \mathcal{I}$ such that $\Lambda' \subseteq \Lambda$.*

► **Lemma 24** (Duality). *Given an intuitionistic interpolant \mathcal{I} , the empty sequent is derivable from $\mathcal{I} \cup (\mathcal{I})^\perp$ using the cut_2 rule and the contraction rule.*

Proof. Similar to Lem. 13. ◀

$$\begin{array}{c}
 \frac{\overline{\mathcal{R}, \Gamma_1, x : p \mid \Gamma_2 \vdash \Delta_1 \mid x : p, \Delta_2 \parallel \{ \vdash x : p \}} \text{ id}}{\overline{\mathcal{R}, \Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid x : \top, \Delta_2 \parallel \{ \vdash x : \perp \}} \top} \quad \frac{\overline{\mathcal{R}, \Gamma_1 \mid x : p, \Gamma_2 \vdash \Delta_1 \mid x : p, \Delta_2 \parallel \{ \vdash x : \perp \}} \text{ id}}{\overline{\mathcal{R}, \Gamma_1 \mid x : \perp, \Gamma_2 \vdash \Delta_1 \mid \Delta_2 \parallel \{ \vdash x : \perp \}} \perp} \\
 \frac{\overline{\mathcal{R}, \Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid \Delta_2 \parallel \mathcal{I}}}{\overline{\mathcal{R}, \Gamma_2 \mid \Gamma_1 \vdash \Delta_2 \mid \Delta_1 \parallel (\mathcal{I})^\perp} \text{ orth}} \\
 \frac{\overline{\mathcal{R}, Rxy, \Gamma_1 \mid x : A, y : A, \Gamma_2 \vdash \Delta_1 \mid \Delta_2 \parallel \mathcal{I}}}{\overline{\mathcal{R}, Rxy, \Gamma_1 \mid x : A, \Gamma_2 \vdash \Delta_1 \mid \Delta_2 \parallel \mathcal{I}} \text{ monl}} \quad \frac{\overline{\mathcal{R}, Rxy, \Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid x : A, y : A, \Delta_2 \parallel \mathcal{I}}}{\overline{\mathcal{R}, Rxy, \Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid y : A, \Delta_2 \parallel \mathcal{I}} \text{ monr}} \\
 \frac{\overline{\mathcal{R}, \Gamma_1 \mid x : A, \Gamma_2 \vdash \Delta_1 \mid \Delta_2 \parallel \mathcal{I}_1} \quad \overline{\mathcal{R}, \Gamma_1 \mid x : B, \Gamma_2 \vdash \Delta_1 \mid \Delta_2 \parallel \mathcal{I}_2}}{\overline{\mathcal{R}, \Gamma_1 \mid x : A \vee B, \Gamma_2 \vdash \Delta_1 \mid \Delta_2 \parallel \mathcal{I}_1 \cup \mathcal{I}_2} \vee_L} \\
 \frac{\overline{\mathcal{R}, \Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid x : A, \Delta_2 \parallel \mathcal{I}_1} \quad \overline{\mathcal{R}, \Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid x : B, \Delta_2 \parallel \mathcal{I}_2}}{\overline{\mathcal{R}, \Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid x : A \wedge B, \Delta_2 \parallel \mathcal{I}_1 \cup \mathcal{I}_2} \wedge_R} \\
 \frac{\overline{\mathcal{R}, \Gamma_1 \mid x : A, x : B, \Gamma_2 \vdash \Delta_1 \mid \Delta_2 \parallel \mathcal{I}}}{\overline{\mathcal{R}, \Gamma_1 \mid x : A \wedge B, \Gamma_2 \vdash \Delta_1 \mid \Delta_2 \parallel \mathcal{I}} \wedge_L} \quad \frac{\overline{\mathcal{R}, \Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid x : A, x : B, \Delta_2 \parallel \mathcal{I}}}{\overline{\mathcal{R}, \Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid x : A \vee B, \Delta_2 \parallel \mathcal{I}} \vee_R} \\
 \frac{\overline{\mathcal{R}, \Gamma_1 \mid x : A \supset B, \Gamma_2 \vdash \Delta_1 \mid x : A, \Delta_2 \parallel \mathcal{I}_2} \quad \overline{\mathcal{R}, \Gamma_1 \mid x : B, \Gamma_1 \vdash \Delta_1 \mid \Delta_2 \parallel \mathcal{I}_1}}{\overline{\mathcal{R}, \Gamma_1 \mid x : A \supset B, \Gamma_2 \vdash \Delta_1 \mid \Delta_2 \parallel \mathcal{I}_1 \cup \mathcal{I}_2} \supset_L} \\
 \frac{\overline{\mathcal{R}, \Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid x : A \prec B, \Delta_2 \parallel \mathcal{I}_1 \cup \mathcal{I}_2}}{\overline{\mathcal{R}, \Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid x : A \prec B, \Delta_2 \parallel \mathcal{I}_1 \cup \mathcal{I}_2} \prec_R} \\
 \frac{\overline{\mathcal{R}, Ryx, \Gamma_1 \mid y : A, \Gamma_2 \vdash \Delta_1 \mid y : B, \Delta_2 \parallel \mathcal{I}}}{\overline{\mathcal{R}, \Gamma_1 \mid x : A \prec B, \Gamma_2 \vdash \Delta_1 \mid \Delta_2 \parallel \prec \mathcal{I}_x^y} \prec_L} \quad \frac{\overline{\mathcal{R}, Rxy, \Gamma_1 \mid y : A, \Gamma_2 \vdash \Delta_1 \mid y : B, \Delta_2 \parallel \mathcal{I}}}{\overline{\mathcal{R}, \Gamma_1 \mid \Gamma_2 \vdash \Delta_1 \mid x : A \supset B, \Delta_2 \parallel \supset \mathcal{I}_x^y} \supset_R}
 \end{array}$$

■ **Figure 8** The calculus BilntLI used to compute interpolants for Bilnt. In \supset_R and \prec_L , y is fresh.

► **Definition 25.** Let \mathcal{I} be the interpolant below:

$$\left\{ (\Gamma_1, y : C_{1,1}, \dots, y : C_{1,k_1} \vdash \Delta_1, y : D_{1,1}, \dots, y : D_{1,j_1}), \dots, \right. \\
 \left. (\Gamma_n, y : C_{n,1}, \dots, y : C_{n,k_n} \vdash \Delta_n, y : D_{n,1}, \dots, y : D_{n,j_n}) \right\}$$

where y does not occur in $\Gamma_1, \dots, \Gamma_n, \Delta_1, \dots, \Delta_n$. The interpolants $\prec \mathcal{I}_y^x$ and $\supset \mathcal{I}_y^x$ are shown below where empty conjunction denotes \top and empty disjunction denotes \perp :

$$\begin{aligned}
 \prec \mathcal{I}_y^x &= \left\{ (\Gamma_1, x : \bigwedge_{i=1}^{k_1} C_{1,i} \prec \bigvee_{i=1}^{j_1} D_{1,i} \vdash \Delta_1), \dots, (\Gamma_n, x : \bigwedge_{i=1}^{k_n} C_{n,i} \prec \bigvee_{i=1}^{j_n} D_{n,i} \vdash \Delta_n) \right\} \\
 \supset \mathcal{I}_y^x &= \left\{ (\Gamma_1 \vdash \Delta_1, x : \bigwedge_{i=1}^{k_1} C_{1,i} \supset \bigvee_{i=1}^{j_1} D_{1,i}), \dots, (\Gamma_n \vdash \Delta_n, x : \bigwedge_{i=1}^{k_n} C_{n,i} \supset \bigvee_{i=1}^{j_n} D_{n,i}) \right\}
 \end{aligned}$$

The proof system BilntLI for constructing intuitionistic interpolants is given in Fig. 8.

► **Lemma 26.** If $\mathcal{R}, \Gamma_1, \Gamma_2 \Vdash \Delta_1, \Delta_2$, then there exists an \mathcal{I} such that $\mathcal{R}, \Gamma_1 \mid \Gamma_2 \Vdash \Delta_1 \mid \Delta_2 \parallel \mathcal{I}$, $\text{var}(\mathcal{I}) \subseteq \text{var}(\Gamma_1, \Delta_1) \cap \text{var}(\Gamma_2, \Delta_2)$, and all labels in \mathcal{I} also occur in $\mathcal{R}, \Gamma_1, \Delta_1$ or Γ_2, Δ_2 .

Proof. Induction on the height of the proof of $\mathcal{R}, \Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2$ using rules of BilntLI. ◀

The main technical lemma below asserts that the interpolants constructed via the proof system in Fig. 8 obey duality properties which are essential for proving the main theorem (Thm. 30). The proof of this lemma can be found in [16].

► **Lemma 27.** For all $\mathcal{R}, \Gamma_1, \Gamma_2, \Delta_1, \Delta_2$ and \mathcal{I} , if $\mathcal{R}, \Gamma_1 \mid \Gamma_2 \Vdash \Delta_1 \mid \Delta_2 \parallel \mathcal{I}$, then

1. For all $(\Sigma \vdash \Theta) \in \mathcal{I}$, we have $\mathcal{R}, \Gamma_1, \Sigma \Vdash \Theta, \Delta_1$ and
2. For all $(\Lambda \vdash \Omega) \in (\mathcal{I})^\perp$, we have $\mathcal{R}, \Gamma_2, \Lambda \Vdash \Omega, \Delta_2$.

Given a sequent Λ , we denote with Λ^L (resp., Λ^R) the multiset of labelled formulas on the left (resp. right) hand side of Λ . The following two lemmas are counterparts of Lem. 12 and Lem. 14. Lem. 28 essentially states that in a specific case, an interpolant can be interpreted

straightforwardly as a conjunction of implications. Its proof is given in [16]. The proof of Lem. 29 follows the same pattern as in the proof of Lem. 14.

► **Lemma 28.** *Let $\mathcal{I} = \{(\Sigma_1 \vdash \Theta_1), \dots, (\Sigma_n \vdash \Theta_n)\}$ be an interpolant with*

$$(\Sigma_i \vdash \Theta_i) = (x : C_{i,1}, \dots, x : C_{i,k_i} \vdash x : D_{i,1}, \dots, x : D_{i,j_i}) \text{ for each } 1 \leq i \leq n.$$

If $\Sigma_i, \Gamma \Vdash \Theta_i$, for all $(\Sigma_i \vdash \Theta_i) \in \mathcal{I}$, and every formula in Γ is labelled with x , then

$$\Gamma \Vdash x : \bigwedge_{i=1}^n \left(\bigwedge_{m=1}^{k_i} C_{i,m} \supset \bigvee_{m=1}^{j_i} D_{i,m} \right).$$

► **Lemma 29.** *Let $\mathcal{I} = \{(\Sigma_1 \vdash \Theta_1), \dots, (\Sigma_n \vdash \Theta_n)\}$ be an interpolant with*

$$(\Sigma_i \vdash \Theta_i) = (x : C_{i,1}, \dots, x : C_{i,k_i} \vdash x : D_{i,1}, \dots, x : D_{i,j_i}) \text{ for each } 1 \leq i \leq n.$$

If $\mathcal{R}, \Sigma_i, \Gamma \Vdash \Delta, \Theta_i$ for all $(\Sigma_i \vdash \Theta_i) \in (\mathcal{I})^\perp$, then

$$\mathcal{R}, \Gamma, x : \bigwedge_{i=1}^n \left(\bigwedge_{m=1}^{k_i} C_{i,m} \supset \bigvee_{m=1}^{j_i} D_{i,m} \right) \Vdash \Delta.$$

Proof. Follows from Lem. 24 and is similar to Lem. 14. ◀

► **Theorem 30.** *If $\Vdash x : A \supset B$, then there exists a C such that (i) $\text{var}(C) \subseteq \text{var}(A) \cap \text{var}(B)$ and (ii) $\Vdash x : A \supset C$ and $\Vdash x : C \supset B$.*

► **Corollary 31.** *The logic BiInt has the Craig interpolation property.*

5 Conclusion and Future work

We have presented a novel approach to proving the interpolation theorem for a range of logics possessing a nested sequent calculus. The key insight in our approach is the generalisation of the interpolation theorem to allow sets of sequents as interpolants. There is a natural definition of duality between interpolants via cut. We have shown that our method can be used to prove interpolation for logics for which interpolation was known to be difficult to prove.

We intend to apply our approach to bi-intuitionistic linear logic (BiILL) [4]. Unlike tense logics and bi-intuitionistic logic, there is no obvious Kripke semantics for BiILL, so Kuznets et. al.'s approach is not immediately applicable, and it seems a proof-theoretic approach like ours would offer some advantage. We conjecture that the key insight in our work, i.e., the generalisation of interpolants to sets of sequents and the use of orthogonality to define duality between interpolants, can be extended to the linear logic setting; for example, via a similar notion of orthogonality as in multiplicative linear logic [5].

References


- 1 Kai Brännler. Deep sequent systems for modal logic. *Arch. Math. Log.*, 48(6):551–577, 2009.
- 2 Agata Ciabattoni, Nikolaos Galatos, and Kazushige Terui. From Axioms to Analytic Rules in Nonclassical Logics. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 229–240, 2008. doi:10.1109/LICS.2008.39.
- 3 Agata Ciabattoni, Tim Lyon, and Revantha Ramanayake. From Display to Labelled Proofs for Tense Logics. In Sergei Artemov and Anil Nerode, editors, *Logical Foundations of Computer Science*, pages 120–139, Cham, 2018. Springer International Publishing.

- 4 Ranald Clouston, Jeremy E. Dawson, Rajeev Goré, and Alwen Tiu. Annotation-Free Sequent Calculi for Full Intuitionistic Linear Logic. In *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPICs*, pages 197–214. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPICs.CSL.2013.197.
- 5 Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Arch. Math. Log.*, 28(3):181–203, 1989. doi:10.1007/BF01622878.
- 6 Melvin Fitting and Roman Kuznets. Modal interpolation via nested sequents. *Ann. Pure Appl. Logic*, 166(3):274–305, 2015. doi:10.1016/j.apal.2014.11.002.
- 7 Rajeev Goré. Substructural Logics on Display. *Logic Journal of the IGPL*, 6(3):451–504, 1998.
- 8 Rajeev Goré, Linda Postniece, and Alwen Tiu. Cut-elimination and proof-search for bi-intuitionistic logic using nested sequents. In *Advances in Modal Logic 7, papers from the seventh conference on “Advances in Modal Logic,” held in Nancy, France, 9-12 September 2008*, pages 43–66, 2008. URL: <http://www.aiml.net/volumes/volume7/Gore-Postniece-Tiu.pdf>.
- 9 Rajeev Goré, Linda Postniece, and Alwen Tiu. On the correspondence between display postulates and deep inference in nested sequent calculi for tense logics. *Log. Methods Comput. Sci.*, 7(2):2:8, 38, 2011.
- 10 Ryo Kashima. Cut-free sequent calculi for some tense logics. *Studia Logica*, 53(1):119–136, 1994. doi:10.1007/BF01053026.
- 11 Hitoshi Kihara and Hiroakira Ono. Interpolation Properties, Beth Definability Properties and Amalgamation Properties for Substructural Logics. *Journal of Logic and Computation*, 20(4), August 2010.
- 12 Tomasz Kowalski and Hiroakira Ono. Analytic Cut and interpolation for bi-intuitionistic Logic. *Rew. Symb. Logic*, 10(2):259–283, 2017. doi:10.1017/S175502031600040X.
- 13 Roman Kuznets. Multicomponent proof-theoretic method for proving interpolation properties. *Ann. Pure Appl. Logic*, 169(12):1369–1418, 2018.
- 14 Roman Kuznets and Björn Lellmann. Interpolation for Intermediate Logics via Hyper- and Linear Nested Sequents. In *Proc. AiML 2018*. Kings College Publications, 2018.
- 15 Carsten Lutz and Frank Wolter. Foundations for Uniform Interpolation and Forgetting in Expressive Description Logics. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 989–995, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-170.
- 16 Tim Lyon, Alwen Tiu, Ranald Clouston, and Rajeev Goré. Syntactic Interpolation for Tense Logics and Bi-Intuitionistic Logic via Nested Sequents. *CoRR*, abs/1910, 2019. arXiv:1910.05215.
- 17 S. Maehara. Craig no interpolation theorem (in Japanese). *Suugaku*, 12:235–237, 1960.
- 18 Kenneth L. McMillan. Interpolation and Model Checking. In *Handbook of Model Checking*, pages 421–446. Springer, 2018.
- 19 Sara Negri. Proof Analysis in Modal Logic. *J. Philosophical Logic*, 34(5-6):507–544, 2005.
- 20 Luís Pinto and Tarmo Uustalu. A proof-theoretic study of bi-intuitionistic propositional sequent calculus. *J. Log. Comput.*, 28(1):165–202, 2018. doi:10.1093/logcom/exx044.
- 21 Francesca Poggiolesi. A Cut-Free Simple Sequent Calculus for Modal Logic S5. *Rew. Symb. Logic*, 1(1):3–15, 2008. doi:10.1017/S1755020308080040.
- 22 Cecylia Rauszer. *An algebraic and Kripke-style approach to a certain extension of intuitionistic logic*. Instytut Matematyczny Polskiej Akademi Nauk, 1980. URL: <http://eudml.org/doc/268511>.
- 23 Balder ten Cate, Enrico Franconi, and Inanç Seylan. Beth Definability in Expressive Description Logics. *J. Artif. Intell. Res.*, 48:347–414, 2013.
- 24 Alwen Tiu, Egor Ianovski, and Rajeev Goré. Grammar Logics in Nested Sequent Calculus: Proof Theory and Decision Procedures. In *Advances in Modal Logic 9, papers from the ninth conference on “Advances in Modal Logic,” held in Copenhagen, Denmark, 22-25 August 2012*, pages 516–537, 2012. URL: <http://www.aiml.net/volumes/volume9/Tiu-Ianovski-Gore.pdf>.
- 25 Anne Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. CUP, 1996.

The Keys to Decidable HyperLTL Satisfiability: Small Models or Very Simple Formulas

Corto Mascle

ENS Paris-Saclay, Cachan, France
corto.mascle@ens-paris-saclay.fr

Martin Zimmermann 

University of Liverpool, Liverpool, United Kingdom
martin.zimmermann@liverpool.ac.uk

Abstract

HyperLTL, the extension of Linear Temporal Logic by trace quantifiers, is a uniform framework for expressing information flow policies by relating multiple traces of a security-critical system. HyperLTL has been successfully applied to express fundamental security policies like noninterference and observational determinism, but has also found applications beyond security, e.g., distributed protocols and coding theory. However, HyperLTL satisfiability is undecidable as soon as there are existential quantifiers in the scope of a universal one. To overcome this severe limitation to applicability, we investigate here restricted variants of the satisfiability problem to pinpoint the decidability border.

First, we restrict the space of admissible models and show decidability when restricting the search space to models of bounded size or to finitely representable ones. Second, we consider formulas with restricted nesting of temporal operators and show that nesting depth one yields decidability for a slightly larger class of quantifier prefixes. We provide tight complexity bounds in almost all cases.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Hyperproperties, Linear Temporal Logic, Satisfiability

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.29

Related Version Full version containing all proofs omitted due to space restrictions available at <https://arxiv.org/abs/1907.05070>.

Funding Partially funded by EPSRC grant EP/S032207/1.

1 Introduction

The introduction of temporal logics for the specification of information flow policies [3] was a significant milestone in the long and successful history of applying logics in computer science [16]. Probably the most important representative of these logics is HyperLTL [3], which extends Linear Temporal Logic (LTL) [23] by trace quantifiers. This addition allows to express properties that relate multiple execution traces, which is typically necessary to capture the flow of information [4]. In contrast, LTL, currently the most influential specification language for reactive systems, is only able to express properties of single traces.

HyperLTL provides a uniform framework for expressing information flow policies in a formalism with intuitive syntax and semantics, and for the automated verification of these policies: A wide range of policies from the literature [15, 19, 20, 21, 22, 27] with specialized verification algorithms is expressible in HyperLTL, i.e., universal HyperLTL verification algorithms are applicable to all of them.

As an example, consider a system with a set I of inputs, which contains a hidden input $h \in I$, and an output o . Now, noninterference [15] between h and o requires that no information about h is leaked via o , i.e., for all execution traces π and π' , if the inputs in



© Corto Mascle and Martin Zimmermann;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 29; pp. 29:1–29:16

Leibniz International Proceedings in Informatics



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

π and π' only differ in h , then they have the same output at all times. Formally, this is captured by the HyperLTL formula

$$\forall\pi.\forall\pi'.\left(\mathbf{G}\bigwedge_{i\in I\setminus\{h\}}(i_\pi \Leftrightarrow i_{\pi'})\right) \Rightarrow \mathbf{G}(o_\pi \Leftrightarrow o_{\pi'}).$$

As another example, consider a system with a public output o and a secret output s (we consider only one of each for simplicity). One may want to express that the behaviour of the secret output cannot be inferred from the behaviour of the public one. HyperLTL can express the property that for all executions of the system, there exists another execution with the same behaviour of o but a different behaviour of s , using the formula

$$\forall\pi.\exists\pi'.\mathbf{G}(o_\pi \Leftrightarrow o_{\pi'}) \wedge \mathbf{F}\neg(s_\pi \Leftrightarrow s_{\pi'}).$$

Today, there are tools for model checking HyperLTL properties [6, 13], for checking satisfiability of HyperLTL properties [9, 11], for synthesizing reactive systems from HyperLTL properties [10], and for runtime monitoring of HyperLTL properties [1, 2, 12]. Furthermore, the extraordinary expressiveness of HyperLTL has been exhibited [14] and connections to first and second-order predicate logics have been established [5, 14].

The major drawback of HyperLTL is the usual price one has to pay for great expressiveness: prohibitively high worst-case complexity. In particular, model checking finite Kripke structures against HyperLTL formulas is nonelementary [3] and satisfiability is even undecidable [8]. These results have to be contrasted with model checking and satisfiability being PSPACE-complete for LTL [25], problems routinely solved in real-life applications [18].

Due to the sobering state of affairs, it is imperative to find fragments of the logic with (more) tractable complexity. In this work, we focus on the satisfiability problem, the most fundamental decision problem for a logic. Nevertheless, it has many applications in verification, e.g., checking the equivalence and implication of specifications can be reduced to satisfiability. Finally, the question whether a property given by some HyperLTL formula is realizable by some system is also a satisfiability problem.

A classical attempt to overcome the undecidability of the satisfiability problem is to restrict the number of quantifier alternations of the formulas under consideration. In fact, the alternation depth is the measure underlying the nonelementary complexity of the HyperLTL model checking problem [3]. However, the situation is different for the satisfiability problem: It is undecidable even when restricted to $\forall\exists$ formulas, i.e., formulas starting with one universal quantifier followed by a single existential one [8]. All remaining prefix classes are decidable by reductions to the LTL satisfiability problem, e.g., the satisfiability problem is PSPACE-complete for the alternation-free prefix classes \exists^* and \forall^* and EXSPACE-complete for the class $\exists^*\forall^*$ [8].

However, there are more complexity measures beyond the alternation depth that can be restricted in order to obtain tractable satisfiability problems, both on formulas and on models. The latter case is of particular interest, since it is known that not every satisfiable HyperLTL has a “simple” model, for various formalizations of “simple” [14]. Thus, for those formulas, such a restriction could make a significant difference. Furthermore, from a more practical point of view, one is often interested in whether there is a, say, finite model while the existence of an intricate infinite model may not be useful.

We study the satisfiability problem for formulas with restricted quantifier prefixes and restricted temporal depth [7], which measures the nesting of temporal operators. Our main result here shows that satisfiability is even undecidable for formulas of the form $\forall^2\exists^*\varphi$, where φ has temporal depth one and only uses eventually \mathbf{F} and always \mathbf{G} , i.e., it is a

Boolean combination of formulas $\mathbf{F} \varphi'$ with propositional φ' . Thereby, we strengthen the previous undecidability result for $\forall\exists$ by bounding the temporal depth to one, but at the price of a second universal quantifier. Moreover, we clarify the border between decidability and undecidability at temporal depth two: Using only one universally quantified variable, temporal depth one, and only \mathbf{F} , \mathbf{G} , and nested applications of next \mathbf{X} leads to decidability. Finally, we show that every HyperLTL formula can be transformed into an equisatisfiable $\forall^2\exists^*$ formula of temporal depth two, i.e., this fragment already captures the full complexity of the satisfiability problem.

Thus, the overall picture is still rather bleak: if one only restricts the formula then the islands of decidability are very small. Phrased differently, even very simple formulas are extremely expressive and allow to encode computations of Turing-complete devices in their models. However, note that such models are necessarily complex, as they need to be able to encode an unbounded amount of information.

Thus, we also consider satisfiability problems for arbitrary formulas, but with respect to restricted models which do not allow to encode such computations. In particular, we consider three variants of increasing complexity: Checking whether a given HyperLTL formula has a model of a given cardinality k is EXPSPACE-complete, whether it has a model containing only ultimately periodic traces of length at most k is N2EXPTIME-complete, and checking whether it has a model induced by a Kripke structure with k states is TOWER-complete. The last result is even true for a fixed Kripke structure, which therefore has implications for the complexity of the model checking problem as well. Thus, the situation is more encouraging when checking for the existence of small models: satisfiability becomes decidable, even with (relatively) moderate complexity in the first two cases.

However, as argued above, all three approaches are (necessarily) incomplete: There are satisfiable formulas that have only infinite models, satisfiable formulas that have only non-ultimately periodic models, and satisfiable formulas that have no ω -regular models [14], a class of models that includes all those that are induced by a finite Kripke structure.

All in all, our work shows that HyperLTL satisfiability remains a challenging problem, but we have provided a complete classification of the tractable cases in terms of alternation depth, temporal depth, and representation of the model (for formulas without until).

2 Definitions

Fix a finite set AP of atomic propositions. A *valuation* is a subset of AP. A *trace* over AP is a map $t: \mathbb{N} \rightarrow 2^{\text{AP}}$, denoted by $t(0)t(1)t(2)\dots$, i.e., an infinite sequence of valuations. The set of all traces over AP is denoted by $(2^{\text{AP}})^\omega$. The *projection* of t to AP' is the trace $(t(0) \cap \text{AP}')(t(1) \cap \text{AP}')(t(2) \cap \text{AP}')\dots$ over AP' . A trace t is *ultimately periodic*, if $t = x \cdot y^\omega$ for some $x, y \in (2^{\text{AP}})^+$, i.e., there are $s, p > 0$ with $t(n) = t(n + p)$ for all $n \geq s$.

The formulas of HyperLTL are given by the grammar

$$\begin{aligned} \varphi &::= \exists\pi.\varphi \mid \forall\pi.\varphi \mid \psi \\ \psi &::= a_\pi \mid \neg\psi \mid \psi \vee \psi \mid \mathbf{X}\psi \mid \psi \mathbf{U} \psi \end{aligned}$$

where a ranges over atomic propositions in AP and where π ranges over a fixed countable set \mathcal{V} of *trace variables*. Conjunction, implication, equivalence, and exclusive disjunction \oplus , as well as the temporal operators eventually \mathbf{F} and always \mathbf{G} are derived as usual. A *sentence* is a closed formula, i.e., a formula without free trace variables. The *size* of a formula φ , denoted by $|\varphi|$, is its number of distinct subformulas.

29:4 The Keys to Decidable HyperLTL Satisfiability

The semantics of HyperLTL is defined with respect to a *trace assignment*, a partial mapping $\Pi: \mathcal{V} \rightarrow (2^{\text{AP}})^\omega$. The assignment with empty domain is denoted by Π_\emptyset . Given a trace assignment Π , a trace variable π , and a trace t we denote by $\Pi[\pi \rightarrow t]$ the assignment that coincides with Π everywhere but at π , which is mapped to t . We also use shorthand notation like $[\pi_1 \rightarrow t_1, \dots, \pi_n \rightarrow t_n]$ and $[(\pi_i \rightarrow t_i)_{1 \leq i \leq n}]$ for $\Pi_\emptyset[\pi_1 \rightarrow t_1] \dots [\pi_n \rightarrow t_n]$, if the π_i are pairwise different. Furthermore, $\Pi[j, \infty)$ denotes the trace assignment mapping every π in Π 's domain to $\Pi(\pi)(j)\Pi(\pi)(j+1)\Pi(\pi)(j+2)\dots$.

For sets T of traces and trace assignments Π we define

- $(T, \Pi) \models a_\pi$, if $a \in \Pi(\pi)(0)$,
- $(T, \Pi) \models \neg\psi$, if $(T, \Pi) \not\models \psi$,
- $(T, \Pi) \models \psi_1 \vee \psi_2$, if $(T, \Pi) \models \psi_1$ or $(T, \Pi) \models \psi_2$,
- $(T, \Pi) \models \mathbf{X}\psi$, if $(T, \Pi[1, \infty)) \models \psi$,
- $(T, \Pi) \models \psi_1 \mathbf{U} \psi_2$, if there is a $j \geq 0$ such that $(T, \Pi[j, \infty)) \models \psi_2$ and for all $0 \leq j' < j$: $(T, \Pi[j', \infty)) \models \psi_1$,
- $(T, \Pi) \models \exists\pi.\varphi$, if there is a trace $t \in T$ such that $(T, \Pi[\pi \rightarrow t]) \models \varphi$, and
- $(T, \Pi) \models \forall\pi.\varphi$, if for all traces $t \in T$: $(T, \Pi[\pi \rightarrow t]) \models \varphi$.

We say that T *satisfies* a sentence φ if $(T, \Pi_\emptyset) \models \varphi$. In this case, we write $T \models \varphi$ and say that T is a *model* of φ . Conversely, satisfaction of quantifier-free formulas does not depend on T . Hence, we say that Π *satisfies* a quantifier-free ψ if $(\emptyset, \Pi) \models \psi$ and write $\Pi \models \psi$ (assuming Π is defined on all trace variables that appear in ψ).

The *alternation depth* of a HyperLTL sentence φ , denoted by $\text{ad}(\varphi)$, is defined as its number of quantifier alternations. Its *temporal depth*, denoted by $\text{td}(\varphi)$, is defined as the maximal depth of the nesting of temporal operators in the sentence. Formally, td and ad are defined as follows:

- | | |
|--|--|
| ■ $\text{td}(a_\pi) = 0$ | ■ $\text{ad}(\exists\pi.\psi) = 0$ for quantifier-free ψ |
| ■ $\text{td}(\neg\psi) = \text{td}(\psi)$ | ■ $\text{ad}(\forall\pi.\psi) = 0$ for quantifier-free ψ |
| ■ $\text{td}(\psi_1 \vee \psi_2) = \max(\text{td}(\psi_1), \text{td}(\psi_2))$ | ■ $\text{ad}(\exists\pi.\exists\pi'.\varphi) = \text{ad}(\exists\pi'.\varphi)$ |
| ■ $\text{td}(\mathbf{X}\psi) = 1 + \text{td}(\psi)$ | ■ $\text{ad}(\forall\pi.\forall\pi'.\varphi) = \text{ad}(\forall\pi'.\varphi)$ |
| ■ $\text{td}(\psi_1 \mathbf{U} \psi_2) = 1 + \max(\text{td}(\psi_1), \text{td}(\psi_2))$ | ■ $\text{ad}(\exists\pi.\forall\pi'.\varphi) = 1 + \text{ad}(\forall\pi'.\varphi)$ |
| ■ $\text{td}(\exists\pi.\varphi) = \text{td}(\varphi)$ | ■ $\text{ad}(\forall\pi.\exists\pi'.\varphi) = 1 + \text{ad}(\exists\pi'.\varphi)$ |
| ■ $\text{td}(\forall\pi.\varphi) = \text{td}(\varphi)$ | |

Although HyperLTL sentences are required to be in prenex normal form, they are closed under Boolean combinations, which can easily be seen by transforming such formulas into prenex normal form. Note that this transformation can be implemented such that it changes neither the temporal nor alternation depth, and can be performed in polynomial time.

The fragment $\text{HyperLTL}^1(\mathbf{F}, \mathbf{G})$ contains formulas of temporal depth one using only \mathbf{F} and \mathbf{G} as temporal operators, and $\text{HyperLTL}^1(\mathbf{F}, \mathbf{G}, \mathbf{X}^*)$ contains formulas using only \mathbf{F} , \mathbf{G} , and \mathbf{X} as temporal operators and of temporal depth one, however we allow iterations of the \mathbf{X} operator. Formally, $\text{HyperLTL}^1(\mathbf{F}, \mathbf{G}, \mathbf{X}^*)$ formulas are generated by the grammar

$$\begin{aligned} \varphi &::= \exists\pi.\varphi \mid \forall\pi.\varphi \mid \psi \\ \psi &::= \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \mathbf{X}^n\psi' \mid \mathbf{F}\psi' \mid \mathbf{G}\psi' \mid \psi' \\ \psi' &::= a_\pi \mid \neg\psi' \mid \psi' \vee \psi' \mid \psi' \wedge \psi' \end{aligned}$$

where n ranges over the natural numbers. The grammar for $\text{HyperLTL}^1(\mathbf{F}, \mathbf{G})$ is obtained by removing $\mathbf{X}^n\psi'$ from the grammar above.

Also, we use standard notation for classes of formulas with restricted quantifier prefixes, e.g., $\forall^2\exists^*$ denotes the set of HyperLTL formulas in prenex normal form with two universal quantifiers followed by an arbitrary number of existential quantifiers, but no other quantifiers.

Finally, we encounter various complexity classes, classical ones from NP to N2EXPTIME, as well as TOWER (see, e.g., [24]). Intuitively, TOWER is the set of problems that can be solved by a Turing machine that, on an input of size n , stops in time $2^{2^{\dots^2}}$, with the height of the tower of exponents bounded by $b(n)$, where b is a fixed elementary function. The reductions presented in this work are polynomial time reductions unless otherwise stated.

3 Satisfiability for Restricted Classes of Models

The satisfiability problem “Given a HyperLTL sentence φ , does φ have a nonempty model?” is undecidable, even when restricted to finite models [8]. Hence, one has to consider simpler problems to regain decidability. In this section, we simplify the problem by checking only for the existence of *simple* models, for the following three formalizations of simplicity, where the bound k is always part of the input:

- Models of cardinality at most k (Theorem 1).
- Models containing only ultimately periodic traces xy^ω with $|x| + |y| \leq k$ (Theorem 2).
- Models induced by finite-state systems with at most k states (Theorem 3).

In every case, we allow arbitrary HyperLTL formulas as input and encode k in binary.

With the following result, we determine the complexity of checking satisfiability with respect to models of bounded cardinality. The algorithm uses a technique introduced by Finkbeiner and Hahn [8, Theorem 3] that allows us to replace existential and universal quantification by disjunctions and conjunctions, if the model is finite. Similarly, the lower bound also follows from Finkbeiner and Hahn.

► **Theorem 1.** *The following problem is EXPSPACE-complete: Given a HyperLTL sentence φ and $k \in \mathbb{N}$ (in binary), does φ have a model with at most k traces?*

Proof. For the EXPSPACE upper bound, one can check, given φ and k , satisfiability of the sentence $\exists\pi_1 \dots \exists\pi_k. \bar{\varphi}$ where $\bar{\varphi}$ is defined inductively as follows:

- $\bar{\varphi} = \varphi$ if φ is quantifier-free.
- $\overline{\forall\pi. \varphi} = \bigwedge_{i=1}^k \bar{\varphi}[\pi \leftarrow \pi_i]$.
- $\overline{\exists\pi. \varphi} = \bigvee_{i=1}^k \bar{\varphi}[\pi \leftarrow \pi_i]$.

Here, $\bar{\varphi}[\pi \leftarrow \pi_i]$ is obtained from $\bar{\varphi}$ by replacing every occurrence of π by π_i . This sentence states the existence of at most k traces satisfying φ by replacing every quantifier by an explicit conjunction or disjunction over the possible assignments.

The resulting sentence is of size at most $|\varphi|k^{|\varphi|} + k$, which is exponential in the size of the input and its satisfiability can be checked in polynomial space in the size of the resulting formula [8]. As a result, the problem is in EXPSPACE as well.

Finkbeiner and Hahn showed that satisfiability is EXPSPACE-complete for sentences of the form $\exists^*\forall^*$ [8]. This implies EXPSPACE-hardness of our problem, as if such a sentence, say with k existential quantifiers, is satisfiable then it has a model with at most k traces. ◀

As the algorithm proceeds by a reduction to the satisfiability problem for \exists^* formulas, which in turn is reduced to LTL satisfiability, one can show that a HyperLTL sentence φ has a model with k traces if and only if it has a model with k ultimately periodic traces.

Next, we consider another variant of the satisfiability problem, where we directly restrict the space of possible models to ultimately periodic ones of the form xy^ω with $|x| + |y| \leq k$.

As we encode k in binary, the length of those traces is exponential in the input and the cardinality of the model is bounded doubly-exponentially. This explains the increase in complexity in the following theorem in comparison to Theorem 1.

► **Theorem 2.** *The following problem is N2EXPTIME-complete: Given a HyperLTL sentence φ and $k \in \mathbb{N}$ (in binary), does φ have a model whose elements are of the form xy^ω with $|x| + |y| \leq k$?*

As expected, the complexity of the satisfiability problem increases the more traces one has at hand to encode computations. In Theorem 1, we have exponentially many; in Theorem 2, we have doubly-exponentially many. In our last theorem, we consider infinite sets of traces that are finitely representable by finite-state systems. Here, satisfiability becomes intractable, yet still decidable, even when restricted to formulas of temporal depth one.

Formally, a *Kripke structure* $\mathcal{K} = (Q, \delta, Q_0, \lambda)$ consists of a finite set Q of states, a set $Q_0 \subseteq Q$ of initial states, a transition function $\delta: Q \rightarrow 2^Q \setminus \{\emptyset\}$, and a labelling function $\lambda: Q \rightarrow 2^{\text{AP}}$. A *run* of \mathcal{K} is an infinite sequence $q_0q_1q_2 \cdots$ of states starting with $q_0 \in Q_0$ and such that $q_{j+1} \in \delta(q_j)$ for all $j \in \mathbb{N}$. A trace of \mathcal{K} is the sequence of labels $\lambda(q_0)\lambda(q_1)\lambda(q_2) \cdots$ associated to a run $q_0q_1q_2 \cdots$ of \mathcal{K} . The set of traces of \mathcal{K} is denoted by $T(\mathcal{K})$.

► **Theorem 3.** *The following problem is TOWER-complete: Given a HyperLTL sentence φ and $k \in \mathbb{N}$ (in binary), does φ have a model $T(\mathcal{K})$ for some Kripke structure \mathcal{K} with at most k states?*

Proof. Clarkson et al. presented a model-checking algorithm for HyperCTL* (and thus for HyperLTL, which is a fragment of HyperCTL*), and showed that its complexity is a tower of exponentials whose height is the alternation depth of the input sentence [3]. Thus, one can enumerate all Kripke structures with at most k states (up to isomorphism) and model-check them one by one in TOWER. This yields the desired upper bound, as there are “only” exponentially many (in k) Kripke structures with k states.

The lower bound is obtained by a reduction from the universality problem for star-free regular expressions with complementation. The equivalence problem for those expressions is TOWER-complete (under elementary reductions, which is standard for TOWER-complete problems), even for two-letter alphabets [24, 26]. As those expressions are closed by complementation and union, the universality problem is TOWER-complete as well.

Star-free expressions with complementation over $\{a, b\}$ are generated by the grammar

$$e ::= a \mid b \mid \varepsilon \mid \emptyset \mid e + e \mid ee \mid \neg e$$

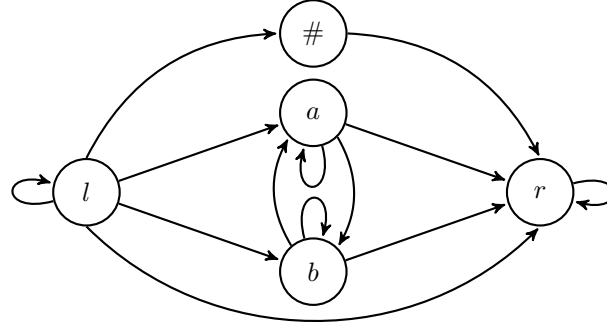
and have the obvious semantics inducing a language over $\{a, b\}^*$, denoted by e as well.

Let e be such an expression. We construct a HyperLTL sentence φ_e and a Kripke structure \mathcal{K} such that $T(\mathcal{K})$ is a model of φ_e if and only if e is universal. \mathcal{K} does not depend on e and is shown in Figure 1. As all sets of variables in \mathcal{K} are singletons, we indifferently use the notation a for the letter a and the singleton $\{a\}$. The set of traces induced by this Kripke structure is

$$T(\mathcal{K}) = l^\omega + l^*(a + b)^\omega + l^*(a + b)^*r^\omega + l^*\#r^\omega.$$

Given an expression e and a trace variable π , we inductively define a formula $\psi_{e,\pi}$ which expresses that when π is mapped by a trace assignment Π to a trace of \mathcal{K} of the form l^nwr^ω with $w \in \{a, b\}^*$, then $w \in e$ if and only if $(T(\mathcal{K}), \Pi) \models \psi_{e,\pi}$.

■ $\psi_{\emptyset,\pi} = a_\pi \wedge \neg a_\pi$: No trace assignment satisfies $\psi_{\emptyset,\pi}$, just as the language of \emptyset does not contain any word.



■ **Figure 1** The Kripke structure \mathcal{K} (all states are initial).

- $\psi_{\varepsilon, \pi} = \mathbf{G}(l_{\pi} \vee r_{\pi})$: $(T(\mathcal{K}), \Pi)$ with $\Pi(\pi) = l^n w r^\omega$ satisfies $\psi_{\varepsilon, \pi}$ if and only if $w = \varepsilon$.
- $\psi_{a, \pi} = \exists \tau. (\mathbf{F} \#_{\tau}) \wedge \mathbf{F}(a_{\pi}) \wedge \mathbf{G}(l_{\tau} \Leftrightarrow l_{\pi} \wedge r_{\tau} \Leftrightarrow r_{\pi})$: The traces of \mathcal{K} with an occurrence of $\#$ are the traces of the form $l^* \# r^\omega$. Thus, $(T(\mathcal{K}), \Pi)$ with $\Pi(\pi) = l^n w r^\omega$ satisfies $\psi_{a, \pi}$ if and only if $l^n w r^\omega$ is a copy of such a trace with $\#$ replaced by a , i.e., if and only if $w = a$.
- $\psi_{b, \pi} = \exists \tau. (\mathbf{F} \#_{\tau}) \wedge \mathbf{F}(b_{\pi}) \wedge \mathbf{G}(l_{\tau} \Leftrightarrow l_{\pi} \wedge r_{\tau} \Leftrightarrow r_{\pi})$: Similarly to $\psi_{a, \pi}$.
- $\psi_{e_1 + e_2, \pi} = \psi_{e_1, \pi} \vee \psi_{e_2, \pi}$.
- $\psi_{e_1 e_2, \pi} = \exists \pi_1. \exists \pi_2. \psi \wedge \psi'$ with

$$\psi = \mathbf{F} r_{\pi_1} \wedge \mathbf{F} r_{\pi_2} \wedge \mathbf{G}(\neg \#_{\pi_1} \wedge \neg \#_{\pi_2}) \wedge \psi_{e_1, \pi_1} \wedge \psi_{e_2, \pi_2}$$

expressing that π_1 and π_2 are of the form $l^{n_1} w_1 r^\omega$ and $l^{n_2} w_2 r^\omega$ with $w_1 \in e_1$ and $w_2 \in e_2$, and with

$$\psi' = \mathbf{G}(l_{\pi_2} \Leftrightarrow \neg r_{\pi_1}) \wedge \mathbf{G}(a_{\pi} \Leftrightarrow (a_{\pi_1} \vee a_{\pi_2}) \wedge b_{\pi} \Leftrightarrow (b_{\pi_1} \vee b_{\pi_2}))$$

expressing that $n_2 = n_1 + |w_1|$ and that $w = w_1 w_2$, where $\Pi(\pi) = l^n w r^\omega$. Thus, $(T(\mathcal{K}), \Pi)$ satisfies $\psi_{e_1 e_2, \pi}$ if and only if there exist $w_1 \in e_1, w_2 \in e_2$ such that $w = w_1 w_2$.

- $\psi_{\neg e, \pi} = \neg \psi_{e, \pi}$.

Although this inductive definition does not necessarily give a formula in prenex normal form, one can easily check that no quantifier is in the scope of a temporal operator, thus the resulting formula can be turned into a HyperLTL formula.

To conclude, consider the sentence $\varphi_e = \forall \pi. \mathbf{G} \neg r_{\pi} \vee \mathbf{F} \#_{\pi} \vee \psi_{e, \pi}$, which can again be brought into prenex normal form. Further, note that no temporal operator is in the scope of another one, thus φ_e has temporal depth one. The set $T(\mathcal{K})$ is a model of φ_e if and only if all its traces are in $\{a, b, l\}^\omega$, in $l^* \# r^\omega$, or of the form $l^* w r^\omega$ with $w \in e$. This is the case if and only if all words $w \in \{a, b\}^*$ are in the language of e , i.e., if and only if e is universal. ◀

As the Kripke structure \mathcal{K} in the lower bound proof above is fixed, we also obtain a novel hardness result for model-checking.

► **Corollary 4.** *HyperLTL model-checking a fixed Kripke structure with five states is TOWER-complete, even for sentences of temporal depth one.*

Note that one could already infer the TOWER-completeness of the model-checking problem by carefully examining the proof of Theorem 5 of [3] concerning HyperCTL* model-checking. The reduction from the satisfiability problem for QPTL presented there also works for HyperLTL, albeit with temporal depth larger than one.

■ **Table 1** Complexity of HyperLTL satisfiability in terms of quantifier prefixes and temporal depth. An asterisk * denotes that the upper bound only holds for until-free formulas. All lower bounds in the second column already hold for temporal depth two.

	temporal depth one	arbitrary temporal depth
\exists^* / \forall^*	NP-complete ([7]+[8])	PSPACE-complete ([8]+[25])
$\exists^*\forall^*$	NEXPTIME-complete (Thm. 12)	EXSPACE-complete ([8])
$\exists^*\forall\exists^*$	in N2EXPTIME* (Thm. 11)	undecidable ([8])
$\forall^2\exists^*$	undecidable (Thm. 9)	undecidable

4 Satisfiability for Restricted Classes of Formulas

After studying the HyperLTL satisfiability problem for classes of restricted models, but arbitrary formulas, we now consider restrictions on formulas, but arbitrary models. Recall that Finkbeiner and Hahn presented a complete picture in terms of quantifier prefixes: Satisfiability is PSPACE-complete for the alternation-free fragments \exists^* and \forall^* as well as EXSPACE-complete for $\exists^*\forall^*$. In all other cases, the problem is undecidable, i.e., as soon as there is a universal quantifier in front of an existential one.

In a sense, the decidable fragments are variants of LTL: Both alternation-free fragments can easily be reduced to LTL satisfiability while the $\exists^*\forall^*$ one is easily reducible to the \exists^* fragment, with an exponential blowup. Thus, the decidable fragments barely exceed the realm of LTL.

In this section, we consider another dimension to measure the complexity of formulas, temporal depth, i.e., we restrict the nesting of temporal operators. The hope is that in this setting, we can obtain decidability for larger quantifier prefix classes. However, a slight adaptation of Finkbeiner and Hahn’s undecidability result for $\forall\exists$, along with an application of Lemma 6 proven below, already shows undecidability for $\forall\exists$ formulas of temporal depth two and without untils.

Thus, we have to restrict our search to fragments of temporal depth one, which contain most of the information flow policies expressible in HyperLTL [3]. And indeed, we prove satisfiability decidable for $\exists^*\forall\exists^*$ HyperLTL¹(**F**, **G**, **X***) formulas. Thus, if the temporal depth is one and untils are excluded, then one can allow a universal quantifier in front of existential ones without losing decidability. This fragment includes, for example, the noninference property [21], as well as the second example presented in the introduction.

However, even allowing the smallest possible extension, i.e., adding a second universal quantifier, leads again to undecidability: HyperLTL satisfiability is undecidable for $\forall^2\exists^*$ formulas of temporal depth one using only **F** as temporal operator. Thus, satisfiability remains hard, even when severely restricting the temporal depth of formulas. Our results for temporal depth one are summarized in Table 1.

We begin this section by showing that every HyperLTL formula can be transformed in polynomial time into an equisatisfiable one with quantifier prefix $\forall^2\exists^*$ with temporal depth two. Thus, this fragment already captures the full complexity of the satisfiability problem. This transformation is later used in several proofs.

► **Theorem 5.** *For every HyperLTL sentence one can compute in polynomial time an equisatisfiable sentence of the form $\forall^2\exists^*$ with temporal depth at most two.*

We decompose the proof into three steps, formalized by the following three lemmas. We begin by reducing the temporal depth to at most two by adapting a construction of Demri and Schnoebelen, which associates to every LTL formula an equisatisfiable formula with temporal depth at most two [7].

► **Lemma 6.** *For every HyperLTL sentence $Q_1\pi_1 \dots Q_n\pi_n.\psi$ with quantifier-free ψ , one can compute in polynomial time an equisatisfiable sentence $Q_1\pi_1 \dots Q_n\pi_n.\exists\pi.\psi'$ with quantifier-free ψ' and temporal depth at most two.*

The idea is to add atomic propositions to witness the satisfaction of subformulas ψ' of ψ . We express the existence, for every n -tuple of traces (t_1, \dots, t_n) of the model, of a *witness trace*. For all $j \in \mathbb{N}$, for all subformulas ψ' of ψ , the valuation $[(\pi_i \rightarrow t_i[j, \infty))_{1 \leq i \leq n}]$ satisfies ψ' if and only if the associated atomic proposition is satisfied at position j of the witness trace.

Next, we turn the quantifier prefix into the form $\forall^*\exists^*$ without increasing the temporal depth.

► **Lemma 7.** *For every HyperLTL sentence φ , one can compute in polynomial time an equisatisfiable sentence φ' of the form $\forall^*\exists^*\psi$ with $\text{td}(\varphi') = \max(\text{td}(\varphi), 1)$.*

Here the key idea is to move existential quantifiers in the scope of universal ones after marking them with fresh atomic propositions: We can replace an $\exists\forall$ by a $\forall\exists$ if we require that the existentially quantified variable is now uniquely marked by a proposition (and therefore cannot depend on the universally quantified variable).

The construction presented in the proof of Lemma 7 may increase the number of universally quantified variables, but we can decrease that number to two without increasing the temporal or alternation depth. This step also completes the proof of Theorem 5.

► **Lemma 8.** *For every HyperLTL sentence φ of the form $\forall^*\exists^*\psi$ with quantifier-free ψ , one can compute in polynomial time an equisatisfiable sentence φ' of the form $\forall^2\exists^*\psi'$ where ψ' is quantifier-free and $\text{td}(\varphi') = \max(\text{td}(\varphi), 1)$.*

This can be achieved by merging several traces into one. To this end, we increase the set of atomic propositions by considering as new atomic propositions tuples of the previous atomic propositions, i.e., one trace now encodes a tuple of traces. However we cannot decrease the number of universal quantifiers below two this way, as we need two universal quantifiers to ensure that every possible combination of traces is represented in the model, i.e., any model of the resulting formula is the set of mergings of traces of another model.

Thus, $\forall^2\exists^*$ formulas with temporal depth two capture the complete complexity of the satisfiability problem for HyperLTL. As the latter problem is undecidable and as all reductions presented above are effective, we immediately obtain that satisfiability for $\forall^2\exists^*$ formulas with temporal depth two is also undecidable.

As alluded to above, an even stronger result can be obtained by strengthening the proof of Finkbeiner and Hahn for $\forall\exists$ formulas to only use temporal depth two.¹ Thus, only formulas of temporal depth one remain to be considered.

Before we start investigating this class let us quickly comment on why we disregard temporal depth zero: Every such sentence can easily be turned to an equisatisfiable instance of QBF, which is known to be solvable in polynomial space.

Thus, it only remains to consider formulas with arbitrary quantifier prefixes, but temporal depth one. Our main result of this section shows that even this problem is undecidable, even for $\text{HyperLTL}^1(\mathbf{F}, \mathbf{G})$ formulas with alternation depth one. Due to the restriction on the temporal depth, our encoding of a Minsky machine is more complicated than it would be with arbitrary temporal depth.

¹ Alternatively, one can also obtain a direct reduction from the Turing machine immortality problem [17] to satisfiability of $\forall\exists$ sentences of temporal depth two.

29:10 The Keys to Decidable HyperLTL Satisfiability

► **Theorem 9.** *The following problem is undecidable: Given a $\forall^2\exists^*$ HyperLTL¹(**F**, **G**) sentence φ , is φ satisfiable?*

Proof. We reduce from the (non)-halting problem for 2-counter Minsky machines. Recall that such a machine can be seen as a tuple $\mathcal{M} = (Q, \Delta, q_0)$ where Q is a finite set of states, $q_0 \in Q$ an initial state, and $\Delta \subseteq Q \times \{1, 2\} \times OP \times Q$ a set of transition rules, where $OP = \{++, --, =0?\}$. A configuration of \mathcal{M} is an element of $Q \times \mathbb{N} \times \mathbb{N}$. For all $n, n' \in \mathbb{N}$, $op \in OP$ we write $n \xrightarrow{op} n'$ if:

- op is $++$ and $n' = n + 1$.
- op is $--$ and $n' = n - 1$ (note that this operation is only applicable if $n > 0$).
- op is $=0?$ and $n' = n = 0$.

There is a transition from (q, n_1, n_2) to (q', n'_1, n'_2) if and only if there is an $i \in \{1, 2\}$ and $op \in OP$ with $(q, i, op, q') \in \Delta$, $n_{3-i} = n'_{3-i}$, and $n_i \xrightarrow{op} n'_i$. It is undecidable whether such a machine has an infinite computation $(q_0, 0, 0) \rightarrow (q_1, n_1^1, n_2^1) \rightarrow (q_2, n_1^2, n_2^2) \rightarrow \dots$.

Let $\mathcal{M} = (Q, \Delta, q_0)$ be a 2-counter Minsky machine. We use $AP = Q \cup \{1, 2\}$ as atomic propositions. Given $i \in \{1, 2\}$, we denote by \bar{i} the other proposition. Consider the formula $\psi_1 = \forall\pi. \forall\pi'. \mathbf{G}(1_\pi \Rightarrow 1_{\pi'}) \vee \mathbf{G}(1_{\pi'} \Rightarrow 1_\pi)$. We define ψ_2 with $2 \in AP$ analogously. In the following, we only consider sets of traces that satisfy $\psi_1 \wedge \psi_2$.

For each trace $t \in (2^{AP})^\omega$ and $i \in \{1, 2\}$, we define the i -set of t as $S_i(t) = \{j \in \mathbb{N} \mid i \in t(j)\}$. Now fix $T \subseteq (2^{AP})^\omega$ that satisfies $\psi_1 \wedge \psi_2$. We define the pre-order \leq_i on T as follows: for all $t, t' \in T$, $t \leq_i t'$ if and only if $S_i(t) \subseteq S_i(t')$. It is straightforward to verify that \leq_i is indeed reflexive and transitive. We write $t <_i t'$ if $S_i(t) \subsetneq S_i(t')$. As T satisfies $\psi_1 \wedge \psi_2$, the \leq_i are total pre-orders on T . We also define for all $t \in T$ and $i \in \{1, 2\}$, the *rank* of t with respect to i as $\text{rk}_i(t) = |\{S_i(t') \mid t' \in T \text{ and } t' <_i t\}|$, which may be infinite. Note that if $S_i(t) = \emptyset$ then $\text{rk}_i(t) = 0$, and that if $S_i(t) = S_i(t')$ then $\text{rk}_i(t) = \text{rk}_i(t')$. Also, note that the rank depends on the fixed set T of traces under consideration.

Finally, as \leq_i is a total pre-order, if we have $t <_i t'$, but there is no t'' with $t <_i t'' <_i t'$, then $\text{rk}_i(t') = \text{rk}_i(t) + 1$. Note that this holds even when $\text{rk}_i(t)$ is infinite, assuming $\infty + 1 = \infty$.

We construct a HyperLTL¹(**F**, **G**) formula φ that encodes the existence of an infinite computation $(q_0, 0, 0) \rightarrow (q_1, n_1^1, n_2^1) \rightarrow (q_2, n_1^2, n_2^2) \rightarrow \dots$ of \mathcal{M} . In a model T of φ , a configuration (q, n_1, n_2) is encoded by a trace t with $t(0) \cap Q = \{q\}$ and for $i \in \{1, 2\}$, $\text{rk}_i(t) = n_i$. Then, φ states the existence of an initial trace t_0 , representing the configuration $(q_0, 0, 0)$, as well as the existence of a successor t' encoding (q', n'_1, n'_2) for each trace t encoding (q, n_1, n_2) , i.e., we require $(q, n_1, n_2) \rightarrow (q', n'_1, n'_2)$. The latter is witnessed by the existence of a transition (q, i, op, q') such that:

1. $t(0) \cap Q = \{q\}$ and $t'(0) \cap Q = \{q'\}$, i.e., t and t' indeed encode the states of their respective configurations correctly.
2. For all $j \in \mathbb{N}$, $\bar{i} \in t(j)$ if and only if $\bar{i} \in t'(j)$, i.e. $S_{\bar{i}}(t) = S_{\bar{i}}(t')$. Thus, as argued above, $\text{rk}_{\bar{i}}(t) = \text{rk}_{\bar{i}}(t')$, which implies $n_{\bar{i}} = n'_{\bar{i}}$.
3. If op is $++$ then $t <_i t'$ and there does not exist any t'' such that $t <_i t'' <_i t'$, i.e., $\text{rk}_i(t') = \text{rk}_i(t) + 1$, as \leq_i is a total pre-order. Then, we have $n'_i = n_i + 1$.
4. If op is $--$ then $t >_i t'$ and there does not exist any t'' such that $t >_i t'' >_i t'$, i.e., $\text{rk}_i(t') = \text{rk}_i(t) - 1$, as \leq_i is a total pre-order. Then, we have $n'_i = n_i - 1$.
5. If op is $=0?$ then for all $j \in \mathbb{N}$, $i \notin t(j)$ and $i \notin t'(j)$. Hence, $S_i(t) = S_i(t') = \emptyset$, i.e., $\text{rk}_i(t) = \text{rk}_i(t') = 0$, which implies $n_i = n'_i = 0$.

We encode those conditions in φ , which is the conjunction of the following three sentences and of $\psi_1 \wedge \psi_2$:

- $\varphi_1 = \forall\pi. \bigwedge_{q \neq q' \in Q} q_\pi \Rightarrow \neg q'_\pi$ expresses that a trace is associated to at most one state.
- $\varphi_2 = \exists\pi_0. (q_0)_{\pi_0} \wedge \mathbf{G}(\neg 1_{\pi_0} \wedge \neg 2_{\pi_0})$ expresses the existence of a trace representing the initial configuration $(q_0, 0, 0)$.

- $\varphi_3 = \forall\pi.\exists\pi'.\bigvee_{(q,i,op,q')\in\Delta} q_\pi \wedge q'_{\pi'} \wedge \varphi_{i,op} \wedge \mathbf{G}(\bar{i}_\pi \Leftrightarrow \bar{i}_{\pi'})$ expresses that all traces have a successor obtained by faithfully simulating a transition of the machine.

Here, we use the formulas

- $\varphi_{1,++} = \forall\pi''.\pi <_1 \pi' \wedge (\pi'' \leq_1 \pi \vee \pi' \leq_1 \pi'')$,
- $\varphi_{1,--} = \forall\pi''.\pi >_1 \pi' \wedge (\pi'' \geq_1 \pi \vee \pi' \geq_1 \pi'')$, and
- $\varphi_{1,=0?} = \mathbf{G}(\neg 1_\pi \wedge \neg 1_{\pi'})$,

where $\pi \leq_1 \pi' = \mathbf{G}(1_\pi \Rightarrow 1_{\pi'})$ and $\pi <_1 \pi' = \pi \leq_1 \pi' \wedge \mathbf{F}(\neg 1_\pi \wedge 1_{\pi'})$. Finally, we define the formulas \leq_2 , $<_2$, and $\varphi_{2,op}$ analogously.

The sentence φ is not in prenex normal form. However, as no quantifier appears in the scope of a temporal operator, it can be put in that form. Further, it is not of the form $\forall^2\exists^*$, but we can apply Lemmas 7 and 8 to bring it into this form while preserving the temporal depth, which is already one. We claim that φ is satisfiable if and only if \mathcal{M} has an infinite computation starting in $(q_0, 0, 0)$.

Suppose φ is satisfied by a model T . The subformulas φ_1 and φ_2 enforce that T contains a trace t_0 encoding the initial configuration $(q_0, 0, 0)$ of \mathcal{M} . Further, φ_3 expresses that every trace t encoding a configuration (q, n_1, n_2) has a successor t' encoding a configuration (q', n'_1, n'_2) with $(q, n_1, n_2) \rightarrow (q', n'_1, n'_2)$. Thus, there exists an infinite sequence t_0, t_1, t_2, \dots of traces encoding an infinite run of \mathcal{M} .

Conversely, suppose \mathcal{M} has an infinite run $(q_0, 0, 0) \rightarrow (q_1, n_1^1, n_2^1) \rightarrow (q_2, n_1^2, n_2^2) \dots$, then for all j let t_j be the trace whose projection to Q is $\{q_j\}\emptyset^\omega$, and whose projection to $\{i\}$ is $\{i\}^{n_i^j}\emptyset^\omega$ for $i \in \{1, 2\}$. One can then easily check that $\{t_j \mid j \in \mathbb{N}\}$ is a model of φ . ◀

Thus, two universal quantifiers before some existential ones and using only \mathbf{F} and \mathbf{G} without nesting yields undecidable satisfiability. Our next result shows that removing one of the two universal quantifiers allows us to recover decidability, even when allowing nested next operators and leading existential quantifiers.

As a first step in the proof, we show that the nested next operators can be eliminated without introducing additional universal quantifiers. This is true, as we are only interested in satisfiability.

► **Lemma 10.** *For every $\exists^*\forall\exists^*$ HyperLTL¹($\mathbf{F}, \mathbf{G}, \mathbf{X}^*$) sentence, one can construct in polynomial time an equisatisfiable $\exists^*\forall\exists^*$ HyperLTL¹(\mathbf{F}, \mathbf{G}) sentence.*

Now, we are ready to prove our main decidability result in this section. Note that we do not claim a matching lower bound here. We comment on this gap in the conclusion.

► **Theorem 11.** *The following problem is in N2EXPTIME: Given a HyperLTL¹($\mathbf{F}, \mathbf{G}, \mathbf{X}^*$) sentence φ of the form $\exists^*\forall\exists^*$, is φ satisfiable?*

Proof. Let $\varphi = \exists\tau_1 \dots \tau_n.\forall\pi.\exists\tau_{n+1} \dots \exists\tau_{n+n'}.\psi$ be a HyperLTL¹($\mathbf{F}, \mathbf{G}, \mathbf{X}^*$) sentence with quantifier-free ψ . Due to Lemma 10, it is enough to consider the case where ψ is a Boolean combination of formulas of the form $\mathbf{F}\beta$ for a Boolean combination β of atomic propositions.

Note that such a formula can only specify the appearance or non-appearance of combinations of atomic propositions on the quantified traces, but not the order of these combinations. Hence, to every tuple (t_1, \dots, t_k) of traces $t_i \in (2^{\text{AP}})^\omega$, we associate a finite set of tuples of valuations $V(t_1, \dots, t_k) = \{(t_1(j), \dots, t_k(j)) \mid j \in \mathbb{N}\} \subseteq (2^{\text{AP}})^k$, i.e., the set all the tuples of valuations that appear eventually. The cardinality of $V(t_1, \dots, t_k)$ is at most $2^{k|\text{AP}|}$.

Let β be a Boolean combination of atomic propositions over trace variables π_1, \dots, π_k . Then, a trace assignment $[(\pi_i \rightarrow t_i)_{1 \leq i \leq k}]$ satisfies $\mathbf{F}\beta$ if and only if there exists $j \in \mathbb{N}$ such that β is satisfied at position j of (t_1, \dots, t_k) , i.e., there exists $(v_1, \dots, v_k) \in V(t_1, \dots, t_k)$ such

29:12 The Keys to Decidable HyperLTL Satisfiability

that (v_1, \dots, v_k) satisfies β (in the sense that any trace assignment Π such that $\Pi(\pi_i)(0) = v_i$ for all i satisfies β). Intuitively, we abstract a tuple of traces into a finite set of tuples of valuations, and then abstract a model as a set of such finite representations. Then, we show that satisfiability can be decided using such abstractions.

So, whether a given trace assignment $[(\pi_i \rightarrow t_i)_{1 \leq i \leq k}]$ satisfies a given Boolean combination ψ of formulas $\mathbf{F}\beta$ only depends on $V(t_1, \dots, t_k)$, and given $V \subseteq (2^{\text{AP}})^k$, one can check in polynomial time whether a trace assignment yielding V satisfies ψ . If it is the case, we say that V satisfies ψ .

To check the satisfiability of φ , we start by nondeterministically guessing a set $S \subseteq 2^{(2^{\text{AP}})^{n+n'+1}}$ of sets of $(n+n'+1)$ -tuples of valuations. This set is supposed to represent a model of φ . The n first valuations represent the fixed values assigned to τ_1, \dots, τ_n . The $(n+1)$ -th represents the valuation of the universally quantified variable. Thus, for every trace of the model there must exist a tuple in which that trace is represented at position $n+1$. The valuations of positions $n+2$ to $n+n'$ have to be such that φ is satisfied by all tuples.

Thus, we check the following requirements:

1. For all $V_1, V_2 \in S$, $\{(v_1, \dots, v_n) \mid (v_1, \dots, v_{n+n'+1}) \in V_1\}$ is equal to $\{(v_1, \dots, v_n) \mid (v_1, \dots, v_{n+n'+1}) \in V_2\}$: The set of values taken by the traces assigned to τ_1, \dots, τ_n cannot depend on the values of the other variables. Thus, we ensure that these values are fixed in the guessed model.
2. For all $V \in S$ and $1 \leq i \leq n+n'+1$, there exists $V' \in S$ such that $\{(v_1, \dots, v_n, v_i) \mid (v_1, \dots, v_{n+n'+1}) \in V\} = \{(v_1, \dots, v_{n+1}) \mid (v_1, \dots, v_{n+n'+1}) \in V'\}$. All the values taken by the existentially quantified variables have to be taken by the universally quantified one as well.
3. For all $V \in S$, V satisfies ψ .

If all requirements are satisfied, we accept, otherwise we reject. This procedure requires nondeterministic doubly-exponential time as $|S| \leq 2^{2^{|\text{AP}|+n+n'+1}}$.

Suppose φ is satisfiable and fix a model T . There exist $t_1, \dots, t_n \in T$ such that $(T, [(\tau_i \rightarrow t_i)_{1 \leq i \leq n}]) \models \forall \pi \exists \tau_{n+1} \dots \exists \tau_{n+n'} \psi$. Furthermore, for a fixed $t \in T$ there exist $t_{n+1}, \dots, t_{n+n'} \in T$ such that $(T, [(\tau_i \rightarrow t_i)_{1 \leq i \leq n+n'}, \pi \rightarrow t]) \models \psi$. Let $V^*(t) = \{(t_1(j), \dots, t_n(j), t(j), t_{n+1}(j), \dots, t_{n+n'}(j)) \mid j \in \mathbb{N}\}$.

Now, one can easily check that Requirements 1, 2, and 3 are satisfied by $\{V^*(t) \mid t \in T\}$. Thus, the algorithm accepts φ .

Conversely, suppose the algorithm accepts φ . Then, there exists some S satisfying all three requirements above. We construct from S a model T of φ .

Let t_1, \dots, t_n be traces such that for all $V \in S$, $\{(v_1, \dots, v_n) \mid (v_1, \dots, v_{n+n'+1}) \in V\} = V(t_1, \dots, t_n)$, and for all $(v_1, \dots, v_{n+n'+1}) \in V$, $(v_1, \dots, v_n) = (t_1(j), \dots, t_n(j))$ for infinitely many j , i.e., each of the valuations appears infinitely often in the traces. Those traces can be constructed due to Requirement 1.

Let $T_0 = \{t_1, \dots, t_n\}$. For all $\ell \in \mathbb{N}$ we construct T_ℓ by induction on $\ell \in \mathbb{N}$, while maintaining the following two invariants:

1. For all $t \in T_\ell$ there exists $V \in S$ such that $V(t_1, \dots, t_n, t)$ is equal to $\{(v_1, \dots, v_{n+1}) \mid (v_1, \dots, v_{n+n'+1}) \in V\}$, and for all $(v_1, \dots, v_{n+n'+1}) \in V$, (v_1, \dots, v_{n+1}) is equal to $(t_1(j), \dots, t_n(j), t(j))$ for infinitely many j , where the t_i are the traces in T_0 .
2. If $\ell > 0$ then for every $t \in T_{\ell-1}$, there exist traces $t_{n+1}, \dots, t_{n+n'} \in T_\ell$ such that $[(\tau_i \rightarrow t_i)_{1 \leq i \leq n+n'}, \pi \rightarrow t] \models \psi$.

By Requirement 2 and by construction, T_0 satisfies Invariant 1, and it clearly satisfies Invariant 2. Let $\ell \in \mathbb{N}$, suppose T_ℓ has been constructed, and that it satisfies Invariants 1 and 2. By Invariant 1, for all $t \in T_\ell$ we can construct traces $t_{n+1}, \dots, t_{n+n'}$ such that $V(t_1, \dots, t_n, t, t_{n+1}, \dots, t_{n+n'}) \in S$ and for all $(v_1, \dots, v_n, v, v_{n+1}, \dots, v_{n+n'}) \in V(t_1, \dots, t_n, t, t_{n+1}, \dots, t_{n+n'})$, it is the case that $(v_1, \dots, v_n, v, v_{n+1}, \dots, v_{n+n'})$ is equal to $(t_1(j), \dots, t_n(j), t(j), t_{n+1}(j), \dots, t_{n+n'}(j))$ for infinitely many j (as all the (v_1, \dots, v_n, v) appear infinitely many times in (t_1, \dots, t_n, t) by Invariant 1). Let $I(t) = \{t_{n+1}, \dots, t_{n+n'}\}$. Let $T_{\ell+1} = \bigcup_{t \in T_\ell} I(t)$, which satisfies Invariant 1 by Requirement 2. It also satisfies Invariant 2 by definition. Furthermore, by Requirement 3, $V(t_1, \dots, t_n, t, t_{n+1}, \dots, t_{n+n'})$ satisfies ψ .

Finally, let $T = \bigcup_{\ell \in \mathbb{N}} T_\ell$ and let $t \in T$. Then, there exists an ℓ such that $t \in T_\ell$. Thus, there also exist $t_{n+1}, \dots, t_{n+n'} \in T_{\ell+1}$ such that $[(\tau_i \rightarrow t_i)_{1 \leq i \leq n+n'}, \pi \rightarrow t]$ satisfies ψ . Therefore, T satisfies φ . \blacktriangleleft

Recall that satisfiability of $\exists^* \forall^*$ formulas is EXPSPACE-complete [8]. The proof of Finkbeiner and Hahn can be slightly adapted to produce a formula of temporal depth two: their approach states the existence of a trace representing a sequence of configurations of an exponential-space bounded Turing machine. The only difficulty that can arise in expressing the correctness of the run described by that trace is relating a position of one of the configurations to the neighbouring positions in the next configuration (in order to simulate the movement of the head). One may then require to combine an until and a next in order to express this requirement, in the scope of an always expressing that it holds for every position. This nesting can be removed by adding a fresh proposition p that is satisfied on all positions of the first configuration, on none of the second one, and so on, i.e., its truth value alternates between the configurations. One can then express the previous requirement with a single until in the scope of an always, yielding temporal depth two.

Our next result shows that one obtains better complexity when restricting the temporal depth of formulas to one.

► **Theorem 12.** *The following problem is NEXPTIME-complete: Given an $\exists^* \forall^*$ HyperLTL sentence φ with temporal depth one, is φ satisfiable?*

We adapt the proof of Finkbeiner and Hahn for EXPSPACE-completeness of the problem with arbitrary temporal depth [8], i.e., we turn the HyperLTL formula into an exponentially larger equisatisfiable LTL one (cp. the proof of Theorem 1). The decrease in complexity is a consequence of the switch from PSPACE to NP of the complexity of LTL satisfiability when restricting temporal depth to one [7].

We conclude by considering the satisfiability problem for $\text{HyperLTL}^1(\mathbf{F}, \mathbf{G})$ with arbitrary quantifier prefixes, but restricted to models induced by finite-state systems. The undecidability of satisfiability for arbitrary formulas over finite-state systems can be easily inferred from the proof of undecidability of satisfiability of Finkbeiner and Hahn, as the formulas they construct, if satisfiable, have a finite and ultimately periodic model, which is therefore representable by a finite-state system. For formulas of $\text{HyperLTL}^1(\mathbf{F}, \mathbf{G})$, we leave decidability open, but prove intractability.

► **Theorem 13.** *The following problem is TOWER-hard: Given a $\text{HyperLTL}^1(\mathbf{F}, \mathbf{G})$ sentence φ , does φ have a model $T(\mathcal{K})$ for some Kripke structure \mathcal{K} ?*

Let us conclude by remarking that the satisfiability problem for $\text{HyperLTL}^1(\mathbf{F}, \mathbf{G})$ over Kripke structures is different from the general one, i.e., there are satisfiable formulas which are not satisfied by the set of traces of any Kripke structure. Consider for instance the sentence $\varphi = \forall \pi. \exists \pi'. \mathbf{G}(a_\pi \Rightarrow a_{\pi'}) \wedge \mathbf{F}(\neg a_\pi \wedge a_{\pi'})$, which is satisfied by $\{a\}^* \emptyset^\omega$.

Suppose there exists a Kripke structure \mathcal{K} with a set of traces satisfying this sentence. We define inductively an increasing sequence of finite trace prefixes p_n for $n \in \mathbb{N}$ as $p_0 = \varepsilon$ and $p_{n+1} = p_n\{a\}$ if $p_n\{a\}$ is a prefix of a trace of \mathcal{K} , and $p_{n+1} = p_n\emptyset$ otherwise. Let t be the limit of the sequence $(p_n)_{n \in \mathbb{N}}$, i.e., the unique trace with prefix p_n for every n . As the p_n are prefixes of traces of \mathcal{K} , t itself is a trace of \mathcal{K} . As \mathcal{K} satisfies φ , there exists t' such that for all j , if $a \in t(j)$ then $a \in t'(j)$ and there exists j^* such that $a \in t'(j^*)$ and $a \notin t(j^*)$. In particular, there exists a minimal such j^* . Then $p_{j^*+1} = p_{j^*}\emptyset$, but $p_{j^*}\{a\}$ is a prefix of t' . This contradicts the choice of p_{j^*+1} , as we prefer to extend by $\{a\}$ instead of \emptyset . Thus, the satisfiable sentence φ is not satisfiable by the set of traces of a finite Kripke structure.

5 Conclusion

We have shown that HyperLTL satisfiability can be decidable, either if one restricts the space of models one is interested in to sufficiently simple ones, or if one restricts the alternation and temporal depth of the formulas under consideration. In particular, we have investigated the formulas of temporal depth one without untils. An interesting open problem is to extend the decidability result presented in Theorem 11 to formulas with untils. Also, we claimed no lower bound on the problem solved in Theorem 11. We claim there is an EXPSpace lower bound obtained by encoding exponential space Turing machines, but the exact complexity of the problem is left open. Another interesting problem left open is the decidability of $\text{HyperLTL}^1(\mathbf{F}, \mathbf{G})$ over Kripke structures. We have presented a TOWER lower bound in Theorem 13, but it is open whether the problem is indeed decidable.

In general, restricting the space of models turns out to be more fruitful than to restrict the formulas under consideration, as satisfiability is undecidable for extremely simple formulas (simplicity being measured in alternation depth and temporal depth). An interesting challenge pertains to finding other measures of simplicity that yield larger decidable fragments.

References

- 1 Shreya Agrawal and Borzoo Bonakdarpour. Runtime Verification of k-Safety Hyperproperties in HyperLTL. In *CSF 2016*, pages 239–252. IEEE Computer Society, 2016. doi:10.1109/CSF.2016.24.
- 2 Borzoo Bonakdarpour and Bernd Finkbeiner. Runtime Verification for HyperLTL. In Yliès Falcone and César Sánchez, editors, *RV 2016*, volume 10012 of *LNCS*, pages 41–45. Springer, 2016. doi:10.1007/978-3-319-46982-9_4.
- 3 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal Logics for Hyperproperties. In Martín Abadi and Steve Kremer, editors, *POST 2014*, volume 8414 of *LNCS*, pages 265–284. Springer, 2014. doi:10.1007/978-3-642-54792-8_15.
- 4 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.
- 5 Norine Coenen, Bernd Finkbeiner, Christopher Hahn, and Jana Hofmann. The hierarchy of hyperlogics. In *LICS 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785713.
- 6 Norine Coenen, Bernd Finkbeiner, César Sánchez, and Leander Tentrup. Verifying hyperliveness. In Isil Dillig and Serdar Tasiran, editors, *CAV 2019*, volume 11561 of *LNCS*, pages 121–139. Springer, 2019. doi:10.1007/978-3-030-25540-4_7.
- 7 Stéphane Demri and Philippe Schnoebelen. The Complexity of Propositional Linear Temporal Logics in Simple Cases. *Inf. Comput.*, 174(1):84–103, 2002. doi:10.1006/inco.2001.3094.
- 8 Bernd Finkbeiner and Christopher Hahn. Deciding Hyperproperties. In Josée Desharnais and Radha Jagadeesan, editors, *CONCUR 2016*, volume 59 of *LIPICs*, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.CONCUR.2016.13.

- 9 Bernd Finkbeiner, Christopher Hahn, and Tobias Hans. MGHyper: Checking Satisfiability of HyperLTL Formulas Beyond the $\exists^*\forall^*$ Fragment. In Shuvendu K. Lahiri and Chao Wang, editors, *ATVA 2018*, volume 11138 of *LNCS*, pages 521–527. Springer, 2018. doi:10.1007/978-3-030-01090-4_31.
- 10 Bernd Finkbeiner, Christopher Hahn, Philip Lukert, Marvin Stenger, and Leander Tentrup. Synthesizing Reactive Systems from Hyperproperties. In Hana Chockler and Georg Weissenbacher, editors, *CAV 2018 (Part I)*, volume 10981 of *LNCS*, pages 289–306. Springer, 2018. doi:10.1007/978-3-319-96145-3_16.
- 11 Bernd Finkbeiner, Christopher Hahn, and Marvin Stenger. EAHyper: Satisfiability, implication, and equivalence checking of hyperproperties. In Rupak Majumdar and Viktor Kuncak, editors, *CAV 2017 (Part II)*, volume 10427 of *LNCS*, pages 564–570. Springer, 2017. doi:10.1007/978-3-319-63390-9_29.
- 12 Bernd Finkbeiner, Christopher Hahn, Marvin Stenger, and Leander Tentrup. RVHyper: A runtime verification tool for temporal hyperproperties. In Dirk Beyer and Marieke Huisman, editors, *TACAS 2018 (Part II)*, volume 10806 of *LNCS*, pages 194–200. Springer, 2018. doi:10.1007/978-3-319-89963-3_11.
- 13 Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for Model Checking HyperLTL and HyperCTL*. In Daniel Kroening and Corina S. Pasareanu, editors, *CAV 2015 (Part I)*, volume 9206 of *LNCS*, pages 30–48. Springer, 2015. doi:10.1007/978-3-319-21690-4_3.
- 14 Bernd Finkbeiner and Martin Zimmermann. The First-Order Logic of Hyperproperties. In Heribert Vollmer and Brigitte Vallée, editors, *STACS 2017*, volume 66 of *LIPICs*, pages 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.STACS.2017.30.
- 15 Joseph A. Goguen and José Meseguer. Security Policies and Security Models. In *1982 IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society, 1982. doi:10.1109/SP.1982.10014.
- 16 Joseph Y. Halpern, Robert Harper, Neil Immerman, Phokion G. Kolaitis, Moshe Y. Vardi, and Victor Vianu. On the unusual effectiveness of logic in computer science. *Bulletin of Symbolic Logic*, 7(2):213–236, 2001. doi:10.2307/2687775.
- 17 Philip K. Hooper. The undecidability of the Turing machine immortality problem. *J. Symb. Log.*, 31(2):219–234, 1966. doi:10.2307/2269811.
- 18 Robert P. Kurshan. Transfer of Model Checking to Industrial Practice. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking.*, pages 763–793. Springer, 2018. doi:10.1007/978-3-319-10575-8_23.
- 19 Daryl McCullough. Noninterference and the composability of security properties. In *1988 IEEE Symposium on Security and Privacy*, pages 177–186. IEEE Computer Society, 1988. doi:10.1109/SECPRI.1988.8110.
- 20 Daryl McCullough. A Hookup Theorem for Multilevel Security. *IEEE Trans. Software Eng.*, 16(6):563–568, 1990. doi:10.1109/32.55085.
- 21 John McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *1994 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 79–93. IEEE Computer Society, 1994. doi:10.1109/RISP.1994.296590.
- 22 Jonathan K. Millen. Unwinding Forward Correctability. *Journal of Computer Security*, 3(1):35–54, 1995. doi:10.3233/JCS-1994/1995-3104.
- 23 Amir Pnueli. The Temporal Logic of Programs. In *FOCS 1977*, pages 46–57, 1977.
- 24 Sylvain Schmitz. Complexity Hierarchies beyond Elementary. *TOCT*, 8(1):3:1–3:36, 2016. doi:10.1145/2858784.
- 25 A. Prasad Sistla and Edmund M. Clarke. The Complexity of Propositional Linear Temporal Logics. *Journal of the ACM*, 32(3):733–749, 1985. doi:10.1145/3828.3837.
- 26 Larry J. Stockmeyer and Albert R. Meyer. Word Problems Requiring Exponential Time: Preliminary Report. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd,

29:16 The Keys to Decidable HyperLTL Satisfiability

Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *STOC 1973*, pages 1–9. ACM, 1973. doi:10.1145/800125.804029.

- 27 Steve Zdancewic and Andrew C. Myers. Observational Determinism for Concurrent Program Security. In *CSFW 2003*, page 29. IEEE Computer Society, 2003. doi:10.1109/CSFW.2003.1212703.

Revisiting the Duality of Computation: An Algebraic Analysis of Classical Realizability Models

Étienne Miquey

Équipe Gallinette, INRIA

LS2N, Université de Nantes, France

etienne.miquey@inria.fr

Abstract

In an impressive series of papers, Krivine showed at the edge of the last decade how classical realizability provides a surprising technique to build models for classical theories. In particular, he proved that classical realizability subsumes Cohen’s forcing, and even more, gives rise to unexpected models of set theories. Pursuing the algebraic analysis of these models that was first undertaken by Streicher, Miquel recently proposed to lay the algebraic foundation of classical realizability and forcing within new structures which he called *implicative algebras*. These structures are a generalization of Boolean algebras based on an internal law representing the implication. Notably, implicative algebras allow for the adequate interpretation of both programs (i.e. proofs) and their types (i.e. formulas) in the same structure.

The very definition of implicative algebras takes position on a presentation of logic through universal quantification and the implication and, computationally, relies on the call-by-name λ -calculus. In this paper, we investigate the relevance of this choice, by introducing two similar structures. On the one hand, we define *disjunctive algebras*, which rely on internal laws for the negation and the disjunction and which we show to be particular cases of implicative algebras. On the other hand, we introduce *conjunctive algebras*, which rather put the focus on conjunctions and on the call-by-value evaluation strategy. We finally show how disjunctive and conjunctive algebras algebraically reflect the well-known duality of computation between call-by-name and call-by-value.

2012 ACM Subject Classification Theory of computation \rightarrow Logic; Theory of computation \rightarrow Proof theory; Theory of computation \rightarrow Type theory

Keywords and phrases realizability, model theory, forcing, proofs-as-programs, λ -calculus, classical logic, duality, call-by-value, call-by-name, lattices, tripos

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.30

Related Version An extended version of this paper including proofs and further details is available at: <https://hal.archives-ouvertes.fr/hal-02305560>.

Funding This research was partially funded by the ANII research project FCE_1_2014_1_104800.

Acknowledgements The author would like to thank Alexandre Miquel to which several ideas in this paper, especially the definition of conjunctive separators, should be credited.

1 Introduction

It is well-known since Griffin’s seminal work [13] that a classical Curry-Howard correspondence can be obtained by adding control operators to the λ -calculus. Several calculi were born from this idea, amongst which Krivine λ_c -calculus [20], defined as the λ -calculus extended with Scheme’s `call/cc` operator (for *call-with-current-continuation*). Elaborating on this calculus, Krivine’s developed in the late 90s the theory of *classical realizability* [20], which is a complete reformulation of its intuitionistic twin. Originally introduced to analyze the computational content of classical programs, it turned out that classical realizability also provides interesting semantics for classical theories. While it was first tailored to Peano second-order arithmetic (i.e. second-order type systems), classical realizability actually scales



© Étienne Miquey;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 30; pp. 30:1–30:18

Leibniz International Proceedings in Informatics



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

to more complex classical theories like ZF [21], and gives rise to surprisingly new models. In particular, it generalizes Cohen’s forcing [21, 30] and allows for the direct definition of a model in which neither the continuum hypothesis nor the axiom of choice holds [23].

Algebraization of classical realizability. During the last decade, the algebraic structure of the models that classical realizability induces has been actively studied. This line of work was first initiated by Streicher, who proposed the concept of *abstract Krivine structure* [38], followed among others by Ferrer, Frey, Guillermo, Malherbe and Miquel who introduced other structures peculiar to classical realizability [8, 9, 6, 10, 11, 40]. In addition to the algebraic study of classical realizability models, these works had the interest of building the bridge with the algebraic structures arising from intuitionistic realizability. In particular, Streicher showed in [38] how classical realizability could be analyzed in terms of *triposes* [37], the categorical framework emerging from intuitionistic realizability models, while the later work of Ferrer et al. [8, 9] connected it to Hofstra and Van Oosten’s notion of *ordered combinatory algebras* [16]. More recently, Alexandre Miquel introduced the concept of *implicative algebra* [31], which appear to encompass the previous approaches and which we present in this paper.

Implicative algebras. In addition to providing an algebraic framework conducive to the analysis of classical realizability, an important feature of implicative structures is that they allow us to identify *realizers* (i.e. λ -terms) and *truth values* (i.e. formulas). Concretely, implicative structures are complete lattices equipped with a binary operation $a \rightarrow b$ satisfying properties coming from the logical implication. As we will see, they indeed allow us to interpret both the formulas and the terms in the same structure. For instance, the ordering relation $a \preceq b$ will encompass different intuitions depending on whether we regard a and b as formulas or as terms. Namely, $a \preceq b$ will be given the following meanings:

- the formula a is a *subtype* of the formula b ;
- the term a is a *realizer* of the formula b ;
- the realizer a is *more defined* than the realizer b .

In terms of the Curry-Howard correspondence, this means that we not only identify types with formulas and proofs with programs, but *we also identify types and programs*.

Side effects. Following Griffin’s discovery on control operators and classical logic, several works have renewed the observation that within the proofs-as-programs correspondence, with side effects come new reasoning principles [19, 18, 29, 14, 17]. More generally, it is now clear that computational features of a calculus may have consequences on the models it induces. For instance, computational proofs of the axiom of dependent choice can be obtained by adding a `quote` instruction [19], using memoisation [15, 33] or with a bar recursor [25]. Yet, such choices may also have an impact on the structures of the corresponding realizability models: the non-deterministic operator \heartsuit is known to make the model collapse on a forcing situation [22], while the bar recursor requires some continuity properties [25].

If we start to have a deep understanding of the algebraic structure of classical realizability models, the algebraic counterpart of side effects on these structures is still unclear. As a first step towards this problem, it is natural to wonder: does the choice of an evaluation strategy have algebraic consequences on realizability models? This paper aims at bringing new tools for addressing this question.

Outline of the paper. We start by recalling the definition of Miquel’s implicative algebras and their main properties in Section 2. We then introduce the notion of *disjunctive algebras*

in Section 3, which naturally arises from the negative decomposition of the implication $A \rightarrow B = \neg A \wp B$. We explain how this decomposition induces realizability models based on a call-by-name fragment of Munch-Maccagnoni’s system L [35], and we show that disjunctive algebras are in fact particular cases of implicative algebras. In Section 4, we explore the positive dual decomposition $A \rightarrow B = \neg(A \otimes \neg B)$, which naturally corresponds to a call-by-value fragment of system L. We show the corresponding realizability models naturally induce a notion of *conjunctive algebras*. Finally, in Section 5 we revisit the well-known duality of computation through this algebraic structures. In particular, we show how to pass from conjunctive to disjunctive algebras and vice-versa, while inducing isomorphic triposes.

Most of the proofs have been formalized in the Coq proof assistant, in which case their statements include hyperlinks to their formalizations¹.

2 Implicative algebras

2.1 Krivine classical realizability in a glimpse

We give here an overview of the main characteristics of Krivine realizability and of the models it induces². Krivine realizability models are usually built above the λ_c -calculus, a language of abstract machines including a set of terms Λ and a set of stacks Π (i.e. evaluation contexts). Processes $t \star \pi$ in the abstract machine are given as pairs of a term t and a stack π .

Krivine realizability interprets a formula A as a set of closed terms $|A| \subseteq \Lambda$, called the *truth value* of A , and whose elements are called the *realizers* of A . Unlike in intuitionistic realizability models, this set is actually defined by orthogonality to a *falsity value* $\|A\|$ made of stacks, which intuitively represents a set of opponents to the formula A . Realizability models are parameterized by a pole $\perp\!\!\!\perp$, a set of processes in the underlying abstract machine which somehow plays the role of a referee between terms and stacks. The pole allows us to define the orthogonal set X^\perp of any falsity value $X \subseteq \Pi$ by: $X^\perp \triangleq \{t \in \Lambda : \forall \pi \in X, t \star \pi \in \perp\!\!\!\perp\}$. Valid formulas A are then defined as the ones admitting a proof-like *realizer*³ $t \in |A|$.

Before defining implicative algebras, we would like to draw the reader’s attention on an important observation about realizability: there is an omnipresent lattice structure, which is reminiscent of the concept of subtyping [3]. Given a realizability model it is indeed always possible to define a semantic notion of subtyping: $A \preceq B \triangleq \|B\| \subseteq \|A\|$. This informally reads as “ A is more precise than B ”, in that A admits more opponents than B . In this case, the relation \preceq being induced from (reversed) set inclusions comes with a richer structure of complete lattice, where the meet \wedge is defined as a union and the join \vee as an intersection. In particular, the interpretation of a universal quantifier $\|\forall x.A\|$ is given by an union $\bigcup_{n \in \mathbb{N}} \|A[n/x]\| = \bigwedge_{n \in \mathbb{N}} \|A[n/x]\|$, while the logical connective \wedge is interpreted as the type of pairs \times i.e. with a computation content. As such, *realizability* corresponds to the following picture: $\forall = \bigwedge \quad \wedge = \times$. This is to compare with *forcing*, that can be expressed in terms of Boolean algebras where both the universal quantifier and the conjunction are interpreted by meets without any computational content: $\forall = \wedge = \bigwedge$ [1].

¹ Available at <https://gitlab.com/emiquey/ImplicativeAlgebras/>

² For a detailed introduction on this topic, we refer the reader to [20] or [32].

³ One specificity of Krivine classical realizability is that the set of terms contains the control operator \mathbf{cc} and continuation constants \mathbf{k}_π . Therefore, to preserve the consistency of the induced models, one has to consider only proof-like terms, i.e. terms that do not contain any continuations constants see [20, 32].

2.2 Implicative algebras

Implicative structures are tailored to represent both the formulas of second-order logic and realizers arising from Krivine's λ_c -calculus. For their logical facet, they are defined as meet-complete lattices (for the universal quantification) with an internal binary operation satisfying the properties of the implication:

► **Definition 1.** *An implicative structure is a complete lattice (\mathcal{A}, \preceq) equipped with an operation $(a, b) \mapsto (a \rightarrow b)$, such that for all $a, a_0, b, b_0 \in \mathcal{A}$ and any subset $B \subseteq \mathcal{A}$:*

1. *If $a_0 \preceq a$ and $b \preceq b_0$ then $(a \rightarrow b) \preceq (a_0 \rightarrow b_0)$.*
2. $\bigwedge_{b \in B} (a \rightarrow b) = a \rightarrow \bigwedge_{b \in B} b$

It is then immediate to embed any closed formula of second-order logic within any implicative structure. Obviously, any complete Heyting algebra or any complete Boolean algebra defines an implicative structure with the canonical arrow. More interestingly, any ordered combinatory algebras, a structure arising naturally from realizability [16, 39, 38, 7], also induces an implicative structure [34]. Last but not least, any classical realizability model induces as expected an implicative structure on the lattice $(\mathcal{P}(\Pi), \supseteq)$ by considering the arrow defined by⁴: $a \rightarrow b \triangleq a^\perp \cdot b = \{t \cdot \pi : t \in a^\perp, \pi \in b\}$ ([31, 34]).

Interestingly, if any implicative structure \mathcal{A} trivially provides us with an embedding of second-order formulas, we can also encode λ -terms with the following definitions:

$$ab \triangleq \bigwedge \{c : a \preceq b \rightarrow c\} \qquad \lambda f \triangleq \bigwedge_{a \in \mathcal{A}} (a \rightarrow f(a))$$

In both cases, one can understand the meet as a conjunction of all the possible approximations of the desired term. From now on, we will denote by t^A (resp. A^A) the interpretation of the closed λ -term t (resp. formula A). Notably, these embeddings are at the same time:

1. Sound with respect to the β -reduction, in the sense that $(\lambda f)a \preceq f(a)$ (and more generally, one can show that if $t \rightarrow_\beta u$ implies $t^A \preceq u^A$);
2. Adequate with respect to typing, in the sense that if t is of type A , then we have $t^A \preceq A^A$ (which can read as “ t realizes A ”).

In the case of certain combinators, including Hilbert's combinator \mathbf{k} and \mathbf{s} , their interpretations as λ -term is even equal to the interpretation of their principal types, that is to say that we have $\mathbf{k}^A = \bigwedge_{a, b \in \mathcal{A}} (a \rightarrow b \rightarrow a)$ and $\mathbf{s}^A = \bigwedge_{a, b, c \in \mathcal{A}} ((a \rightarrow b \rightarrow c) \rightarrow (a \rightarrow b) \rightarrow a \rightarrow c)$. This justifies the definition $\mathbf{cc}^A \triangleq \bigwedge_{a, b} (((a \rightarrow b) \rightarrow a) \rightarrow a)$.

Implicative structure are thus suited to interpret both terms and their types. To give an account for realizability models, one then has to define a notion of validity:

► **Definition 2 (Separator).** *Let $(\mathcal{A}, \preceq, \rightarrow)$ be an implicative structure. We call a separator over \mathcal{A} any set $\mathcal{S} \subseteq \mathcal{A}$ such that for all $a, b \in \mathcal{A}$, the following conditions hold:*

1. *If $a \in \mathcal{S}$ and $a \preceq b$, then $b \in \mathcal{S}$.*
2. $\mathbf{k}^A \in \mathcal{S}$, and $\mathbf{s}^A \in \mathcal{S}$.
3. *If $(a \rightarrow b) \in \mathcal{S}$ and $a \in \mathcal{S}$, then $b \in \mathcal{S}$.*

A separator \mathcal{S} is said to be classical if $\mathbf{cc}^A \in \mathcal{S}$ and consistent if $\perp \notin \mathcal{S}$. We call implicative algebra any implicative structure $(\mathcal{A}, \preceq, \rightarrow, \mathcal{S})$ equipped with a separator \mathcal{S} over \mathcal{A} .

Intuitively, thinking of elements of an implicative structure as truth values, a separator should be understood as the set which distinguishes the valid formulas (think of a filter in a

⁴ This is actually nothing more than the definition of the falsity value $\|A \Rightarrow B\|$.

Boolean algebra). Considering the elements as terms, it should rather be viewed as the set of valid realizers. Indeed, conditions (2) and (3) ensure that all closed λ -terms are in any separator⁵. Reading $a \preceq b$ as “the formula a is a subtype of the formula b ”, condition (2) ensures the validity of semantic subtyping. Thinking of the ordering as “ a is a realizer of the formula b ”, condition (3) states that if a formula is realized, then it is in the separator.

► **Example 3.** Any Krivine realizability model induces an implicative structure $(\mathcal{A}, \preceq, \rightarrow)$ where $\mathcal{A} = \mathcal{P}(\Pi)$, $a \preceq b \Leftrightarrow a \supseteq b$ and $a \rightarrow b = a^\perp \cdot b$. The set of realized formulas, namely $\mathcal{S} = \{a \in \mathcal{A} : \exists t \in a^\perp, t \text{ proof-like}\}$, defines a valid separator [31].

2.3 Internal logic & implicative tripos

In order to study the internal logic of implicative algebras, we define an *entailment* relation: we say that a entails b and we write $a \vdash_{\mathcal{S}} b$ if $a \rightarrow b \in \mathcal{S}$. This relation induces a preorder on \mathcal{A} . Then, by defining products $a \times b$ and sums $a + b$ through their usual impredicative encodings in System F^6 , we recover a structure of pre-Heyting algebra with respect to the entailment relation: $a \vdash_{\mathcal{S}} b \rightarrow c$ if and only if $a \times b \vdash_{\mathcal{S}} c$.

In order to recover a Heyting algebra, it suffices to consider the quotient $\mathcal{H} = \mathcal{A}/\cong_{\mathcal{S}}$ by the equivalence relation $\cong_{\mathcal{S}}$ induced by $\vdash_{\mathcal{S}}$, which is naturally equipped with an order relation: $[a] \preceq_{\mathcal{H}} [b] \triangleq a \vdash_{\mathcal{S}} b$ (where we write $[a]$ for the equivalence class of $a \in \mathcal{A}$). Likewise, we can extend the product, the sum and the arrow to equivalence classes to obtain a Heyting algebra $(\mathcal{H}, \preceq_{\mathcal{H}}, \wedge_{\mathcal{H}}, \vee_{\mathcal{H}}, \rightarrow_{\mathcal{H}})$.

Given any implicative algebra, we can define construction of the implicative tripos is quite similar. Recall that a (set-based) *tripos* is a first-order hyperdoctrine $\mathcal{T} : \mathbf{Set}^{op} \rightarrow \mathbf{HA}$ which admits a generic predicate. To define a tripos, we roughly consider the functor of the form $I \in \mathbf{Set}^{op} \mapsto \mathcal{A}^I$. Again, to recover a Heyting algebra we quotient the product \mathcal{A}^I (which defines an implicative structure) by the *uniform separator* $\mathcal{S}[I]$ defined by:

$$\mathcal{S}[I] \triangleq \{a \in \mathcal{A}^I : \exists s \in S. \forall i \in I. s \preceq a_i\}$$

► **Theorem 4** (Implicative tripos [31]). Let $(\mathcal{A}, \preceq, \rightarrow, \mathcal{S})$ be an implicative algebra. The following functor (where $f : J \rightarrow I$) defines a tripos:

$$\mathcal{T} : I \mapsto \mathcal{A}^I / \mathcal{S}[I] \qquad \mathcal{T}(f) : \begin{cases} \mathcal{A}^I / \mathcal{S}[I] & \rightarrow & \mathcal{A}^J / \mathcal{S}[J] \\ [(a_i)_{i \in I}] & \mapsto & [(a_{f(j)})_{j \in J}] \end{cases}$$

Observe that we could also quotient the product \mathcal{A}^I by the separator product \mathcal{S}^I . Actually, the quotient $\mathcal{A}^I / \mathcal{S}^I$ is in bijection with $(\mathcal{A}/\mathcal{S})^{\mathcal{I}}$, and in the case where \mathcal{S} is a classical separator, \mathcal{A}/\mathcal{S} is actually a Boolean algebra, so that the product $(\mathcal{A}/\mathcal{S})^{\mathcal{I}}$ is nothing more than a Boolean-valued model (as in the case of forcing). Since $\mathcal{S}[I] \subseteq \mathcal{S}^I$, the realizability models that can not be obtained by forcing are exactly those for which $\mathcal{S}[I] \neq \mathcal{S}^I$ (see [31]).

3 Decomposing the arrow: disjunctive algebras

We shall now introduce the notion of disjunctive algebra, which is a structure primarily based on disjunctions, negations (for the connectives) and meets (for the universal quantifier). Our main purpose is to draw the comparison with implicative algebras, as an attempt to

⁵ The latter indeed implies the closure of separators under application.

⁶ That is to say that we define $a \times b \triangleq \lambda_{c \in \mathcal{A}}((a \rightarrow b \rightarrow c) \rightarrow c)$ and $a + b \triangleq \lambda_{c \in \mathcal{A}}((a \rightarrow c) \rightarrow (b \rightarrow c) \rightarrow c)$.

justify eventually that the latter are more general than the former, and to lay the bases for a dualizable definition. In the seminal paper introducing linear logic [12], Girard refines the structure of the sequent calculus LK, introducing in particular negative and positive connectives for disjunctions and conjunctions⁷. With this finer set of connectives, the usual implication can be retrieved using either the negative disjunction: $A \rightarrow B \triangleq \neg A \wp B$ or the positive conjunction: $A \rightarrow B \triangleq \neg(A \otimes \neg B)$.

In 2009, Munch-Maccagnoni gave a computational account of Girard’s presentation for classical logic [35]. In his calculus, named L, each connective corresponds to the type of a particular constructor (or destructor). While L is in essence close to Curien and Herbelin’s $\lambda\mu\tilde{\mu}$ -calculus [4] (in particular it is presented with the same paradigm of duality between proofs and contexts), the syntax of terms does not include λ -abstraction (and neither does the syntax of formulas includes an implication). The two decompositions of the arrow evoked above are precisely reflected in decompositions of λ -abstractions (and dually, of stacks) in terms of L constructors. Notably, the choice of a decomposition corresponds to a particular choice of an evaluation strategy⁸ for the encoded λ -calculus: picking the negative \wp connective corresponds to call-by-name, while the decomposition using the \otimes connective reduces in a call-by-value fashion.

We shall begin by considering the call-by-name case, which is closer to the situation of implicative algebras. The definition of disjunctive structures and algebras are guided by an analysis of the realizability model induced by L^{\wp} , that is Munch-Maccagnoni’s system L restricted to the fragment corresponding to negative formulas: $A, B := X \mid A \wp B \mid \neg A \mid \forall X.A$ [35]. To leave room for more details on disjunctive algebras, we elude here the introduction of L^{\wp} and its relation to the call-by-name λ -calculus, we refer the interested reader to the extended version.

3.1 Disjunctive structures

We are now going to define the notion of *disjunctive structure*. Since we choose negative connectives and in particular a universal quantifier, we should define commutations with respect to arbitrary meets. The realizability interpretation for L^{\wp} provides us with a safeguard in this regard, since in the corresponding models, if $X \notin FV(B)$ the following equalities⁹ hold:

1. $\|\forall X.(A \wp B)\|_V = \|(\forall X.A) \wp B\|_V$
2. $\|\forall X.(B \wp A)\|_V = \|B \wp (\forall X.A)\|_V$
3. $\|\neg(\forall X.A)\|_V = \bigcap_{S \in \mathcal{P}(\mathcal{V}_0)} \|\neg A\{X := \dot{S}\}\|_V$

Algebraically, the previous proposition advocates for the following definition (remember that the order is defined as the reversed inclusion of primitive falsity values (whence \cap is Υ) and that the \forall quantifier is interpreted by \wedge):

► **Definition 5** (Disjunctive structure). *A disjunctive structure is a complete lattice (\mathcal{A}, \preceq) equipped with a binary operation $(a, b) \mapsto a \wp b$, together with a unary operation $a \mapsto \neg a$, such that for all $a, a', b, b' \in \mathcal{A}$ and for any $B \subseteq \mathcal{A}$:*

1. if $a \preceq a'$ then $\neg a' \preceq \neg a$
2. if $a \preceq a'$ and $b \preceq b'$ then $a \wp b \preceq a' \wp b'$
3. $\wedge_{b \in B} (b \wp a) = (\wedge_{b \in B} b) \wp a$
4. $\wedge_{b \in B} (a \wp b) = a \wp (\wedge_{b \in B} b)$
5. $\neg \wedge_{a \in A} a = \Upsilon_{a \in A} \neg a$

⁷ We insist on the fact that even though we use linear notations afterwards, nothing will be linear here.

⁸ Phrased differently, this observation can be traced back to different works, for instance by Blain-Levy [28, Fig. 5.10], Laurent [26] or Danos, Joinet and Schellinx [5].

⁹ Technically, \mathcal{V}_0 is the set of closed values which, in this setting, are evaluation contexts (think of Π in usual Krivine models), and $\|A\|_V \in \mathcal{P}(\mathcal{V}_0)$ is the (ground) falsity value of a formula A .

Observe that the commutation laws imply the value of the internal laws when applied to the maximal element \top : **1.** $\top \wp a = \top$ **2.** $a \wp \top = \top$ **3.** $\neg\top = \perp$

We give here some examples of disjunctive structures.

► **Example 6** (Dummy disjunctive structure). *Given any complete lattice (\mathcal{L}, \preceq) , defining $a \wp b \triangleq \top$ and $\neg a \triangleq \perp$ gives rise to a dummy structure that fulfills the required properties.*

► **Example 7** (Complete Boolean algebras). *Let \mathcal{B} be a complete Boolean algebra. It encompasses a disjunctive structure defined by:*

$$\blacksquare \mathcal{A} \triangleq \mathcal{B} \qquad \blacksquare a \preceq b \triangleq a \preceq b \qquad \blacksquare a \wp b \triangleq a \vee b \qquad \blacksquare \neg a \triangleq \neg a$$

► **Example 8** (L^\wp realizability models). *Given a realizability interpretation of L^\wp , we define:*

$$\begin{aligned} \blacksquare \mathcal{A} &\triangleq \mathcal{P}(\mathcal{V}_0) & \blacksquare a \wp b &\triangleq \{(V_1, V_2) : V_1 \in a \wedge V_2 \in b\} \\ \blacksquare a \preceq b &\triangleq a \supseteq b & \blacksquare \neg a &\triangleq [a^\perp] = \{[t] : t \in a^\perp\} \end{aligned}$$

where \perp is the pole, \mathcal{V}_0 is the set of closed values⁹, and (\cdot, \cdot) and $[\cdot]$ are the maps corresponding to \wp and \neg . The resulting quadruple $(\mathcal{A}, \preceq, \wp, \neg)$ is a disjunctive structure.

Following the interpretation of the λ -terms in implicative structures, we can embed L^\wp terms within disjunctive structures. We do not have the necessary space here to fully introduce here¹⁰, but it is worth mentioning that the orthogonality relation $t \perp e$ is interpreted via the ordering $t^A \preceq e^A$ (as suggested in [8, Theorem 5.13] by the definition of an abstract Krivine structure and its pole from an ordered combinatory algebra).

3.2 The induced implicative structure

As expected, any disjunctive structure directly induces an implicative structure through the definition $a \wp b \triangleq \neg a \wp b$:

► **Proposition 9.** *If $(\mathcal{A}, \preceq, \wp, \neg)$ is a disjunctive structure, then $(\mathcal{A}, \preceq, \wp)$ is an implicative structure.*

Therefore, we can again define for all a, b of \mathcal{A} the application ab as well as the abstraction λf for any function f from \mathcal{A} to \mathcal{A} ; and we get for free the properties of these encodings in implicative structures.

Up to this point, we have two ways of interpreting a λ -term into a disjunctive structure: either through the implicative structure which is induced by the disjunctive one, or by embedding into the L^\wp -calculus which is then interpreted within the disjunctive structure. As a sanity check, we verify that both coincide:

► **Proposition 10** (λ -calculus). *Let $\mathcal{A}^\wp = (\mathcal{A}, \preceq, \wp, \neg)$ be a disjunctive structure, and $\mathcal{A}^\rightarrow = (\mathcal{A}, \preceq, \wp)$ the implicative structure it canonically defines, we write ι for the corresponding inclusion. Let t be a closed λ -term (with parameter in \mathcal{A}), and $\llbracket t \rrbracket$ his embedding in L^\wp . Then we have $\iota(t^{\mathcal{A}^\rightarrow}) = \llbracket t \rrbracket^{\mathcal{A}^\wp}$.*

¹⁰See the extended version for more details.

3.3 Disjunctive algebras

We shall now introduce the notion of disjunctive separator. To this purpose, we adapt the definition of implicative separators, using standard axioms¹¹ for the disjunction and the negation instead of Hilbert's combinators **s** and **k**. We thus consider the following combinators:

$$\begin{array}{l|l} \mathfrak{s}_1^\exists \triangleq \lambda_{a \in \mathcal{A}} [(a \exists a) \rightarrow a] & \mathfrak{s}_4^\exists \triangleq \lambda_{a,b,c \in \mathcal{A}} [(a \rightarrow b) \rightarrow (c \exists a) \rightarrow (c \exists b)] \\ \mathfrak{s}_2^\exists \triangleq \lambda_{a,b \in \mathcal{A}} [a \rightarrow (a \exists b)] & \mathfrak{s}_5^\exists \triangleq \lambda_{a,b,c \in \mathcal{A}} [(a \exists (b \exists c)) \rightarrow ((a \exists b) \exists c)] \\ \mathfrak{s}_3^\exists \triangleq \lambda_{a,b \in \mathcal{A}} [(a \exists b) \rightarrow b \exists a] & \end{array}$$

Separators for \mathcal{A} are defined similarly to the separators for implicative structures, replacing the combinators **k**, **s** and **cc** by the previous ones.

► **Definition 11** (Separator). *We call separator for the disjunctive structure \mathcal{A} any subset $\mathcal{S} \subseteq \mathcal{A}$ that fulfills the following conditions for all $a, b \in \mathcal{A}$:*

1. *If $a \in \mathcal{S}$ and $a \preceq b$ then $b \in \mathcal{S}$.*
2. *$\mathfrak{s}_1^\exists, \mathfrak{s}_2^\exists, \mathfrak{s}_3^\exists, \mathfrak{s}_4^\exists$ and \mathfrak{s}_5^\exists are in \mathcal{S} .*
3. *If $a \rightarrow b \in \mathcal{S}$ and $a \in \mathcal{S}$ then $b \in \mathcal{S}$.*

A separator \mathcal{S} is said to be consistent if $\perp \notin \mathcal{S}$. We call disjunctive algebra the given of a disjunctive structure together with a separator $\mathcal{S} \subseteq \mathcal{A}$.

► **Remark 12.** *The reader may notice that in this section, we do not distinguish between classical and intuitionistic separators. Indeed, L^\exists and the corresponding fragment of the sequent calculus are intrinsically classical. As we shall see thereafter, so are the disjunctive algebras: the negation is always involutive modulo the equivalence $\cong_{\mathcal{S}}$ (Proposition 16).*

► **Remark 13** (Generalized modus ponens). *The modus ponens, that is the unique deduction rule we have, is actually compatible with meets. Consider a set I and two families $(a_i)_{i \in I}, (b_i)_{i \in I} \in \mathcal{A}^I$, we have:*

$$\frac{a \vdash_I b \quad \vdash_I a}{\vdash_I b}$$

where we write $a \vdash_I b$ for $(\lambda_{i \in I} a_i \rightarrow b_i) \in \mathcal{S}$ and $\vdash_I a$ for $(\lambda_{i \in I} a_i) \in \mathcal{S}$. As our axioms are themselves expressed as meets, the results that we will obtain internally (that is by deduction from the separator's axioms) can all be generalized to meets.

► **Example 14** (Complete Boolean algebras). *Once again, if \mathcal{B} is a complete Boolean algebra, \mathcal{B} induces a disjunctive structure in which it is easy to verify that the combinators $\mathfrak{s}_1^\exists, \mathfrak{s}_2^\exists, \mathfrak{s}_3^\exists, \mathfrak{s}_4^\exists$ and \mathfrak{s}_5^\exists are equal to the maximal element \top . Therefore, the singleton $\{\top\}$ is a valid separator for the induced disjunctive structure. In fact, the filters for \mathcal{B} are exactly its separators.*

► **Example 15** (L^\exists realizability model). *Remember from Example 8 that any model of classical realizability based on the L^\exists -calculus induces a disjunctive structure. As in the implicative case, the set of formulas realized by a closed term¹² defines a valid separator.*

¹¹ These axioms can be found for instance in Whitehead and Russell's presentation of logic [41]. In fact, the fifth axiom is deducible from the first four as was later shown by Bernays [2]. For simplicity reasons, we preferred to keep it as an axiom.

¹² Proof-like terms in L^\exists simply correspond to closed terms.

3.4 Internal logic

As in the case of implicative algebras, we say that a entails b and write $a \vdash_{\mathcal{S}} b$ if $a \rightarrow b \in \mathcal{S}$. Through this relation, which is again a preorder relation, we can relate the primitive negation and disjunction to the negation and sum type induced by the underlying implicative structure:

$$a + b \triangleq \bigwedge_{c \in \mathcal{A}} ((a \rightarrow c) \rightarrow (b \rightarrow c) \rightarrow c) \quad (\forall a, b \in \mathcal{A})$$

In particular, we show that from the point of view of the separator the principle of double negation elimination is valid and the disjunction and this sum type are equivalent:

► **Proposition 16** (Implicative connectives). *For all $a, b \in \mathcal{A}$, the following holds:*

- | | | |
|--|--|---|
| 1. $\neg a \vdash_{\mathcal{S}} a \rightarrow \perp$ | 3. $a \vdash_{\mathcal{S}} \neg\neg a$ | 5. $a \wp b \vdash_{\mathcal{S}} a + b$ |
| 2. $a \rightarrow \perp \vdash_{\mathcal{S}} \neg a$ | 4. $\neg\neg a \vdash_{\mathcal{S}} a$ | 6. $a + b \vdash_{\mathcal{S}} a \wp b$ |

3.5 Induced implicative algebras

In order to show that any disjunctive algebra is a particular case of implicative algebra, we first verify that Hilbert's combinators belong to any disjunctive separator:

► **Proposition 17** (Combinators). *We have: 1. $\mathbf{k}^{\mathcal{A}} \in \mathcal{S}$ 2. $\mathbf{s}^{\mathcal{A}} \in \mathcal{S}$ 3. $\mathbf{cc}^{\mathcal{A}} \in \mathcal{S}$*

As a consequence, we get the expected theorem:

► **Theorem 18.** *Any disjunctive algebra is a classical implicative algebra.*

Since any disjunctive algebra is actually a particular case of implicative algebra, the construction leading to the implicative tripos can be rephrased entirely in this framework. In particular, the same criteria allows us to determine whether the implicative tripos is isomorphic to a forcing tripos. Notably, a disjunctive algebra admitting an extra-commutation rule the negation \neg with arbitrary joins ($\neg \bigvee_{a \in A} a = \bigwedge_{a \in A} \neg a$) will induce an implicative algebra where the arrow commutes with arbitrary joins. In that case, the induced tripos would collapse to a forcing situation (see [31]).

4 A positive decomposition: conjunctive algebras

4.1 Call-by-value realizability models

While there exists now several models build of classical theories constructed via Krivine realizability [22, 24, 25, 29], they all have in common that they rely on a presentation of logic based on negative connectives/quantifiers. If this might not seem shocking from a mathematical perspective, it has the computational counterpart that these models all build on a call-by-name calculus, namely the λ_c -calculus¹³. In light of the logical consequences that computational choices have on the induced theory, it is natural to wonder whether the choice of a call-by-name evaluation strategy is anecdotal or fundamental.

As a first step in this direction, we analyze here the algebraic structure of realizability models based on the L^{\otimes} calculus, the positive fragment of Munch-Maccagnoni's system L

¹³ Actually, there is two occurrences of realizability interpretations for call-by-value calculus, including Munch-Maccagnoni's system L, but both are focused on the analysis of the computational behavior of programs rather than constructing models of a given logic [35, 27].

30:10 Revisiting the Duality of Computation

corresponding to the formulas defined by: $A, B ::= X \mid \neg A \mid A \otimes B \mid \exists X.A$. Through the well-known duality between terms and evaluation contexts [4, 35], this fragment is dual to the L^\exists calculus and it naturally allows to embed the λ -terms evaluated in a call-by-value fashion. We shall now reproduce the approach we had for L^\exists : guided by the analysis of the realizability models induced by the L^\otimes calculus, we first define *conjunctive structures*. We then show how these structures can be equipped with a separator and how the resulting *conjunctive algebras* lead to the construction of a *conjunctive tripos*. We will finally show in the next section how conjunctive and disjunctive algebras are related by an algebraic duality.

4.2 Conjunctive structures

As in the previous section, we will not introduce here the L^\otimes calculus and the corresponding realizability models (see the extended version for details). Their main characteristic is that, being build on top of a call-by-value calculus, a formula A is primitively interpreted by its *ground truth value* $|A|_v \in \mathcal{P}(\mathcal{V}_O)$ which is a set of values. Its falsity and truth values are then defined by orthogonality [35, 27]. Once again, we can observe the existing commutations in these realizability models. Insofar as we are in a structure centered on positive connectives, we especially pay attention to the commutations with joins. As a matter of fact, in any L^\otimes realizability model, we have that if $X \notin FV(B)$:

1. $|\exists X.(A \otimes B)|_V = |(\exists X.A) \otimes B|_V$.
2. $|\exists X.(B \otimes A)|_V = |B \otimes (\exists X.A)|_V$.
3. $|\neg(\exists X.A)|_V = \bigcap_{S \in \mathcal{P}(\mathcal{V}_O)} |\neg A\{X := \dot{S}\}|_V$

Since we are now interested in primitive truth values, which are logically ordered by inclusion (in particular, the existential quantifier is interpreted by unions, thus joins), the previous proposition advocates for the following definition:

► **Definition 19** (Conjunctive structure). *A conjunctive structure is a complete join-semilattice (\mathcal{A}, \preceq) equipped with a binary operation $(a, b) \mapsto a \otimes b$, and a unary operation $a \mapsto \neg a$, such that for all $a, a', b, b' \in \mathcal{A}$ and for all subset $B \subseteq \mathcal{A}$ we have:*

1. if $a \preceq a'$ then $\neg a' \preceq \neg a$
2. if $a \preceq a'$ and $b \preceq b'$ then $a \otimes b \preceq a' \otimes b'$
3. $\bigvee_{b \in B} (a \otimes b) = a \otimes (\bigvee_{b \in B} b)$
4. $\bigvee_{b \in B} (b \otimes a) = (\bigvee_{b \in B} b) \otimes a$
5. $\neg \bigvee_{a \in A} a = \bigwedge_{a \in A} \neg a$

As in the cases of implicative and disjunctive structures, the commutation rules imply that: 1. $\perp \otimes a = \perp$ 2. $a \otimes \perp = \perp$ 3. $\neg \perp = \top$

► **Example 20** (Dummy conjunctive structure). *Given a complete lattice L , the following definitions give rise to a dummy conjunctive structure: $a \otimes b \triangleq \perp$ $\neg a \triangleq \top$.*

► **Example 21** (Complete Boolean algebras). *Let \mathcal{B} be a complete Boolean algebra. It embodies a conjunctive structure, that is defined by:*

- $\mathcal{A} \triangleq \mathcal{B}$ ■ $a \preceq b \triangleq a \leq b$ ■ $a \otimes b \triangleq a \wedge b$ ■ $\neg a \triangleq \neg a$

► **Example 22** (L^\otimes realizability models). *As for the disjunctive case, we can abstract the structure of the realizability interpretation of L^\otimes to define:*

- $\mathcal{A} \triangleq \mathcal{P}(\mathcal{V}_O)$ ■ $a \preceq b \triangleq a \subseteq b$
- $a \otimes b \triangleq \{(V_1, V_2) : V_1 \in a \wedge V_2 \in b\}$ ■ $\neg a \triangleq [a^\perp] = \{[e] : e \in a^\perp\}$

where \perp is the pole, \mathcal{V}_O is the set of closed values and (\cdot, \cdot) and $[\cdot]$ are the maps corresponding to \otimes and \neg . The resulting quadruple $(\mathcal{A}, \preceq, \otimes, \neg)$ is a conjunctive structure.

It is worth noting that even though we can define an arrow by $a \multimap b \triangleq \neg(a \otimes \neg b)$, it does not induce an implicative structure: indeed, the distributivity law is not true in general¹⁴. In turns, we have another distributivity law which is usually wrong in implicative structure:

$$\left(\prod_{a \in A} a \right) \multimap b = \prod_{a \in A} (a \multimap b) \qquad \prod_{b \in B} (a \multimap b) \not\leq a \multimap \left(\prod_{b \in B} b \right)$$

Actually, implicative structures where both are true corresponds precisely to a degenerated forcing situation.

Here again, we can define an embedding of L^\otimes into any conjunctive structure which is sound with respect to typing and reductions¹⁵.

4.3 Conjunctive algebras

The definition of conjunctive separators turns out to be more subtle than in the disjunctive case. Among others things, conjunctive structures mainly axiomatize joins, while the combinators or usual mathematical axioms that we could wish to have in a separator are more naturally expressed via universal quantifications, hence meets. Yet, an analysis of the sequent calculus underlying L^\otimes type system¹⁵, shows that we could consider a tensorial calculus where deduction systematically involves a conclusion of the shape $\neg A$. This justifies to consider the following combinators¹⁶:

$$\begin{array}{l} \mathfrak{s}_1^\otimes \triangleq \lambda_{a \in \mathcal{A}} \neg [\neg(a \otimes a) \otimes a] \\ \mathfrak{s}_2^\otimes \triangleq \lambda_{a, b \in \mathcal{A}} \neg [\neg a \otimes (a \otimes b)] \\ \mathfrak{s}_3^\otimes \triangleq \lambda_{a, b \in \mathcal{A}} \neg [(a \otimes b) \otimes (b \otimes a)] \end{array} \quad \left| \quad \begin{array}{l} \mathfrak{s}_4^\otimes \triangleq \lambda_{a, b, c \in \mathcal{A}} \neg [\neg(\neg a \otimes b) \otimes (\neg(c \otimes a) \otimes (c \otimes b))] \\ \mathfrak{s}_5^\otimes \triangleq \lambda_{a, b, c \in \mathcal{A}} \neg [\neg(a \otimes (b \otimes c)) \otimes ((a \otimes b) \otimes c)] \end{array}$$

and to define conjunctive separators as follows:

► **Definition 23** (Separator). *We call separator for the disjunctive structure \mathcal{A} any subset $\mathcal{S} \subseteq \mathcal{A}$ that fulfills the following conditions for all $a, b \in \mathcal{A}$:*

1. If $a \in \mathcal{S}$ and $a \preceq b$ then $b \in \mathcal{S}$.
2. $\mathfrak{s}_1^\otimes, \mathfrak{s}_2^\otimes, \mathfrak{s}_3^\otimes, \mathfrak{s}_4^\otimes$ and \mathfrak{s}_5^\otimes are in \mathcal{S} .
3. If $\neg(a \otimes b) \in \mathcal{S}$ and $a \in \mathcal{S}$ then $\neg b \in \mathcal{S}$.
4. If $a \in \mathcal{S}$ and $b \in \mathcal{S}$ then $a \otimes b \in \mathcal{S}$.

A separator \mathcal{S} is said to be classical if besides $\neg\neg a \in \mathcal{S}$ implies $a \in \mathcal{S}$.

► **Remark 24** (Modus Ponens). *If the separator is classical, it is easy to see that the modus ponens is valid: if $a \multimap b \in \mathcal{S}$ and $a \in \mathcal{S}$, then $\neg\neg b \in \mathcal{S}$ by (3) and thus $b \in \mathcal{S}$.*

► **Example 25** (Complete Boolean algebras). *Once again, if \mathcal{B} is a complete Boolean algebra, \mathcal{B} induces a conjunctive structure in which it is easy to verify that the combinators $\mathfrak{s}_1^\otimes, \mathfrak{s}_3^\otimes, \mathfrak{s}_3^\otimes, \mathfrak{s}_4^\otimes$ and \mathfrak{s}_5^\otimes are equal to the maximal element \top . Therefore, the singleton $\{\top\}$ is a valid separator.*

► **Example 26** (L^\otimes realizability model). *As expected, the set of realized formulas by a proof-like term: defines a valid separator for the conjunctive structures induced by L^\otimes realizability models.*

¹⁴ For instance, it is false in L^\otimes realizability models.

¹⁵ See the extended version for more details.

¹⁶ Observe that are directly dual to the combinators for disjunctive separators and that they can be alternatively given the shape $\neg \prod_{a \in \mathcal{A}} \dots$

30:12 Revisiting the Duality of Computation

► **Example 27** (Kleene realizability). *We do not want to enter into too much details here, but it is worth mentioning that realizability interpretations à la Kleene of intuitionistic calculi equipped with primitive pairs (e.g. (partial) combinatory algebras, the λ -calculus) induce conjunctive algebras. Insofar as many Kleene realizability models takes position against classical reasoning (for $\forall X.X \vee \neg X$ is not realized and hence its negation is), these algebras have the interesting properties of not being classical (and are even incompatible with a classical completion).*

► **Remark 28** (Generalized axioms). *Once again, the axioms (3) and (4) generalize to meet of families $(a_i)_{i \in I}, (b_i)_{i \in I}$:*

$$\frac{\vdash_I \neg(a \otimes b) \quad \vdash_I a}{\vdash_I \neg b} \qquad \frac{\vdash_I a \quad \vdash_I b}{\vdash_I a \otimes b}$$

where we write $\vdash_I a$ for $(\bigwedge_{i \in I} a_i) \in \mathcal{S}$ and where the negation and conjunction of families are taken pointwise. Once again, the axioms being themselves expressed as meets, this means that any result obtained from the separator's axioms (but the classical one) can be generalized to meets.

4.4 Internal logic

As before, we consider the entailment relation defined by $a \vdash_{\mathcal{S}} b \triangleq (a \overset{\otimes}{\rightarrow} b) \in \mathcal{S}$. Observe that if the separator is not classical, we do not have that $a \vdash_{\mathcal{S}} b$ and $a \in \mathcal{S}$ entails¹⁷ $b \in \mathcal{S}$. Nonetheless, this relation still defines a preorder in the sense that:

► **Proposition 29** (Preorder). *For any $a, b, c \in \mathcal{A}$, we have:*

1. $a \vdash_{\mathcal{S}} a$
2. If $a \vdash_{\mathcal{S}} b$ and $b \vdash_{\mathcal{S}} c$ then $a \vdash_{\mathcal{S}} c$

Intuitively, this reflects the fact that despite we may not be able to extract the value of a computation, we can always chain it with another computation expecting a value.

Here again, we can relate the negation $\neg a$ to the one induced by the arrow $a \overset{\otimes}{\rightarrow} \perp$:

► **Proposition 30** (Implicative negation). *For all $a \in \mathcal{A}$, the following holds:*

1. $\neg a \vdash_{\mathcal{S}} a \overset{\otimes}{\rightarrow} \perp$
2. $a \overset{\otimes}{\rightarrow} \perp \vdash_{\mathcal{S}} \neg a$
3. $a \vdash_{\mathcal{S}} \neg \neg a$
4. $\neg \neg a \vdash_{\mathcal{S}} a$

As in implicative structures, we can define the abstraction and application of the λ -calculus:

$$\lambda f \triangleq \bigwedge_{a \in \mathcal{A}} (a \overset{\otimes}{\rightarrow} f(a)) \qquad ab \triangleq \bigwedge \{ \neg \neg c : a \preceq b \overset{\otimes}{\rightarrow} c \}$$

Observe that here we need to add a double negation, since intuitively ab is a *computation* of type $\neg \neg c$ rather than a value of type c . In other words, values are not stable by applications, and extracting a value from a computation requires a form of classical control. Nevertheless, for any separator we have:

► **Proposition 31.** *If $a \in \mathcal{S}$ and $b \in \mathcal{S}$ then $ab \in \mathcal{S}$.*

Similarly, the beta reduction rule now involves a double-negation on the reduced term:

¹⁷ Actually we can consider a different relation $a \vdash^{\neg} b \triangleq \neg(a \otimes b)$ for which $a \vdash^{\neg} b$ and $a \in \mathcal{S}$ entails $\neg b$. This one turns out to be useful to ease proofs, but from a logical perspective, the significant entailment is the one given by $a \vdash_{\mathcal{S}} b$.

► **Proposition 32.** $(\lambda f)a \preceq \neg\neg f(a)$

We show that Hilbert’s combinators **k** and **s** belong to any conjunctive separator:

► **Proposition 33 (k and s).** *We have:*

1. $(\lambda xy.x)^A \in \mathcal{S}$
2. $(\lambda xyz.xz(yz))^A \in \mathcal{S}$

By combinatorial completeness, for any closed λ -term t we thus have the a combinatorial term t_0 (i.e. a composition of **k** and **s**) such that $t_0 \rightarrow^* t$. Since \mathcal{S} is closed under application, t_0^A also belong to \mathcal{S} . Besides, since for each reduction step $t_n \rightarrow t_{n+1}$, we have $t_n^A \preceq \neg\neg t_{n+1}^A$, if the separator is classical¹⁸, we can thus deduce that it contains the interpretation of t :

► **Theorem 34 (λ -calculus).** *If \mathcal{S} is classical and t is a closed λ -term, then $t^A \in \mathcal{S}$.*

Once more, the entailment relation induces a structure of (pre)-Heyting algebra, whose conjunction and disjunction are naturally given by $a \times b \triangleq a \otimes b$ and $a + b \triangleq \neg(\neg a \otimes \neg b)$:

► **Proposition 35 (Heyting Algebra).** *For any $a, b, c \in \mathcal{A}$ For any $a, b, c \in \mathcal{A}$, we have:*

1. $a \times b \vdash_{\mathcal{S}} a$
2. $a \times b \vdash_{\mathcal{S}} b$
3. $a \vdash_{\mathcal{S}} a + b$
4. $b \vdash_{\mathcal{S}} a + b$
5. $a \vdash_{\mathcal{S}} b \overset{\otimes}{\rightarrow} c$ iff $a \times b \vdash_{\mathcal{S}} c$

We can thus quotient the algebra by the equivalence relation $\cong_{\mathcal{S}}$ and extend the previous operation to equivalence classes in order to obtain a Heyting algebra $\mathcal{A}/\cong_{\mathcal{S}}$. In particular, this allows us to obtain a tripos out of a conjunctive algebra by reproducing the construction of the implicative tripos in our setting:

► **Theorem 36 (Conjunctive tripos).** *Let $(\mathcal{A}, \preceq, \rightarrow, \mathcal{S})$ be a classical¹⁹ conjunctive algebra. The following functor (where $f : J \rightarrow I$) defines a tripos:*

$$\mathcal{T} : I \mapsto \mathcal{A}^I/\mathcal{S}[I] \qquad \mathcal{T}(f) : \begin{cases} \mathcal{A}^I/\mathcal{S}[I] & \rightarrow & \mathcal{A}^J/\mathcal{S}[J] \\ [(a_i)_{i \in I}] & \mapsto & [(a_{f(j)})_{j \in J}] \end{cases}$$

5 The duality of computation, algebraically

In [4], Curien and Herbelin introduce the $\lambda\mu\tilde{\mu}$ in order to emphasize the so-called duality of computation between terms and evaluation contexts. They define a simple translation inverting the role of terms and stacks within the calculus, which has the notable consequence of translating a call-by-value calculus into a call-by-name calculus and vice-versa. The very same translation can be expressed within L, in particular it corresponds to the trivial translation from mapping every constructor on terms (resp. destructors) in L^{\otimes} to the corresponding constructor on stacks (resp. destructors) in L^{\otimes} . We shall now see how this fundamental duality of computation can be retrieved algebraically between disjunctive and conjunctive algebras.

We first show that we can simply pass from one structure to another by reversing the order relation. We know that reversing the order in a complete lattice yields a complete

¹⁸ Actually, since we always have that if $\neg\neg\neg\neg a \in \mathcal{S}$ then $\neg\neg a \in \mathcal{S}$, the same proof shows that in the intuitionistic case we have $a \neg\neg t^A \in \mathcal{S}$.

¹⁹ For technical reasons, we only give the proof in case where the separator is classical (recall that it allows to directly use λ -terms), but as explained, by adding double negation everywhere the same reasoning should work for the general case as well. Yet, this is enough to express our main result in the next section which only deals with the classical case.

lattice in which meets and joins are exchanged. Therefore, it only remains to verify that the axioms of disjunctive and conjunctive structures can be deduced through this duality one from each other, which is the case.

► **Proposition 37.** *Let $(\mathcal{A}, \preceq, \wp, \neg)$ be a disjunctive structure. Let us define:*
 ■ $\mathcal{A}^\otimes \triangleq \mathcal{A}^\wp$ ■ $a \triangleleft b \triangleq b \preceq a$ ■ $a \otimes b \triangleq a \wp b$ ■ $\neg a \triangleq \neg a$
then $(\mathcal{A}^\otimes, \triangleleft, \otimes, \neg)$ is a conjunctive structure.

► **Proposition 38.** *Let $(\mathcal{A}, \preceq, \otimes, \neg)$ be a conjunctive structure. Let us define:*
 ■ $\mathcal{A}^\wp \triangleq \mathcal{A}^\otimes$ ■ $a \triangleleft b \triangleq b \preceq a$ ■ $a \wp b \triangleq a \otimes b$ ■ $\neg a \triangleq \neg a$
then $(\mathcal{A}^\wp, \triangleleft, \wp, \neg)$ is a disjunctive structure.

Intuitively, by considering stacks as realizers, we somehow reverse the algebraic structure, and we consider as valid formulas the ones whose orthogonals were valid. In terms of separator, it means that when reversing a structure we should consider the separator defined as the preimage through the negation of the original separator.

► **Theorem 39.** *Let $(\mathcal{A}^\otimes, \mathcal{S}^\otimes)$ be a conjunctive algebra, the set $\mathcal{S}^\wp \triangleq \{a \in \mathcal{A} : \neg a \in \mathcal{S}^\otimes\}$ defines a valid separator for the dual disjunctive structure \mathcal{A}^\wp .*

► **Theorem 40.** *Let $(\mathcal{A}^\wp, \mathcal{S}^\wp)$ be a disjunctive algebra. The set $\mathcal{S}^\otimes \triangleq \{a \in \mathcal{A} : \neg a \in \mathcal{S}^\wp\}$ defines a classical separator for the dual conjunctive structure \mathcal{A}^\otimes .*

It is worth noting that reversing in both cases, the dual separator is classical. This is to connect with the fact that classical reasoning principles are true on negated formulas. Moreover, starting from a non-classical conjunctive algebra, one can reverse it twice to get a classical algebra. This corresponds to a classical completion of the original separator \mathcal{S} : it is easy to see that $a \in \mathcal{S}$ implies $\neg\neg a \in \mathcal{S}$, hence $\mathcal{S} \subseteq \{a : \neg\neg a \in \mathcal{S}\}$.

Actually, the duality between disjunctive and (classical) conjunctive algebras is even stronger, in the sense that through the translation, the induced triposes are isomorphic. Remember that an isomorphism φ between two (**Set**-based) triposes $\mathcal{T}, \mathcal{T}'$ is defined as a natural isomorphism $\mathcal{T} \Rightarrow \mathcal{T}'$ in the category **HA**, that is as a family of isomorphisms $\varphi_I : \mathcal{T}(I) \xrightarrow{\sim} \mathcal{T}'(I)$ (indexed by all $I \in \mathbf{Set}$) that is natural in \mathcal{I} .

► **Theorem 41 (Main result).** *Let $(\mathcal{A}, \mathcal{S})$ be a disjunctive algebra and $(\bar{\mathcal{A}}, \bar{\mathcal{S}})$ its dual conjunctive algebra. The following family of maps defines a tripos isomorphism:*

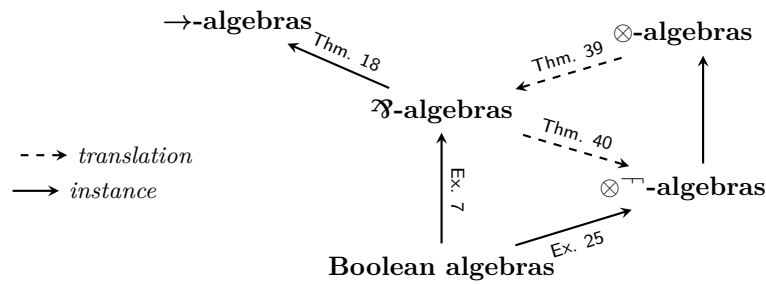
$$\varphi_I : \begin{cases} \bar{\mathcal{A}}/\bar{\mathcal{S}}[I] & \rightarrow & \mathcal{A}/\mathcal{S}[I] \\ [a_i] & \mapsto & [\neg a_i] \end{cases}$$

6 Conclusion

6.1 An algebraic view on the duality of computation

To sum up, in this paper we saw how the two decompositions of the arrow $a \rightarrow b$ as $\neg a \wp b$ and $\neg(a \otimes \neg b)$, which respectively induce decompositions of a call-by-name and call-by-value λ -calculi within Munch-Maccagnoni's system L [35], yield two different algebraic structures reflecting the corresponding realizability models. Namely, call-by-name models give rise to disjunctive algebras, which are particular cases of Miquel's implicative algebras [31]; while conjunctive algebras correspond to call-by-value realizability models.

The well-known duality of computation between terms and contexts is reflected here by simple translations from conjunctive to disjunctive algebras and vice-versa, where the underlying lattices are simply reversed. Besides, we showed that (classical) conjunctive algebras induce triposes that are isomorphic to disjunctive triposes. The situation is summarized in Figure 1, where \otimes^\top denotes classical conjunctive algebras.



■ **Figure 1** Final picture.

6.2 From Kleene to Krivine via negative translation

We could now re-read within our algebraic landscape the result of Oliva and Streicher stating that Krivine realizability models for PA2 can be obtained as a composition of Kleene realizability for HA2 and Friedman’s negative translation [36, 30]. Interestingly, in this setting the fragment of formulas that is interpreted in HA2 correspond exactly to the positive formulas of L^\otimes , so that it gives rise to an (intuitionistic) conjunctive algebra. Friedman’s translation is then used to encode the type of stacks within this fragment via a negation. In the end, realized formulas are precisely the ones that are realized through Friedman’s translation: the whole construction exactly matches the passage from an intuitionistic conjunctive structure defined by Kleene realizability to a classical implicative algebras through the arrow from \otimes -algebras to \rightarrow -algebras via \wp -algebras.

6.3 Future work

While Theorem 41 implies that call-by-value and call-by-name models based on the L^\otimes and L^\wp calculi are equivalents, it does not provide us with a definitive answer to our original question. Indeed, just as (by-name) implicative algebras are more general than disjunctive algebras, it could be the case that there exists a notion of (by-value) implicative algebras that is strictly more general than conjunctive algebras and which is not isomorphic to a by-name situation.

Also, if we managed to obtain various results about conjunctive algebras, there is still a lot to understand about them. Notably, the interpretation we have of the λ -calculus is a bit disappointing in that it does not provide us with an adequacy result as nice as in implicative algebras. In particular, the fact that each application implicitly gives rise to a double negation breaks the compositionality. This is of course to connect with the definition of *truth values* in by-value models which requires three layers and a double orthogonal. We thus feel that many things remain to understand about the underlying structure of by-value realizability models.

Finally, on a long-term perspective, the next step would be to understand the algebraic impact of more sophisticated evaluation strategy (e.g., call-by-need) or side effects (e.g., a monotonic memory). While both have been used in concrete cases to give a computational content to certain axioms (e.g., the axiom of dependent choice [15]) or model constructions (e.g., forcing [21]), for the time being we have no idea on how to interpret them in the realm of implicative algebras.

References

- 1 John L. Bell. *Set Theory: Boolean-Valued Models and Independence Proofs*. Oxford: Clarendon Press, 2005.
- 2 P. Bernays. Axiomatische Untersuchung des Aussagen-Kalküls der "Principia Mathematica". *Mathematische Zeitschrift*, 25:305–320, 1926.
- 3 Luca Cardelli, Simone Martini, John C. Mitchell, and Andre Scedrov. *An extension of system F with subtyping*, pages 750–770. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991. doi:10.1007/3-540-54415-1_73.
- 4 Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proceedings of ICFP 2000*, SIGPLAN Notices 35(9), pages 233–243. ACM, 2000. doi:10.1145/351240.351262.
- 5 Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. LKQ and LKT : Sequent calculi for second order logic based upon dual linear decompositions of classical implication. In *Advances in Linear Logic*, 1995.
- 6 W. Ferrer and O. Malherbe. The category of implicative algebras and realizability. *ArXiv e-prints*, December 2017. arXiv:1712.06043.
- 7 W. Ferrer Santos, M. Guillermo, and O. Malherbe. A Report on Realizability. *ArXiv e-prints*, 2013. arXiv:1309.0706.
- 8 W. Ferrer Santos, M. Guillermo, and O. Malherbe. Realizability in OCAs and AKSs. *ArXiv e-prints*, 2015. arXiv:1512.07879.
- 9 Walter Ferrer Santos, Jonas Frey, Mauricio Guillermo, Octavio Malherbe, and Alexandre Miquel. Ordered combinatory algebras and realizability. *Mathematical Structures in Computer Science*, 27(3):428–458, 2017. doi:10.1017/S0960129515000432.
- 10 Jonas Frey. Realizability Toposes from Specifications. In Thorsten Altenkirch, editor, *13th International Conference on Typed Lambda Calculi and Applications (TLCA 2015)*, volume 38 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 196–210, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.TLCA.2015.196.
- 11 Jonas Frey. Classical Realizability in the CPS Target Language. *Electronic Notes in Theoretical Computer Science*, 325(Supplement C):111–126, 2016. The Thirty-second Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXII). doi:10.1016/j.entcs.2016.09.034.
- 12 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987. doi:10.1016/0304-3975(87)90045-4.
- 13 Timothy G. Griffin. A Formulae-as-type Notion of Control. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, pages 47–58, New York, NY, USA, 1990. ACM. doi:10.1145/96709.96714.
- 14 Hugo Herbelin. An Intuitionistic Logic that Proves Markov's Principle. In *LICS 2010, Proceedings*, 2010. doi:10.1109/LICS.2010.49.
- 15 Hugo Herbelin. A Constructive Proof of Dependent Choice, Compatible with Classical Logic. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 365–374. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.47.
- 16 Pieter Hofstra and Jaap Van Oosten. Ordered partial combinatory algebras. *Mathematical Proceedings of the Cambridge Philosophical Society*, 134(3):445–463, 2003. doi:10.1017/S0305004102006424.
- 17 Guilhem Jaber, Gabriel Lewertowski, Pierre-Marie Pédrot, Matthieu Sozeau, and Nicolas Tabareau. The Definitional Side of the Forcing. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16*, pages 367–376, New York, NY, USA, 2016. ACM. doi:10.1145/2933575.2935320.
- 18 Guilhem Jaber, Nicolas Tabareau, and Matthieu Sozeau. Extending Type Theory with Forcing. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer*

- Science*, LICS '12, pages 395–404, Washington, DC, USA, 2012. IEEE Computer Society. doi:10.1109/LICS.2012.49.
- 19 J.-L. Krivine. Dependent choice, ‘quote’ and the clock. *Th. Comp. Sc.*, 308:259–276, 2003.
 - 20 J.-L. Krivine. Realizability in classical logic. In Interactive models of computation and program behaviour. *Panoramas et synthèses*, 27, 2009.
 - 21 J.-L. Krivine. Realizability algebras: a program to well order R. *Logical Methods in Computer Science*, 7(3), 2011. doi:10.2168/LMCS-7(3:2)2011.
 - 22 J.-L. Krivine. Realizability algebras II : new models of ZF + DC. *Logical Methods in Computer Science*, 8(1):10, February 2012. 28 p. doi:10.2168/LMCS-8(1:10)2012.
 - 23 J.-L. Krivine. Quelques propriétés des modèles de réalisabilité de ZF, February 2014. URL: <http://hal.archives-ouvertes.fr/hal-00940254>.
 - 24 Jean-Louis Krivine. On the Structure of Classical Realizability Models of ZF. In Hugo Herbelin, Pierre Letouzey, and Matthieu Sozeau, editors, *20th International Conference on Types for Proofs and Programs (TYPES 2014)*, volume 39 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 146–161, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.TYPES.2014.146.
 - 25 Jean-Louis Krivine. Bar Recursion in Classical Realisability: Dependent Choice and Continuum Hypothesis. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:11, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2016.25.
 - 26 Olivier Laurent. *A study of polarization in logic*. PhD thesis, Université de la Méditerranée - Aix-Marseille II, March 2002. URL: <https://tel.archives-ouvertes.fr/tel-00007884>.
 - 27 Rodolphe Lepigre. A Classical Realizability Model for a Semantical Value Restriction. In Peter Thiemann, editor, *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 476–502. Springer, 2016. doi:10.1007/978-3-662-49498-1_19.
 - 28 Paul Blain Levy. *Call-By-Push-Value: A Functional/Imperative Synthesis*, volume 2 of *Semantics Structures in Computation*. Springer, 2004. doi:10.1007/978-94-007-0954-6.
 - 29 A. Miquel. Forcing as a Program Transformation. In *LICS*, pages 197–206. IEEE Computer Society, 2011. doi:10.1109/LICS.2011.47.
 - 30 Alexandre Miquel. Existential witness extraction in classical realizability and via a negative translation. *Logical Methods in Computer Science*, 7(2):188–202, 2011. doi:10.2168/LMCS-7(2:2)2011.
 - 31 Alexandre Miquel. Implicative algebras: a new foundation for realizability and forcing. *ArXiv e-prints*, 2018. arXiv:1802.00528.
 - 32 Étienne Miquey. *Classical realizability and side-effects*. Ph.D. thesis, Université Paris Diderot ; Universidad de la República, Uruguay, November 2017. URL: <https://hal.inria.fr/tel-01653733>.
 - 33 Étienne Miquey. A Sequent Calculus with Dependent Types for Classical Arithmetic. In *LICS 2018*, pages 720–729. ACM, 2018. doi:10.1145/3209108.3209199.
 - 34 Étienne Miquey. Formalizing Implicative Algebras in Coq. In Jeremy Avigad and Assia Mahboubi, editors, *Interactive Theorem Proving*, pages 459–476. Springer International Publishing, 2018. doi:10.1007/978-3-319-94821-8_27.
 - 35 Guillaume Munch-Maccagnoni. Focalisation and Classical Realisability. In Erich Grädel and Reinhard Kahle, editors, *Computer Science Logic '09*, volume 5771 of *Lecture Notes in Computer Science*, pages 409–423. Springer, Heidelberg, 2009. doi:10.1007/978-3-642-04027-6_30.
 - 36 P. Oliva and T. Streicher. On Krivine’s Realizability Interpretation of Classical Second-Order Arithmetic. *Fundam. Inform.*, 84(2):207–220, 2008. URL: <http://iospress.metapress.com/content/f51774wm73404583/>.

30:18 Revisiting the Duality of Computation

- 37 Andrew M. Pitts. Tripos theory in retrospect. *Mathematical Structures in Computer Science*, 12(3):265–279, 2002. doi:10.1017/S096012950200364X.
- 38 Thomas Streicher. Krivine’s classical realisability from a categorical perspective. *Mathematical Structures in Computer Science*, 23(6):1234–1256, 2013. doi:10.1017/S0960129512000989.
- 39 Jaap van Oosten. Studies in Logic and the Foundations of Mathematics. In *Realizability: An Introduction to its Categorical Side*, volume 152 of *Studies in Logic and the Foundations of Mathematics*, pages ii–. Elsevier, 2008. doi:10.1016/S0049-237X(13)72046-9.
- 40 Jaap van Oosten and Zou Tingxiang. Classical and Relative Realizability. *Theory and Applications of Categories*, 31:571–593, March 2016.
- 41 Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. Cambridge University Press, 1925–1927.

Separation and Renaming in Nominal Sets

Joshua Moerman

RWTH Aachen University, Germany
joshua@cs.rwth-aachen.de

Jurriaan Rot

University College London, United Kingdom and Radboud University, The Netherlands
jrot@cs.ru.nl

Abstract

Nominal sets provide a foundation for reasoning about names. They are used primarily in syntax with binders, but also, e.g., to model automata over infinite alphabets. In this paper, nominal sets are related to *nominal renaming sets*, which involve arbitrary substitutions rather than permutations, through a categorical adjunction. In particular, the left adjoint relates the separated product of nominal sets to the Cartesian product of nominal renaming sets. Based on these results, we define the new notion of *separated nominal automata*. We show that these automata can be exponentially smaller than classical nominal automata, if the semantics is closed under substitutions.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Nominal sets, Separated product, Adjunction, Automata, Coalgebra

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.31

Related Version An extended version of the paper is available at [19], <https://arxiv.org/abs/1906.00763>.

Funding This work was partially supported by the ERC AdG project 787914 FRAPPANT and a Marie Curie Fellowship (grant code 795119).

Acknowledgements We would like to thank Jamie Gabbay, Gerco van Heerdt, Tom Hirschowitz, Bart Jacobs, and the anonymous reviewers for their useful comments.

1 Introduction

Nominal sets are abstract sets which allow one to reason over sets with names, in terms of permutations and symmetries. Since their introduction in computer science [11], they have been widely used for implementing and reasoning over syntax with binders [22]. Further, nominal techniques have been related to computability theory [4] and automata theory [3], where they provide an elegant means of studying languages over infinite alphabets. This embeds nominal techniques in a broader setting of *symmetry aware computation* [24].

Gabbay, one of the pioneers of nominal techniques described a variation on the theme: *nominal renaming sets* [9, 10]. Nominal renaming sets are equipped with a monoid action of arbitrary (possibly non-injective) substitution of names, in contrast to nominal sets, which only involve a group action of permutations.

In this paper, we further investigate the relationship between nominal renaming sets and nominal sets, and apply the results to nominal automata theory. We start by establishing a categorical adjunction (Section 3):

$$\text{Pm-Nom} \begin{array}{c} \xrightarrow{F} \\ \perp \\ \xleftarrow{U} \end{array} \text{Sb-Nom} ,$$

where **Pm-Nom** is the usual category of nominal sets and **Sb-Nom** the category of nominal renaming sets. The right adjoint U simply forgets the action of non-injective substitutions.



© Joshua Moerman and Jurriaan Rot;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 31; pp. 31:1–31:17



Leibniz International Proceedings in Informatics

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

The functor F was presented by Dowek and Gabbay [6]; it freely extends a nominal set with elements representing the application of such substitutions. For instance, F maps the nominal set $\mathbb{A}^{(*)}$ of all words consisting of distinct atoms to the nominal renaming set \mathbb{A}^* consisting of all words over the atoms.

In fact, the latter equivalence is a consequence of one of the main results of this paper: the left adjoint F maps the *separated product* $X * Y$ of nominal sets to the Cartesian product of nominal renaming sets (Theorem 3.6 & 3.7). The separated product consists of those pairs whose elements have disjoint supports. This is relevant for name abstraction [22], and has also been studied in the setting of presheaf categories, aimed towards separation logic [21]. As a further consequence, under certain conditions, U maps the exponent to the *magic wand* $X \multimap Y$, which is the right adjoint of the separated product.

We apply these connections between nominal sets and renaming sets in the context of automata theory. In terms of expressivity, nominal automata and the more classical register automata are equivalent, but nominal automata have appealing properties that register automata lack, such as unique minimal automata [2]. Unfortunately, moving from register automata to nominal automata can lead to an exponential blow-up in the number of states.¹

As a motivating example, we consider a language modelling an n -bounded FIFO queue. The input alphabet is given by $\Sigma = \{\text{Put}(a) \mid a \in \mathbb{A}\} \cup \{\text{Pop}\}$, and the output alphabet by $O = \mathbb{A} \cup \{\perp\}$ (here, \perp is a *null* value). The (generalised) language $L_n: \Sigma^* \rightarrow O$ maps a sequence of queue operations to the resulting top element when starting from the empty queue, or to \perp if this is undefined. The language L_n can be recognised by a nominal (Moore) automaton, but this requires an exponential number of states in n , as the automaton distinguishes internally between all possible equalities among elements in the queue [20].

Based on the observation that L_n is closed under substitutions, we can come up with a *linear* automata-theoretic representation. To this end, we define the new notion of *separated nominal automaton*, where the transition function is only defined for pairs of states and letters with a disjoint support (Section 4). Using the aforementioned categorical framework, we can go back and forth between languages from separated automata and languages which are closed under substitutions. In the FIFO example, the separated automaton obtained from the original nominal automaton has only $n + 2$ states, thus dramatically reducing the number of states. We expect that such a reduction is useful in many applications, such as active learning of register automata [20].

2 Monoid actions and nominal sets

In order to capture both the standard notion of nominal sets [22] and sets with more general renaming actions [10], we start by defining monoid actions.

► **Definition 2.1.** *Let $(M, \cdot, 1)$ be a monoid. An M -set is a set X together with a function $\cdot: M \times X \rightarrow X$ such that $1 \cdot x = x$ and $m \cdot (n \cdot x) = (m \cdot n) \cdot x$ for all $m, n \in M$ and $x \in X$. The function \cdot is called an M -action and $m \cdot x$ is often written by juxtaposition mx . A function $f: X \rightarrow Y$ between two M -sets is M -equivariant if $m \cdot f(x) = f(m \cdot x)$ for all $m \in M$ and $x \in X$. The class of M -sets together with equivariant maps forms a category $M\text{-Set}$.*

¹ Here, “number of states” refers to the number of orbits in the state space.

Let $\mathbb{A} = \{a, b, c, \dots\}$ be a countable infinite set of *atoms*. The two main instances of M considered in this paper are the monoid

$$\mathbf{Sb} = \{m: \mathbb{A} \rightarrow \mathbb{A} \mid m(a) \neq a \text{ for finitely many } a\}$$

of all (finite) substitutions (with composition as multiplication), and the monoid

$$\mathbf{Pm} = \{g \in \mathbf{Sb} \mid g \text{ is a bijection}\}$$

of all (finite) permutations. Since \mathbf{Pm} is a submonoid of \mathbf{Sb} , any \mathbf{Sb} -set is also a \mathbf{Pm} -set; and any \mathbf{Sb} -equivariant map is also \mathbf{Pm} -equivariant. This gives rise to a forgetful functor

$$U: \mathbf{Sb}\text{-Set} \rightarrow \mathbf{Pm}\text{-Set}. \quad (1)$$

The set \mathbb{A} is an \mathbf{Sb} -set by defining $m \cdot a = m(a)$. Given an M -set X , the set $\mathcal{P}(X)$ of subsets of X is an M -set, with the action defined by direct image.

For a \mathbf{Pm} -set X , the *orbit* of an element x is the set $\text{orb}(x) = \{g \cdot x \mid g \in \mathbf{Pm}\}$. We say X is *orbit-finite* if the set $\{\text{orb}(x) \mid x \in X\}$ is finite.

For any monoid M , the category $M\text{-Set}$ is symmetric monoidal closed. The product of two M -sets is given by the Cartesian product, with the action defined pointwise: $m \cdot (x, y) = (m \cdot x, m \cdot y)$. In $M\text{-Set}$, the exponent $X \rightarrow^M Y$ is given by the set $\{f: M \times X \rightarrow Y \mid f \text{ is equivariant}\}$.² The action on such an $f: M \times X \rightarrow Y$ is defined by $(m \cdot f)(n, x) = f(mn, x)$. A good introduction to the construction of the exponent is given by Simmons [28]. If M is a group, a simpler description of the exponent may be given, carried by the set of all functions $f: X \rightarrow Y$, with the action given by $(g \cdot f)(x) = g \cdot f(g^{-1} \cdot x)$.

2.1 Nominal M -sets

The notion of *nominal* set is usually defined w.r.t. a \mathbf{Pm} -action. Here, we use the generalisation to \mathbf{Sb} -actions from [10]. Throughout this section, let M denote a submonoid of \mathbf{Sb} .

► **Definition 2.2.** *Let X be an M -set, and $x \in X$ an element. A set $C \subset \mathbb{A}$ is an (M)-support of x if for all $m_1, m_2 \in M$ s.t. $m_1|_C = m_2|_C$ we have $m_1x = m_2x$. An M -set X is called *nominal* if every element x has a finite M -support.*

Nominal M -sets and equivariant maps form a full subcategory of $M\text{-Set}$, denoted by $M\text{-Nom}$. The M -set \mathbb{A} of atoms is nominal. The powerset $\mathcal{P}(X)$ of a nominal set is not nominal in general; the restriction to finitely supported elements is.

If M is a group, then the notion of support can be simplified by using inverses. To see this, first note that, given elements $g_1, g_2 \in M$, $g_1|_C = g_2|_C$ can equivalently be written as $g_2^{-1}g_1|_C = \text{id}|_C$. Second, the statement $g_1x = g_2x$ can be expressed as $g_2^{-1}g_1x = x$. Hence, C is a support iff $g|_C = \text{id}_C$ implies $gx = x$ for all g , which is the standard definition for nominal sets over a group [3, 22]. Surprisingly, a similar characterisation also holds for \mathbf{Sb} -sets [10]. Moreover, recall that every \mathbf{Sb} -set is also a \mathbf{Pm} -set; the associated notions of support coincide on nominal \mathbf{Sb} -sets, as shown by the following result. In particular, this means that the forgetful functor (1) restricts to $U: \mathbf{Sb}\text{-Nom} \rightarrow \mathbf{Pm}\text{-Nom}$.

► **Lemma 2.3.** [9, Theorem 4.8] *Let X be a nominal \mathbf{Sb} -set, $x \in X$, and $C \subset \mathbb{A}$. Then C is an \mathbf{Sb} -support of x iff it is a \mathbf{Pm} -support of x .*

² If we write a regular arrow \rightarrow , then we mean a map in the category. Exponent objects will always be denoted by annotated arrows.

► **Remark 2.4.** It is not true that any Pm-support is an Sb-support. The condition that X is nominal, in the above lemma, is crucial. Let $X = \mathbb{A} \cup \{*\}$ and define the following Sb-action: $m \cdot a = m(a)$ if m is injective, $m \cdot a = *$ if m is non-injective, and $m \cdot * = *$. This is a well-defined Sb-set, but is *not nominal*. Now consider $U(X)$; this is the Pm-set $\mathbb{A} \cup \{*\}$ with the natural action, which is a *nominal* Pm-set! In particular, as a Pm-set each element has a finite support, but as a Sb-set the supports are infinite.

This counterexample is similar to the “exploding nominal sets” in [9], but even worse behaved. We like to call them *nuclear sets*, since an element will collapse when hit by a non-injective map, no matter how far away the non-injectivity occurs.

For $M \in \{\mathbf{Sb}, \mathbf{Pm}\}$, any element $x \in X$ of a nominal M -set X has a *least* finite support (w.r.t. set inclusion). We denote the least finite support of an element $x \in X$ by $\text{supp}(x)$. Note that by Lemma 2.3, the set $\text{supp}(x)$ is independent of whether a nominal Sb-set X is viewed as an Sb-set or a Pm-set. The *dimension* of X is given by $\dim(X) = \max\{|\text{supp}(x)| \mid x \in X\}$, where $|\text{supp}(x)|$ is the cardinality of $\text{supp}(x)$.

We list some basic properties of nominal M -sets, which have known counterparts for the case that M is a group [3], and when $M = \mathbf{Sb}$ [10].

► **Lemma 2.5.** *Let X be an M -nominal set. If C supports an element $x \in X$, then $m \cdot C$ supports $m \cdot x$ for all $m \in M$. Moreover, any $g \in \mathbf{Pm}$ preserves least supports: $g \cdot \text{supp}(x) = \text{supp}(gx)$.*

The latter equality does not hold in general for a monoid M . For instance, the “exploding” nominal renaming sets [10] give counterexamples for $M = \mathbf{Sb}$.

► **Lemma 2.6.** *Given M -nominal sets X, Y and a map $f: X \rightarrow Y$, if f is M -equivariant and C supports an element $x \in X$, then C supports $f(x)$.*

The category $M\text{-Nom}$ is symmetric monoidal closed, with the product inherited from $M\text{-Set}$, thus simply given by Cartesian product. The exponent is given by the restriction of the exponent $X \rightarrow^M Y$ in $M\text{-Set}$ to the set of finitely supported functions, denoted by $X \rightarrow_{\text{fs}}^M Y$. This is similar to the exponents of nominal sets with 01-substitutions from [23].

► **Remark 2.7.** In [10] a different presentation of the exponent in $M\text{-Nom}$ is given, based on a certain extension of partial functions. We prefer the previous characterisation, as it is derived in a straightforward way from the exponent in $M\text{-Set}$.

2.2 Separated product

► **Definition 2.8.** *Let X and Y be Pm-nominal sets. Two elements $x \in X, y \in Y$ are called separated, denoted by $x \# y$, if there are disjoint sets $C_1, C_2 \subset \mathbb{A}$ such that C_1 supports x and C_2 supports y . The separated product of Pm-nominal sets X and Y is defined as*

$$X * Y = \{(x, y) \in X \times Y \mid x \# y\}.$$

We extend the separated product to the *separated power*, defined by $X^{(0)} = 1$ and $X^{(n+1)} = X^{(n)} * X$, and the *set of separated words* $X^{(*)} = \bigcup_i X^{(i)}$. The separated product is an equivariant subset $X * Y \subseteq X \times Y$. Consequently, we have equivariant projection maps $X * Y \rightarrow X$ and $X * Y \rightarrow Y$.

► **Example 2.9.** Two finite sets $C, D \in \mathcal{P}(\mathbb{A})$ are separated precisely when they are disjoint. An important example is the set $\mathbb{A}^{(*)}$ of separated words over the atoms: it consists of those words where all letters are distinct.

The separated product gives rise to another symmetric closed monoidal structure on $\mathbf{Pm-Nom}$, with 1 as unit, and the exponential object given by *magic wand* $X \multimap Y$. An explicit characterisation of $X \multimap Y$ is not needed in the remainder of this paper, but for a complete presentation we briefly recall the description from [26] (see also [22] and [5]). First, define a \mathbf{Pm} -action on the set of partial functions $f: X \multimap Y$ by $(g \cdot f)(x) = g \cdot f(g^{-1} \cdot x)$ if $f(g^{-1} \cdot x)$ is defined. Now, such a partial function $f: X \multimap Y$ is called *separating* if f is finitely supported, $f(x)$ is defined iff $f \# x$, and $\text{supp}(f) = \bigcup_{x \in \text{dom}(f)} \text{supp}(f(x)) \setminus \text{supp}(x)$. Finally, $X \multimap Y = \{f: X \multimap Y \mid f \text{ is separating}\}$. See [26] for a proof and explanation.

► **Remark 2.10.** The special case $\mathbb{A} \multimap Y$ coincides with $[\mathbb{A}]Y$, the set of *name abstractions* [22]. The latter is generalised to $[X]Y$ in [8]. In [5] it is shown that the coincidence $[X]Y \cong (X \multimap Y)$ only holds under strong assumptions (including that X is single-orbit).

► **Remark 2.11.** An analogue of the separated product does not seem to exist for nominal \mathbf{Sb} -sets. For instance, consider the set $\mathbb{A} \times \mathbb{A}$. As a \mathbf{Pm} -set, it has four equivariant subsets: \emptyset , $\{(a, a) \mid a \in \mathbb{A}\}$, $\mathbb{A} * \mathbb{A}$, and $\mathbb{A} \times \mathbb{A}$. However, the set $\mathbb{A} * \mathbb{A}$ is not an equivariant subset when considering $\mathbb{A} \times \mathbb{A}$ as an \mathbf{Sb} -set.

3 A monoidal construction from \mathbf{Pm} -sets to \mathbf{Sb} -sets

In this section, we provide a free construction, extending nominal \mathbf{Pm} -sets to nominal \mathbf{Sb} -sets. We use this as a basis to relate the separated product and exponent (in $\mathbf{Pm-Nom}$) to the product and exponent in $\mathbf{Sb-Nom}$. The main results are:

1. Theorem 3.6: the forgetful functor $U: \mathbf{Sb-Nom} \rightarrow \mathbf{Pm-Nom}$ has a left adjoint F ;
 2. Theorem 3.7: this F is monoidal: it maps separated products to products;
 3. Theorem 3.13 and Corollary 3.14: U maps the exponent object in $\mathbf{Sb-Nom}$ to the right adjoint \multimap of the separated product, if the domain has dimension smaller or equal to 1.
- Together, these results form the categorical infrastructure to relate nominal languages to separated languages and automata in Section 4.

► **Definition 3.1** (From [6]). *Given a \mathbf{Pm} -nominal set X , we define a nominal \mathbf{Sb} -set $F(X)$ as follows. Define the set*

$$F(X) = \{(m, x) \mid m \in \mathbf{Sb}, x \in X\} / \sim,$$

where \sim is the least equivalence relation containing:

$$(m, gx) \sim (mg, x), \tag{2}$$

$$(m, x) \sim (m', x) \quad \text{if } m|_C = m'|_C \text{ for a } \mathbf{Pm}\text{-support } C \text{ of } x, \tag{3}$$

for all $x \in X$, $m, m' \in \mathbf{Sb}$ and $g \in \mathbf{Pm}$. Note that $mg = m \circ g$, i.e., simply the monoid operation of \mathbf{Sb} . The equivalence class of a pair (m, x) is denoted by $[m, x]$. We define an \mathbf{Sb} -action on $F(X)$ as $n \cdot [m, x] = [nm, x]$.

Well-definedness is proved as part of Proposition 3.5 below. Informally, an equivalence class $[m, x] \in F(X)$ behaves “as if m acted on x .” The first equation (2) ensures compatibility with the \mathbf{Pm} -action on x , and the second equation (3) ensures that $[m, x]$ only depends the relevant part of m .

► **Example 3.2.** Here are a few examples of the application of F . We do not give direct proofs, but the first two will be treated more systematically later in this section (see Corollary 3.12). For the third, note that $\mathbb{A} \times \mathbb{A}$ consist of two orbits, $\mathbb{A} * \mathbb{A}$ and the diagonal $\{(a, a) \mid a \in \mathbb{A}\}$.

- $F(\mathbb{A}) \cong \mathbb{A}$.
- $F(\mathbb{A}^{(*)}) \cong \mathbb{A}^*$.
- $F(\mathbb{A} \times \mathbb{A}) \cong \mathbb{A}^2 + \mathbb{A}$.

The first example is also given in [6], where it is additionally shown that F does not preserve products. As we will see in Section 3.1, F does preserve a different monoidal structure, namely the separated product. The following characterisation of \sim is useful in proofs. This lemma is proven in [19].

► **Lemma 3.3.** *We have $(m_1, x_1) \sim (m_2, x_2)$ iff there is a permutation $g \in \text{Pm}$ such that $gx_1 = x_2$ and $m_1|_C = m_2g|_C$, for C some Pm-support of x_1 .*

► **Remark 3.4.** The first relation (2) in Definition 3.1 comes from the construction of “extension of scalars” in commutative algebra [1]. In that context, one has a ring homomorphism $f: A \rightarrow B$ and an A -module M and wishes to obtain a B -module. This is constructed by the tensor product $B \otimes_A M$ and it is here that the relation $(b, am) \sim (ba, m)$ is used (B is a right A -module via f).

In [6] it is stated that F is a functor, and a proof outline for well-definedness on arrows is given. Here we give a full proof, including well-definedness on objects.

► **Proposition 3.5.** *The construction F in Definition 3.1 extends to a functor*

$$F: \text{Pm-Nom} \rightarrow \text{Sb-Nom},$$

defined on an equivariant map $f: X \rightarrow Y$ by $F(f)([m, x]) = [m, f(x)] \in F(Y)$.

Proof. We first prove well-definedness and then the functoriality.

$F(X)$ is an Sb-set. To this end we check that the Sb-action is well-defined. Let $[m_1, x_1] = [m_2, x_2] \in F(X)$ and let $m \in \text{Sb}$. By Lemma 3.3, there is some permutation g such that $gx_1 = x_2$ and $m_1|_C = m_2g|_C$ for some support C of x_1 . By post-composition with m we get $mm_1|_C = mm_2g|_C$, which means (again by the lemma) that $[mm_1, x_1] = [mm_2, x_2]$. Thus $m[m_1, x_1] = m[m_2, x_2]$, which concludes well-definedness.

For associativity and unitality of the Sb-action, we note that it is directly defined by left multiplication of Sb which is associative and unital. This concludes that $F(X)$ is an Sb-set.

$F(X)$ is a nominal Sb set. Given an element $[m, x] \in F(X)$ and a Pm-support C of x , we will prove that $m \cdot C$ is an Sb-support for $[m, x]$. Suppose that we have $m_1, m_2 \in \text{Sb}$ such that $m_1|_{m \cdot C} = m_2|_{m \cdot C}$. By pre-composition with m we get $m_1m|_C = m_2m|_C$ and this leads us to conclude $[m_1m, x] = [m_2m, x]$. So $m_1[m, x] = m_2[m, x]$ as required.

Functoriality. Let $f: X \rightarrow Y$ be a Pm-equivariant map. To see that $F(f)$ is well-defined consider $[m_1, x_1] = [m_2, x_2]$. By Lemma 3.3, there is a permutation g such that $gx_1 = x_2$ and $m_1|_C = m_2g|_C$ for some support C of x_1 . Applying $F(f)$ gives on one hand $[m_1, f(x_1)]$ and on the other hand $[m_2, f(x_2)] = [m_2, f(gx_1)] = [m_2, gf(x_1)] = [m_2g, f(x_1)]$ (we used equivariance in the second step). Since $m_1|_C = m_2g|_C$ and f preserves supports we have $[m_2g, f(x_1)] = [m_1, f(x_1)]$.

For Sb-equivariance we consider both $n \cdot F(f)([m, x]) = n[m, f(x)] = [nm, f(x)]$ and $F(f)(n \cdot [m, x]) = F(f)([nm, x]) = [nm, f(x)]$. This shows $nF(f)([m, x]) = F(f)(n[m, x])$ and concludes that we have a map $F(f): F(X) \rightarrow F(Y)$.

Preservation of the identity function and composition follows from the definition. ◀

► **Theorem 3.6.** *The functor F is left adjoint to U :*

$$\begin{array}{ccc} & F & \\ \text{Pm-Nom} & \xrightarrow{\quad} & \text{Sb-Nom} \\ & \perp & \\ & U & \end{array}$$

Proof. We define, for every nominal set X , a map $\eta_X: X \rightarrow UF(X)$ with the necessary universal property: for every **Pm**-equivariant $f: X \rightarrow U(Y)$ there is a unique **Sb**-equivariant map $f^\sharp: FX \rightarrow Y$ such that $U(f^\sharp) \circ \eta_X = f$. Define $\eta_X(x) = [\text{id}, x]$. This is equivariant: $g \cdot \eta_X(x) = g[\text{id}, x] = [g, x] = [\text{id}, gx] = \eta_X(gx)$. Now, for $f: X \rightarrow U(Y)$, define $f^\sharp([m, x]) = m \cdot f(x)$ for $x \in X$ and $m \in \mathbf{Sb}$. Then $U(f^\sharp) \circ \eta_X(x) = f^\sharp([\text{id}, x]) = \text{id} \cdot f(x) = f(x)$.

For well-definedness of f^\sharp , consider $[m_1, x_1] = [m_2, x_2]$ (we have to prove that $m_1 \cdot f(x_1) = m_2 \cdot f(x_2)$). By Lemma 3.3, there is a $g \in \mathbf{Pm}$ such that $gx_1 = x_2$ and $m_2g|_C = m_1|_C$ for a **Pm**-support C of x_1 . Now C is also a **Pm**-support for $f(x)$ and hence it is an **Sb**-support of $f(x)$ (Lemma 2.3). Thus $m_2 \cdot f(x_2) = m_2 \cdot f(gx_1) = m_2g \cdot f(x_1) = m_1 \cdot f(x_1)$ (we use **Pm**-equivariance in the one but last step and **Sb**-support in the last step). For **Sb**-equivariance, we compute $n \cdot f^\sharp([m, x]) = nm \cdot f(x) = f^\sharp([nm, x]) = f^\sharp(n[m, x])$. For uniqueness, suppose $h: FX \rightarrow Y$ is such that $U(h) \circ \eta_X = f$, i.e., $h([\text{id}, x]) = f(x)$. Then $h([m, x]) = h(m[\text{id}, x]) = m \cdot h([\text{id}, x]) = m \cdot f(x) = m \cdot f^\sharp([\text{id}, x]) = f^\sharp(m \cdot [\text{id}, x]) = f^\sharp([m, x])$. ◀

The counit $\epsilon: FU(Y) \rightarrow Y$ is given by $\epsilon([m, x]) = m \cdot x$. For the inverse of $-^\sharp$, let $g: F(X) \rightarrow Y$ be an **Sb**-equivariant map; then $g^\flat: X \rightarrow U(Y)$ is given by $g^\flat(x) = g([\text{id}, x])$. Note that the unit η is a **Pm**-equivariant map, hence it preserves supports (i.e., any support of x also supports $[\text{id}, x]$). This also means that if C is a support of x , then $m \cdot C$ is a support of $[m, x]$ (by Lemma 2.5).

3.1 On (separated) products

The functor F not only preserves coproducts, being a left adjoint, but it also maps the separated product to products:

► **Theorem 3.7.** *The functor F is strong monoidal, from the monoidal category $(\mathbf{Pm}\text{-Nom}, *, 1)$ to $(\mathbf{Sb}\text{-Nom}, \times, 1)$. In particular, the map p given by*

$$p = \langle F(\pi_1), F(\pi_2) \rangle: F(X * Y) \rightarrow F(X) \times F(Y)$$

is an isomorphism, natural in X and Y .

Proof. We prove that p is an isomorphism. It suffices to show that p is injective and surjective. Note that $p([m, (x, y)]) = ([m, x], [m, y])$.

Surjectivity. Let $([m_1, x], [m_2, y])$ be an element of $F(X) \times F(Y)$. We take an element $y' \in Y$ such that $y' \# x$ and $y' = gy$ for some $g \in \mathbf{Pm}$. Now we have an element $(x, y') \in X * Y$. By Lemma 2.5, we have $\text{supp}(y') = g \text{supp}(y)$. Define the map

$$m(a) = \begin{cases} m_1(a) & \text{if } a \in \text{supp}(x) \\ m_2(g^{-1}(a)) & \text{if } a \in \text{supp}(y') \\ a & \text{otherwise.} \end{cases}$$

(Observe that $\text{supp}(x) \# \text{supp}(y')$, so the cases are not overlapping.) The map m is an element of **Sb**. Now consider the element $z = [m, (x, y')] \in F(X * Y)$. Applying p to z gives the element $([m, x], [m, y'])$. First, we note that $[m, x] = [m_1, x]$ by the definition of m . Second, we show that $[m, y'] = [m_2, y]$. Observe that $mg|_{\text{supp}(y)} = m_2|_{\text{supp}(y)}$ by definition of m . Since $\text{supp}(y)$ is a support of y , we have $[mg, y] = [m_2, y]$, and since $[mg, y] = [m, gy] = [m, y']$ we are done. Hence $p([m, (x, y')]) = ([m, x], [m, y']) = ([m_1, x], [m_2, y])$, so p is surjective.

Injectivity. Let $[m_1, (x_1, y_1)]$ and $[m_2, (x_2, y_2)]$ be two elements. Suppose that they are mapped to the same element, i.e., $[m_1, x_1] = [m_2, x_2]$ and $[m_1, y_1] = [m_2, y_2]$. Then there are permutations g_x, g_y such that $x_2 = g_x x_1$ and $y_2 = g_y y_1$. Moreover, let $C = \text{supp}(x_1)$ and

$D = \text{supp}(y_1)$; then we have $m_1|_C = m_2g_x|_C$ and $m_1|_D = m_2g_y|_D$. In order to show the two original elements are equal, we have to provide a single permutation g . Define for, $z \in C \cup D$,

$$g_0(z) = \begin{cases} g_x(z) & \text{if } z \in C \\ g_y(z) & \text{if } z \in D. \end{cases}$$

(Again, C and D are disjoint.) The function g_0 is injective since the least supports of x_2 and y_2 are disjoint. Hence g_0 defines a local isomorphism from $C \cup D$ to $g_0(C \cup D)$. By homogeneity [22], the map g_0 extends to a permutation $g \in \text{Pm}$ with $g(z) = g_x(z)$ for $z \in C$ and $g(z) = g_y(z)$ for $z \in D$. In particular we get $(x_2, y_2) = g(x_1, y_1)$. We also obtain $m_1|_{C \cup D} = m_2g|_{C \cup D}$. Thus $[m_1, (x_1, y_1)] = [m_2, (x_2, y_2)]$, and so the map p is injective.

Unit and coherence. To show that F preserves the unit, we note that $[m, 1] = [m', 1]$ for every $m, m' \in \text{Sb}$, as the empty set supports 1 and so $m|_\emptyset = m'|_\emptyset$ vacuously holds. We conclude $F(1)$ is a singleton. \blacktriangleleft

Since F also preserves coproducts (being a left adjoint), we obtain that F maps the set of separated words to the set of all words.

► **Corollary 3.8.** *For any Pm-nominal set X , we have $F(X^{(*)}) \cong (FX)^*$.*

As we will show below, the functor F preserves the set \mathbb{A} of atoms. This is an instance of a more general result about preservation of one-dimensional objects.

► **Proposition 3.9.** *The functors F and U are equivalences on ≤ 1 -dimensional objects. Concretely, for $X \in \text{Pm-Nom}$ and $Y \in \text{Sb-Nom}$:*

1. *If $\dim(X) \leq 1$, then the unit $\eta: X \rightarrow UF(X)$ is an isomorphism.*
2. *If $\dim(Y) \leq 1$, then the co-unit $\epsilon: FU(Y) \rightarrow Y$ is an isomorphism.*

In the proof, we will use the following property of Sb -sets with dimension ≤ 1 .

► **Lemma 3.10.** *Let Y be a nominal Sb -set. If an element $y \in Y$ is supported by a singleton set (or even the empty set), then*

$$\{my \mid m \in \text{Sb}\} = \{gy \mid g \in \text{Pm}\}.$$

Proof. Let $y \in Y$ be supported by $\{a\}$ and let $m \in \text{Sb}$. Now consider $b = m(a)$ and the bijection $g = (ab)$. Now $m|_{\{a\}} = g|_{\{a\}}$, meaning that $my = gy$. So the set $\{my \mid m \in \text{Sb}\}$ is contained in $\{gy \mid g \in \text{Pm}\}$. The inclusion the other way is trivial, which means $\{my \mid m \in \text{Sb}\} = \{gy \mid g \in \text{Pm}\}$. \blacktriangleleft

Proof of Proposition 3.9. It is easy to see that $\eta: x \mapsto [\text{id}, x]$ is injective. Now to see that η is surjective, let $[m, x] \in UF(X)$ and consider a support $\{a\}$ of x (this is a singleton or empty since $\dim(X) \leq 1$). Let $b = m(a)$ and consider the swap $g = (ab)$. Now $[m, x] = [mg^{-1}, gx]$ and note that $\{b\}$ supports gx and $mg^{-1}|_{\{b\}} = \text{id}|_{\{b\}}$. We conclude with $[mg^{-1}, gx] = [\text{id}, gx]$, which implies that gx is the preimage of $[m, x]$. Hence η is an isomorphism.

To see that $\epsilon: [m, y] \mapsto my$ is surjective, just consider $m = \text{id}$. To see that ϵ is injective, let $[m, y], [m', y'] \in FU(Y)$ be two elements such that $my = m'y'$. Then by using Lemma 3.10 we find $g, g' \in \text{Pm}$ such that $gy = my = m'y' = g'y'$. This means that y and y' are in the same orbit (of $U(Y)$) and have the same dimension. Case 1: $\text{supp}(y) = \text{supp}(y') = \emptyset$, then $[m, y] = [\text{id}, y] = [\text{id}, y'] = [m', y']$. Case 2: $\text{supp}(y) = \{a\}$ and $\text{supp}(y') = \{b\}$, then $\text{supp}(gy) = \{g(a)\}$ (Lemma 2.5). In particular we have that m and g map a to $c = g(a)$, likewise m' and g' map b to c . Now $[m, y] = [m, g^{-1}g'y'] = [mg^{-1}g', y'] = [m', y']$, where we used $mg^{-1}g(b) = c = m'(b)$ in the last step. Thus ϵ is injective and hence an isomorphism. \blacktriangleleft

By Proposition 3.9, we may consider the set \mathbb{A} as both **Sb**-set and **Pm**-set (abusing notation). And we get an isomorphism $F(\mathbb{A}) \cong \mathbb{A}$ of nominal **Sb**-sets. To appreciate the above results, we give a concrete characterisation of one-dimensional nominal sets:

► **Lemma 3.11.** *Let X be a nominal M -set, for $M \in \{\mathbf{Sb}, \mathbf{Pm}\}$. Then $\dim(X) \leq 1$ iff there exist discrete³ sets Y and I such that $X \cong Y + \coprod_I \mathbb{A}$.*

In particular, the one-dimensional objects include the alphabets used for *data words*, consisting of a product $S \times \mathbb{A}$ of a discrete set S of action labels and the set of atoms. These alphabets are very common in the study of register automata (see, e.g., [13]).

By the above and Theorem 3.7, F maps separated powers of \mathbb{A} to powers, and the set of separated words over \mathbb{A} to the **Sb**-set of words over \mathbb{A} .

► **Corollary 3.12.** *We have $F(\mathbb{A}^{(n)}) \cong \mathbb{A}^n$ and $F(\mathbb{A}^{(*)}) \cong \mathbb{A}^*$.*

3.2 On exponents

We have described how F and U interact with (separated) products. Next, we establish a relationship between the magic wand (\multimap) and the exponent of nominal **Sb**-sets ($\rightarrow_{\text{fs}}^{\text{Sb}}$). These results on exponents will be useful in Section 4.1, where we discuss automata using coalgebras.

► **Theorem 3.13.** *The sets $X \multimap U(Y)$ and $U(F(X) \rightarrow_{\text{fs}}^{\text{Sb}} Y)$ are naturally isomorphic as nominal **Pm**-sets.*

Proof. We have the composite adjunctions

$$F \circ (X * -) \dashv (X \multimap -) \circ U \quad \text{and} \quad (FX \times -) \circ F \dashv U \circ (FX \rightarrow_{\text{fs}}^{\text{Sb}} -).$$

Theorem 3.7 gives a natural isomorphism between the left adjoints. Hence, the right adjoints are also isomorphic, which is the desired result. ◀

Note that this theorem gives an alternative characterisation of the magic wand in terms of the exponent in **Sb-Nom**, if the codomain is $U(Y)$. Moreover, for a 1-dimensional object X in **Sb-Nom**, we obtain the following special case of the theorem (using the co-unit isomorphism from Proposition 3.9):

► **Corollary 3.14.** *Let X, Y be nominal **Sb**-sets. For 1-dimensional X , the nominal **Pm**-set $U(X) \multimap U(Y)$ is naturally isomorphic to $U(X \rightarrow_{\text{fs}}^{\text{Sb}} Y)$.*

► **Remark 3.15.** The set $\mathbb{A} \multimap U(X)$ coincides with the atom abstraction $[\mathbb{A}]UX$ (Remark 2.10). Hence, as a special case of Corollary 3.14, we recover [10, Theorem 34], which states a bijective correspondence between $[\mathbb{A}]UX$ and $U(\mathbb{A} \rightarrow_{\text{fs}}^{\text{Sb}} X)$.

4 Nominal and separated automata

In this section, we study nominal (Moore) automata, which recognise languages over infinite alphabets. After recalling the basic definitions, we introduce a new variant of automata based on the separating product, which we call *separated nominal automata*. These automata

³ Any set Z can be equipped with a trivial action $m \cdot x = x$, which makes Z a nominal set. Such sets are called discrete.

31:10 Separation and Renaming in Nominal Sets

represent nominal languages which are **Sb**-equivariant, essentially meaning they are closed under substitution. Our main result is that, if a “classical” nominal automaton (over **Pm**) represents a language L which is **Sb**-equivariant, then L can also be represented by a separated nominal automaton. The latter can be exponentially smaller (in number of orbits) than the original automaton, as we show in a concrete example.

► **Remark 4.1.** We will work with a general output set O instead of just acceptance. The reason for this is that **Sb**-equivariant functions $L: \mathbb{A}^{(*)} \rightarrow 2$ are not very interesting: they are defined purely by the length of the input. By using more general output O , we may still capture interesting behaviour, e.g., the language in Example 4.3.

► **Definition 4.2.** Let Σ, O be **Pm**-sets, called *input/output alphabet* respectively.

- A (**Pm**)-nominal language is an equivariant map of the form $L: \Sigma^* \rightarrow O$.
- A nominal (Moore) automaton $\mathcal{A} = (Q, \delta, o, q_0)$ consists of a nominal set of states Q , an equivariant transition function $\delta: Q \times \Sigma \rightarrow Q$, an equivariant output function $o: Q \rightarrow O$, and an initial state $q_0 \in Q$ with an empty support.
- The language semantics is the map $l: Q \times \Sigma^* \rightarrow O$, defined inductively by

$$l(x, \varepsilon) = o(x), \quad l(x, aw) = l(\delta(x, a), w)$$

for all $x \in Q$, $a \in \Sigma$ and $w \in \Sigma^*$.

- For $l^\flat: Q \rightarrow (\Sigma^* \rightarrow_{\text{fs}}^{\text{Pm}} O)$ the transpose of l , we have that $l^\flat(q_0): \Sigma^* \rightarrow O$ is equivariant; this is called the language accepted by \mathcal{A} .

Note that the language accepted by an automaton can equivalently be characterised by considering paths through the automaton from the initial state.

If the state space Q and the alphabets Σ, O are orbit finite, this allows us to run algorithms (reachability, minimisation, etc.) on such automata [3], but there is no need to assume this for now. For an automaton $\mathcal{A} = (Q, \delta, o, q_0)$, we define the set of *reachable states* as the least set $R(\mathcal{A}) \subseteq Q$ such that $q_0 \in R(\mathcal{A})$ and for all $x \in R(\mathcal{A})$ and $a \in \Sigma$, $\delta(x, a) \in R(\mathcal{A})$.

► **Example 4.3.** We model a bounded FIFO queue of size n as a nominal Moore automaton, explicitly handling the data in the automaton structure.⁴ The input alphabet Σ and output alphabet O are as follows:

$$\Sigma = \{\text{Put}(a) \mid a \in \mathbb{A}\} \cup \{\text{Pop}\}, \quad O = \mathbb{A} \cup \{\perp\}.$$

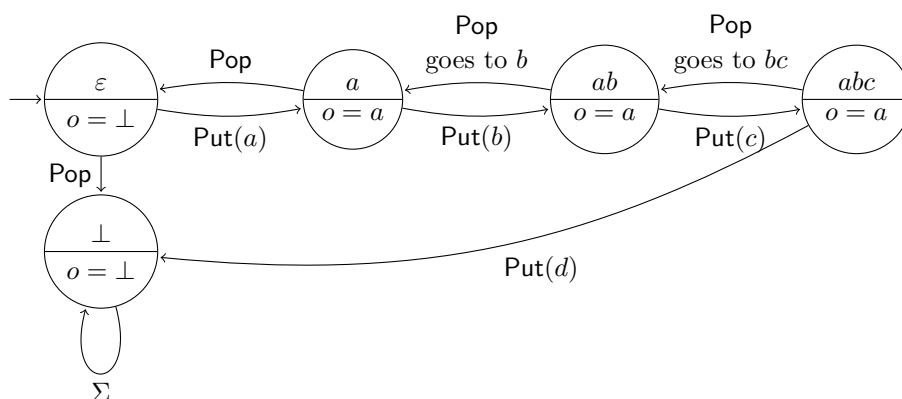
The input alphabet encodes two actions: putting a new value on the queue and popping a value. The output is either a value (the front of the queue) or \perp if the queue is empty. A queue of size n is modelled by the automaton (Q, δ, o, q_0) defined as follows.

$$Q = \mathbb{A}^{\leq n} \cup \{\perp\} \quad q_0 = \varepsilon \quad o(a_1 \dots a_k) = \begin{cases} a_1 & \text{if } k \geq 1 \\ \perp & \text{otherwise} \end{cases}$$

$$\delta(a_1 \dots a_k, \text{Put}(b)) = \begin{cases} a_1 \dots a_k b & \text{if } k < n \\ \perp & \text{otherwise} \end{cases} \quad \delta(\perp, x) = \perp$$

$$\delta(a_1 \dots a_k, \text{Pop}) = \begin{cases} a_2 \dots a_k & \text{if } k > 0 \\ \perp & \text{otherwise} \end{cases}$$

⁴ We use a reactive version of the queue data structure, which slightly differs from the versions in [20, 13].



■ **Figure 1** The FIFO automaton from Example 4.3 with $n = 3$. The right-most state consists of five orbits as we can take a, b, c distinct, all the same, or two of them equal in three different ways. Consequently, the complete state space has ten orbits. The output of each state is denoted in the lower part.

The automaton is depicted in Figure 1 for the case $n = 3$. The language accepted by this automaton assigns to a word w the first element of the queue after executing the instructions in w from left to right, and \perp if the input is ill-behaved, i.e., **Pop** is applied to an empty queue or **Put**(a) to a full queue.

► **Definition 4.4.** Let Σ, O be Pm-sets. A separated language is an equivariant map of the form $\Sigma^{(*)} \rightarrow O$. A separated automaton $\mathcal{A} = (Q, \delta, o, q_0)$ consists of Q, o and q_0 defined as in a nominal automaton, and an equivariant transition function $\delta: Q * \Sigma \rightarrow Q$.

The separated language semantics of such an automaton \mathcal{A} is given by the function $s: Q * \Sigma^{(*)} \rightarrow O$, defined inductively by

$$s(x, \varepsilon) = o(x), \quad s(x, aw) = s(\delta(x, a), w)$$

for all $x \in Q, a \in \Sigma$ and $w \in \Sigma^{(*)}$ such that $x \# aw$ and $a \# w$.

Let $s^b: Q \rightarrow (\Sigma^{(*)} * O)$ be the transpose of s . Then $s^b(q_0): \Sigma^{(*)} \rightarrow O$ corresponds to a separated language; this is called the separated language accepted by \mathcal{A} .

By definition of the separated product, the transition function is only defined on a state x and letter $a \in \Sigma$ if $x \# a$. In Example 4.10 below, we describe the bounded FIFO as a separated automaton, and describe its accepted language.

First, we show how the language semantics of separated nominal automata extends to a language over all words, provided that both the input alphabet Σ and the output alphabet O are Sb-sets.

► **Definition 4.5.** Let Σ and O be nominal Sb-sets. An Sb-equivariant function $L: \Sigma^* \rightarrow O$ is called an Sb-language.

Notice the difference between an Sb-language $L: \Sigma^* \rightarrow O$ and a Pm-language $L': (U\Sigma)^* \rightarrow U(O)$. They are both functions from Σ^* to O , but the latter is only Pm-equivariant, while the former satisfies the stronger property of Sb-equivariance. Languages over separated words, and Sb-languages, are connected as follows.

31:12 Separation and Renaming in Nominal Sets

► **Proposition 4.6.** *Suppose Σ, O are both nominal **Sb**-sets, and suppose $\dim(\Sigma) \leq 1$. There is a one-to-one correspondence*

$$\frac{S: (U\Sigma)^{(*)} \rightarrow UO \quad \text{Pm-equivariant}}{\bar{S}: \Sigma^* \rightarrow O \quad \text{Sb-equivariant}}$$

between separated languages and **Sb**-nominal languages. From \bar{S} to S , this is given by application of the forgetful functor and restricting to the subset of separated words.

For the converse direction, given $w = a_1 \dots a_n \in \Sigma^*$, let $b_1, \dots, b_n \in \Sigma$ such that $w \# b_i$ for all i , and $b_i \# b_j$ for all i, j with $i \neq j$. Define $m \in \text{Sb}$ by

$$m(a) = \begin{cases} a_i & \text{if } a = b_i \text{ for some } i \\ a & \text{otherwise} \end{cases}$$

Then $\bar{S}(a_1 a_2 a_3 \dots a_n) = m \cdot S(b_1 b_2 b_3 \dots b_n)$.

Proof. There is the following chain of one-to-one correspondences, from the results of the previous section:

$$\frac{\frac{(U\Sigma)^{(*)} \rightarrow UO}{F(U\Sigma)^{(*)} \rightarrow O} \text{ by Theorem 3.6}}{(FU\Sigma)^* \rightarrow O} \text{ by Corollary 3.8}}{\Sigma^* \rightarrow O} \text{ by Proposition 3.9}$$

◀

Thus, every separated automaton over $U(\Sigma), U(O)$ gives rise to an **Sb**-language \bar{S} , corresponding to the language S accepted by the automaton.

Any nominal automaton \mathcal{A} restricts to a separated automaton, formally described in Definition 4.7. It turns out that if the (Pm)-language accepted by \mathcal{A} is actually an **Sb**-language, then the restricted automaton already represents this language, as the extension \bar{S} of the associated separated language S (Proposition 4.8). Hence, in such a case, the restricted separated automaton suffices to describe the language of \mathcal{A} .

► **Definition 4.7.** *Let $i: Q * U(\Sigma) \hookrightarrow Q \times U(\Sigma)$ be the natural inclusion map. A nominal automaton $\mathcal{A} = (Q, \delta, o, q_0)$ induces a separated automaton \mathcal{A}_* , by setting $\mathcal{A}_* = (Q, \delta \circ i, o, q_0)$.*

► **Proposition 4.8.** *Suppose Σ, O are both **Sb**-sets, and suppose $\dim(\Sigma) \leq 1$. Let $L: (U\Sigma)^* \rightarrow UO$ be the Pm-nominal language accepted by a nominal automaton \mathcal{A} , and suppose L is **Sb**-equivariant. Let S be the separated language accepted by \mathcal{A}_* . Then $L = U(\bar{S})$.*

Proof. It follows from the one-to-one correspondence in Proposition 4.6: on the bottom there are two languages (L and $U(\bar{S})$), while there is only the restriction of L on the top. We conclude that $L = U(\bar{S})$. ◀

As we will see in Example 4.10, separated automata allow us to represent **Sb**-languages in a smaller way than nominal automata. Given a nominal automaton \mathcal{A} , a smaller separated automaton can be obtained by computing the reachable part of the restriction \mathcal{A}_* . The reachable part is defined similarly (but only where δ is defined) and also denoted by $R(\mathcal{A}_*)$.

► **Lemma 4.9.** *For any nominal automaton \mathcal{A} , we have $R(\mathcal{A}_*) \subseteq R(\mathcal{A})$.*

The converse inclusion of the above proposition does certainly not hold, as shown by the following example.

► **Example 4.10.** Let \mathcal{A} be the automaton modelling a bounded FIFO queue (for some n), from Example 4.3. The Pm-nominal language L accepted by \mathcal{A} is Sb-equivariant: it is closed under application of arbitrary substitutions.

The separated automaton \mathcal{A}_* is given simply by restricting the transition function to $Q * \Sigma$, i.e., a Put(a)-transition from a state $w \in Q$ exists only if a does not occur in w . The separated language S accepted by this new automaton is the restriction of the nominal language of \mathcal{A} to separated words. By Proposition 4.8, we have $L = U(\overline{S})$. Hence, the separated automaton \mathcal{A}_* represents L , essentially by closing the associated separated language S under all substitutions.

The *reachable* part of \mathcal{A}_* is given by

$$R_{\mathcal{A}_*} = \mathbb{A}^{(\leq n)} \cup \{\perp\}.$$

Clearly, restricting \mathcal{A}_* to the reachable part does not affect the accepted language. However, while the original state space Q has exponentially many orbits in n , $R_{\mathcal{A}_*}$ has only $n + 2$ orbits! Thus, taking the reachable part of $R_{\mathcal{A}_*}$ yields a separated automaton which represents the FIFO language L in a much smaller way than the original automaton.

4.1 Separated automata: coalgebraic perspective

Nominal automata and separated automata can be presented as *coalgebras* on the category of Pm-nominal sets. In this section we revisit the above results from this perspective, and generalise from (equivariant) languages to finitely supported languages. In particular, we retrieve the extension from separated languages to Sb-languages, by establishing Sb-languages as a final separated automaton. The latter result follows by instantiating a well-known technique for lifting adjunctions to categories of coalgebras, using the results of Section 3. We assume familiarity with the theory of coalgebras, see, e.g., [14, 25].

► **Definition 4.11.** Let M be a submonoid of Sb, and let Σ, O be nominal M -sets, referred to as the input and output alphabet respectively. Define the functor $B_M: M\text{-Nom} \rightarrow M\text{-Nom}$ by $B_M(X) = O \times (\Sigma \rightarrow_{\text{fs}}^M X)$. An (M)-nominal (Moore) automaton is a B_M -coalgebra.

A B_M -coalgebra can be presented as a nominal set Q together with the pairing

$$\langle o, \delta^b \rangle: Q \rightarrow O \times (\Sigma \rightarrow_{\text{fs}}^M Q)$$

of an equivariant *output* function $o: Q \rightarrow O$, and (the transpose of) an equivariant *transition* function $\delta: Q \times \Sigma \rightarrow Q$. In case $M = \text{Pm}$, this coincides with the automata of Definition 4.2, omitting initial states. The language semantics is generalised accordingly, as follows. Given such a B_M -coalgebra $(Q, \langle o, \delta^b \rangle)$, the *language semantics* $l: Q \times \Sigma^* \rightarrow O$ is given by

$$l(x, \varepsilon) = o(x), \quad l(x, aw) = l(\delta(x, a), w) \tag{4}$$

for all $x \in Q$, $a \in \Sigma$ and $w \in \Sigma^*$.

► **Proposition 4.12.** Let M be a submonoid of Sb, let Σ, O be nominal M -sets. The nominal M -set $\Sigma^* \rightarrow_{\text{fs}}^M O$ extends to a final B_M -coalgebra $(\Sigma^* \rightarrow_{\text{fs}}^M O, \zeta)$, such that the unique homomorphism from a given B_M -coalgebra is the transpose l^b of the language semantics (4).

A *separated automaton* (Definition 4.4, without initial states) corresponds to a coalgebra for the functor $B_*: \text{Pm-Nom} \rightarrow \text{Pm-Nom}$ given by $B_*(X) = O \times (\Sigma \multimap X)$. The separated language semantics arises by finality.

31:14 Separation and Renaming in Nominal Sets

► **Proposition 4.13.** *The set $\Sigma^{(*)} \multimap O$ is the carrier of a final B_* -coalgebra, such that the unique coalgebra homomorphism from a given B_* -coalgebra $(Q, \langle o, \delta \rangle)$ is the transpose s^\flat of the separated language semantics $s: Q \multimap \Sigma^{(*)} \rightarrow O$ (Definition 4.4).*

Next, we provide an alternative description of the final B_* -coalgebra which assigns \mathbf{Sb} -nominal languages to states of separated nominal automata. The essence is to obtain a final B_* -coalgebra from the final $B_{\mathbf{Sb}}$ -coalgebra. In order to prove this, we use a technique to lift adjunctions to categories of coalgebras. This technique occurs regularly in the coalgebraic study of automata [15, 17, 16].

► **Theorem 4.14.** *Let Σ be a \mathbf{Pm} -set, and O an \mathbf{Sb} -set. Define B_* and $B_{\mathbf{Sb}}$ accordingly, as $B_*(X) = UO \times (\Sigma \multimap X)$ and $B_{\mathbf{Sb}}(X) = O \times (F\Sigma \rightarrow_{\mathbf{fs}}^{\mathbf{Sb}} X)$. There is an adjunction*

$$\text{CoAlg}(B_*) \begin{array}{c} \xrightarrow{\bar{F}} \\ \perp \\ \xleftarrow{\bar{U}} \end{array} \text{CoAlg}(B_{\mathbf{Sb}})$$

where \bar{F} and \bar{U} coincide with F and U respectively on carriers.

Proof. There is a natural isomorphism $\lambda: B_*U \Rightarrow UB_{\mathbf{Sb}}$ given by

$$\lambda: UO \times (\Sigma \multimap UX) \xrightarrow{\text{id} \times \phi} UO \times U(F\Sigma \rightarrow_{\mathbf{fs}}^{\mathbf{Sb}} X) \xrightarrow{\cong} U(O \times (F\Sigma \rightarrow_{\mathbf{fs}}^{\mathbf{Sb}} X)),$$

where ϕ is the isomorphism from Theorem 3.13 and the isomorphism on the right comes from U being a right adjoint. The result now follows from Theorem 2.14 in [12]. In particular, $\bar{U}(X, \gamma) = (UX, \lambda^{-1} \circ U(\gamma))$. ◀

Since right adjoints preserve limits, and final objects in particular, we obtain the following, giving semantics of separated automata through finality.

► **Corollary 4.15.** *Let $((F\Sigma)^* \rightarrow_{\mathbf{fs}}^{\mathbf{Sb}} O, \zeta)$ be the final $B_{\mathbf{Sb}}$ -coalgebra (Proposition 4.12). Then the B_* -coalgebra $\bar{U}(\Sigma^* \rightarrow_{\mathbf{fs}}^{\mathbf{Sb}} O, \zeta)$ is final and carried by the set $(F\Sigma)^* \rightarrow_{\mathbf{fs}}^{\mathbf{Sb}} O$ of \mathbf{Sb} -nominal languages.*

5 Relation to (pre)sheaf categories

Fiore and Turi described a similar adjunction between certain presheaf categories [7]. However, Staton describes in his thesis that the usage of presheaves allows for many degenerate models and one should look at sheaves instead [29]. The category of sheaves is equivalent to the category of nominal sets. We will describe these equivalences in this section.

Let us define the index categories \mathbb{I} and \mathbb{F} . Both categories have finite subsets $C \subset \mathbb{A}$ as objects. The morphisms in \mathbb{I} are all injective functions between those sets, and for \mathbb{F} we take all functions. The presheaves Fiore and Turi considered are $\mathbf{Set}^{\mathbb{I}}$ and $\mathbf{Set}^{\mathbb{F}}$. The interpretation of an object $X \in \mathbf{Set}^{\mathbb{I}}$ is that $X(C)$ is the set of elements supported by C . Although very similar to nominal sets, the categories are not equivalent. The inclusion $\mathbb{I} \subseteq \mathbb{F}$ induces a forgetful functor $\mathbf{Set}^{\mathbb{F}} \rightarrow \mathbf{Set}^{\mathbb{I}}$. It has a left adjoint, which can be defined by a Kan extension [7].

The subcategory of functors $\mathbb{I} \rightarrow \mathbf{Set}$ which preserve pullbacks is a sheaf category $\mathbf{Sh}(\mathbb{I})$.⁵ (For the precise sheaf conditions, see Staton's thesis [29].) The category $\mathbf{Sh}(\mathbb{I})$ is equivalent to \mathbf{Nom} . Similarly, there is a sheaf category $\mathbf{Sh}(\mathbb{F}) \subseteq \mathbf{Set}^{\mathbb{F}}$ and Staton has shown that the adjunction $\mathbf{Set}^{\mathbb{I}} \rightleftarrows \mathbf{Set}^{\mathbb{F}}$ restricts to an adjunction $\mathbf{Sh}(\mathbb{I}) \rightleftarrows \mathbf{Sh}(\mathbb{F})$.

⁵ We use the notation from [29], since we only deal with covariant functors here.

How does this compare to the adjunction described in this paper? Staton defines a category of nominal sets with a substitution operator, defined by certain axioms. This fits in the theory of universal algebra on nominal sets, as described by Kurz and Petrişan [18]. This category **NomSub** is equivalent to $\mathbf{Sh}(\mathbb{F})$, and most likely equivalent to **Sb-Nom** as defined here. These equivalences then give an abstract way of defining the adjunction from Theorem 3.6. Together with the fact that the separated product is a Day convolution (this fact is hinted at in [5] and [21]), one might obtain Theorem 3.7 from abstract reasoning alone (using the fact that both the left adjoint and the separated product are left Kan extensions). Nevertheless, we think that the explicit constructions and proofs given in this paper are useful, as they provide a concrete interpretation of the abstract concepts.

6 Related and future work

An interesting line of research is the generalisation to other symmetries by Bojańczyk et al. [3]. In particular, the total order symmetry is relevant, since it allows one to compare elements on their order, as often used in data words. In this case the symmetries are given by the group of all monotone bijections. Many results of nominal sets generalise to this symmetry. For monotone substitutions, however, the situation seems more subtle. For example, we note that a substitution which maps two values to the same value actually maps *all* the values in between to that value. Whether the adjunction from Theorem 3.6 generalises to other symmetries is left as future work.

This research was motivated by learning register automata. If we know a register automaton recognises an **Sb**-language, then we are better off learning a separated automaton instead of a nominal automaton. From the **Sb**-semantics of separated automata, it follows that we have a Myhill-Nerode theorem, which means that learning is feasible. We expect that this can be useful, since we can achieve an exponential reduction this way.

Bojańczyk et al. prove that nominal automata are equivalent to register automata in terms of expressiveness [3]. However, when translating from register automata with n states to nominal automata, we may get exponentially many orbits. This happens for instance in the FIFO automaton (Example 4.3). We have shown that the exponential blow-up is avoidable by using separated automata, for this example and in general for **Sb**-equivariant languages. Such languages come from register automata which manipulate data but where control flow does not depend on comparisons. This typically occurs in data structures.

An important open problem is whether the latter requirement can be relaxed, by adding separated transitions only locally in a nominal automaton. A possible step in this direction is to consider the monad $T = UF$ on **Pm-Nom** and incorporate it in the automaton model. We believe that this is the hypothesised “substitution monad” from [20]. The monad is monoidal (sending separated products to Cartesian products) and if X is an orbit-finite nominal set, then so is $T(X)$. This means that we can consider nominal T -automata and we can perhaps determine them using coalgebraic methods [27].

References

- 1 Michael Francis Atiyah and Ian G. MacDonald. *Introduction to commutative algebra*. Addison-Wesley-Longman, 1969.
- 2 Mikołaj Bojańczyk. *Slightly Infinite Sets*. Draft May 22, 2019. URL: <https://www.mimuw.edu.pl/~bojan/upload/main-9.pdf>.
- 3 Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *Logical Methods in Computer Science*, 10(3), 2014. doi:10.2168/LMCS-10(3:4)2014.

- 4 Mikołaj Bojańczyk, Bartek Klin, Sławomir Lasota, and Szymon Toruńczyk. Turing Machines with Atoms. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 183–192. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.24.
- 5 Ranald Clouston. Generalised Name Abstraction for Nominal Sets. In Frank Pfenning, editor, *Foundations of Software Science and Computation Structures - 16th International Conference, FOSSACS 2013. Proceedings*, volume 7794 of *Lecture Notes in Computer Science*, pages 434–449. Springer, 2013. doi:10.1007/978-3-642-37075-5_28.
- 6 Gilles Dowek and Murdoch James Gabbay. PNL to HOL: from the logic of nominal sets to the logic of higher-order functions. *Theor. Comput. Sci.*, 451:38–69, 2012. doi:10.1016/j.tcs.2012.06.007.
- 7 Marcelo P. Fiore and Daniele Turi. Semantics of Name and Value Passing. In *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*, pages 93–104. IEEE Computer Society, 2001. doi:10.1109/LICS.2001.932486.
- 8 Murdoch J. Gabbay. FM-HOL, a higher-order theory of names. *Thirty Five years of Automath, Heriot-Watt University, Edinburgh*, 2002.
- 9 Murdoch J. Gabbay. Nominal Renaming Sets. Technical report, Heriot-Watt University, 2007. URL: <https://www.gabbay.org/paper.html#nomrs-tr>.
- 10 Murdoch J. 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. Proceedings*, volume 5330 of *Lecture Notes in Computer Science*, pages 158–173. Springer, 2008. doi:10.1007/978-3-540-89439-1_11.
- 11 Murdoch J. 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.
- 12 Claudio Hermida and Bart Jacobs. Structural Induction and Coinduction in a Fibrational Setting. *Inf. Comput.*, 145(2):107–152, 1998. doi:10.1006/inco.1998.2725.
- 13 Malte Isberner, Falk Howar, and Bernhard Steffen. Learning register automata: from languages to program structures. *Machine Learning*, 96(1-2):65–98, 2014. doi:10.1007/s10994-013-5419-7.
- 14 Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*, volume 59 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2016. doi:10.1017/CB09781316823187.
- 15 Bart Jacobs, Alexandra Silva, and Ana Sokolova. Trace semantics via determinization. *J. Comput. Syst. Sci.*, 81(5):859–879, 2015. doi:10.1016/j.jcss.2014.12.005.
- 16 Henning Kerstan, Barbara König, and Bram Westerbaan. Lifting Adjunctions to Coalgebras to (Re)Discover Automata Constructions. In Marcello M. Bonsangue, editor, *Coalgebraic Methods in Computer Science - 12th IFIP WG 1.3 International Workshop, CMCS 2014, Colocated with ETAPS 2014, Grenoble, France, April 5-6, 2014, Revised Selected Papers*, volume 8446 of *Lecture Notes in Computer Science*, pages 168–188. Springer, 2014. doi:10.1007/978-3-662-44124-4_10.
- 17 Bartek Klin and Jurriaan Rot. Coalgebraic trace semantics via forgetful logics. *Logical Methods in Computer Science*, 12(4), 2016. doi:10.2168/LMCS-12(4:10)2016.
- 18 Alexander Kurz and Daniela Petrisan. On universal algebra over nominal sets. *Mathematical Structures in Computer Science*, 20(2):285–318, 2010. doi:10.1017/S0960129509990399.
- 19 Joshua Moerman and Jurriaan Rot. Separation and Renaming in Nominal Sets. *CoRR*, abs/1906.00763, 2019.
- 20 Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michał Szyrwelski. Learning nominal automata. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL*

- 2017, Paris, France, January 18-20, 2017, pages 613–625. ACM, 2017. URL: <http://dl.acm.org/citation.cfm?id=3009879>.
- 21 Peter W. O’Hearn. On bunched typing. *J. Funct. Program.*, 13(4):747–796, 2003. doi: 10.1017/S0956796802004495.
 - 22 Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2013.
 - 23 Andrew M. Pitts. Nominal Presentation of Cubical Sets Models of Type Theory. In Hugo Herbelin, Pierre Letouzey, and Matthieu Sozeau, editors, *20th International Conference on Types for Proofs and Programs, TYPES 2014, May 12-15, 2014, Paris, France*, volume 39 of *LIPICs*, pages 202–220. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. doi: 10.4230/LIPICs.TYPES.2014.202.
 - 24 Andrew M. Pitts. Nominal techniques. *SIGLOG News*, 3(1):57–72, 2016. doi:10.1145/2893582.2893594.
 - 25 Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.
 - 26 Ulrich Schöpp. *Names and binding in type theory*. PhD thesis, University of Edinburgh, UK, 2006. URL: <http://hdl.handle.net/1842/1203>.
 - 27 Alexandra Silva, Filippo Bonchi, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Generalizing determinization from automata to coalgebras. *Logical Methods in Computer Science*, 9(1), 2013. doi:10.2168/LMCS-9(1:9)2013.
 - 28 Harold Simmons. The topos of actions on a monoid. Unpublished manuscript, number 12N. URL: <http://www.cs.man.ac.uk/~hsimmons/DOCUMENTS/PAPERSandNOTES/Rsets.pdf>.
 - 29 Sam Staton. *Name-passing process calculi: operational models and structural operational semantics*. PhD thesis, University of Cambridge, Computer Laboratory, June 2007. URL: <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-688.pdf>.

Parity Games: Another View on Lehtinen’s Algorithm

Paweł Parys 

Institute of Informatics, University of Warsaw, Poland

parys@mimuw.edu.pl

Abstract

Recently, five quasi-polynomial-time algorithms solving parity games were proposed. We elaborate on one of the algorithms, by Lehtinen (2018).

Czerwiński et al. (2019) observe that four of the algorithms can be expressed as constructions of separating automata (of quasi-polynomial size), that is, automata that accept all plays decisively won by one of the players, and rejecting all plays decisively won by the other player. The separating automata corresponding to three of the algorithms are deterministic, and it is clear that deterministic separating automata can be used to solve parity games. The separating automaton corresponding to the algorithm of Lehtinen is nondeterministic, though. While this particular automaton can be used to solve parity games, this is not true for every nondeterministic separating automaton. As a first (more conceptual) contribution, we specify when a nondeterministic separating automaton can be used to solve parity games.

We also repeat the correctness proof of the Lehtinen’s algorithm, using separating automata. In this part, we prove that her construction actually leads to a faster algorithm than originally claimed in her paper: its complexity is $n^{O(\log n)}$ rather than $n^{O(\log d \cdot \log n)}$ (where n is the number of nodes, and d the number of priorities of a considered parity game), which is similar to complexities of the other quasi-polynomial-time algorithms.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Parity games, quasi-polynomial time, separating automata, good-for-games automata

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.32

Related Version A full version of the paper is available at <http://arxiv.org/abs/1910.03919>.

Funding Work supported by the National Science Centre, Poland (grant no. 2016/22/E/ST6/00041).

1 Introduction

Parity games have played a fundamental role in automata theory, logic, and their applications to verification and synthesis since early 1990’s. The algorithmic problem of finding the winner in parity games can be seen as the algorithmic backend to problems in automated verification and controller synthesis. It is polynomial-time equivalent to the emptiness problem for nondeterministic automata on infinite trees with parity acceptance conditions, and to the model-checking problem for modal μ -calculus [12]. Also, decision problems like validity or satisfiability for modal logics can be reduced to parity game solving. Moreover, it lies at the heart of algorithmic solutions to the Church’s synthesis problem [28]. The impact of parity games reaches relatively far areas of computer science, like Markov decision processes [13] and linear programming [16].

The problem of solving parity games has interesting complexity-theoretic status. It is a long-standing open question whether parity games can be solved in polynomial-time. Several results show that they belong to some classes “slightly above” polynomial time. Namely, deciding the winner of parity games was shown to be in $\text{NP} \cap \text{coNP}$ [12], and in $\text{UP} \cap \text{coUP}$ [20], while computing winning strategies is in PLS, PPAD, and even in their subclass CLS [10]. The



© Paweł Parys;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 32; pp. 32:1–32:15

Leibniz International Proceedings in Informatics



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

same holds for other kinds of games: mean-payoff games [33], discounted games, and simple stochastic games [8]; parity games, however, are the easiest among them, in the sense that there are polynomial-time reductions from parity games to the other kinds of games [20, 33], but no reductions in the opposite direction are known.

For almost three decades researchers were trying to cutback the complexity of solving parity games, which resulted in a series of algorithms, all of which were either exponential [32, 5, 30, 21, 31, 29, 1], or mildly subexponential [3, 23]. The next era came unexpectedly in 2017 with a breakthrough result of Calude, Jain, Khoussainov, Li, and Stephan [6] (see also [17, 24]), who designed an algorithm working in quasi-polynomial time (QPT for short). This invoked a series of QPT algorithms, which appeared soon after [22, 14, 25, 27].

Four of the QPT algorithms [6, 22, 14, 25], at first glance being quite different, actually proceed along a similar line – as observed by Bojańczyk and Czerwiński [4, Section 3] and Czerwiński et al. [9]. Namely, out of all the four algorithms one can extract a construction of a *PG separator*, that is, a safety automaton (nondeterministic in the case of Lehtinen [25], and deterministic in the other algorithms), which accepts all words encoding plays that are decisively won by one of the players (more precisely: plays consistent with some positional winning strategy), and rejects all words encoding plays in which the player loses (for plays that are won by the player, but not decisively, the automaton can behave arbitrarily). The PG separator does not depend at all on the game graph; it depends only on its size. Having a PG separator, it is not difficult to convert the original parity game into an equivalent safety game (by taking a “product” of the parity game and the PG separator), which can be solved easily – and all the four algorithms actually proceed this way, even if it is not stated explicitly that a PG separator is constructed. As shown in Czerwiński et al. [9] (see also Colcombet and Fijalkow [7] for another view on this proof), all PG separators have to look very similar: their states have to be leaves of some so-called universal tree; particular papers propose different constructions of these trees, and of the resulting PG separators (of quasi-polynomial size). Moreover, Czerwiński et al. [9] show a quasi-polynomial lower bound for the size of a PG separator. Let us also mention that, beside of the four algorithms, there is a fifth QPT algorithm [27] obtained by speeding up the Zielonka’s recursive algorithm [32]; this algorithm does not fit into the separator approach of Czerwiński et al. [9].

Of course the idea of converting a parity game into an equivalent safety game is itself much older than QPT algorithms for parity games (see e.g. Bernet, Janin, and Walukiewicz [2]), and was applied not only to finite games, but also to pushdown and collapsible pushdown games [15, 18].

In this paper we deliberate on the Lehtinen’s algorithm [25]. As already said, PG separators corresponding to the other algorithms [6, 22, 14] are deterministic; in such a situation it is straightforward that the product game (obtained from an original parity game and the PG separator) is equivalent to the original game (see, e.g., [9, Proposition 3.2]). The PG separator corresponding to the Lehtinen’s algorithm [25] is nondeterministic, though, and in general while taking a product of a game with a nondeterministic automaton we do not obtain an equivalent game. Actually, a notion of *good-for-games* (GFG) automata was introduced [19]; this is a subclass of nondeterministic automata for which it is guaranteed that the product game remains equivalent. But one can see that the Lehtinen’s separator is not GFG; in consequence, the fact that the Lehtinen’s algorithm actually works is quite intriguing. As a first contribution we explain this phenomenon. Namely, we define a notion of *suitable-for-parity-games* (SFPG) separators, which is more comprehensive than the GFG notion (but, unlike GFG, applies only to parity games, not to arbitrary games), and which covers the Lehtinen’s separator. We then prove that the winner does not change while taking

a product of a parity game with an arbitrary SFPG separator, which means that every SFPG separator can be used to solve parity games. In this way, we establish a framework for solving parity games via nondeterministic PG separators.

As a second contribution, we improve the complexity of the Lehtinen’s algorithm. Let us recall that the algorithm converts the original parity game with n nodes and d priorities into a parity game with $n^{O(\log d)}$ nodes and $O(\log n)$ priorities (which is actually a product of the original game and of an appropriate SFPG separator). Once the new game is created, it has to be solved, say by the small progress measures algorithm [21], which is exponential in the number of priorities: the resulting complexity is $n^{O(\log d \cdot \log n)}$.¹ We observe here that the resulting parity game is of a special form – it is possible to win the game without seeing n opponent’s priorities in a row – and in consequence it can be solved faster: in time $n^{O(\log n)}$. This locates the complexity of the Lehtinen’s algorithm much closer to the complexity of the other QPT algorithms [6, 22, 14, 27], which is $n^{O(\log d)}$ (being the same for d close to n , but better for games with a small number of priorities).

Our paper is structured as follows. In Section 2 we give all necessary definitions. In Section 3 we define SFPG separators, and we prove that they can be used to solve parity games. In Section 4 we recall the Lehtinen’s separator, and we prove that the product game is of a special form. In Section 5 we prove that this product game can be solved quickly.

2 Preliminaries

Parity Games. Parity games are played on *game graphs* of the form $G = (V, V_{\square}, V_{\triangle}, v_I, E)$, where V is a set of nodes, $(V_{\square}, V_{\triangle})$ is a partition of V (which satisfies $V_{\square} \cup V_{\triangle} = V$ and $V_{\square} \cap V_{\triangle} = \emptyset$), $v_I \in V$ is a *starting node*, and $E \subseteq V \times \{1, 2, \dots, d\} \times V$ is a set of directed edges labeled by numbers called *priorities*. Typically, we assume that $V = \{1, 2, \dots, n\}$ for some natural number n . We use d to denote an upper bound for priorities of edges. Without loss of generality, we assume that every node has at least one outgoing edge.

The game is played by two players who are called Even and Odd. A play starts at the starting node v_I and then the players move by following outgoing edges forever, thus forming an infinite path. Every node of the graph is owned by one of the two players: nodes from V_{\square} and V_{\triangle} belong to Even and Odd, respectively. It is always the owner of the node who moves by following an outgoing edge from the current node to a next one.

The outcome of the two players interacting in a parity game by making moves is an infinite path in the game graph. We identify such infinite paths with sequences of edges constituting these paths; thus an infinite path is an infinite word over the alphabet $\Sigma_{n,d} = \{1, 2, \dots, n\} \times \{1, 2, \dots, d\} \times \{1, 2, \dots, n\} \supseteq E$. The set of all infinite words over $\Sigma_{n,d}$ is denoted $\Sigma_{n,d}^{\omega}$.

We write $\text{LimsupEven}_{n,d}$ for the set of infinite words $w \in \Sigma_{n,d}^{\omega}$ in which the largest number that occurs infinitely many times in the priority component of the letters is even, and we write $\text{LimsupOdd}_{n,d}$ for the set of infinite words $w \in \Sigma_{n,d}^{\omega}$ in which that number is odd. Observe that the sets $\text{LimsupEven}_{n,d}$ and $\text{LimsupOdd}_{n,d}$ form a partition of the set $\Sigma_{n,d}^{\omega}$ of all infinite words over the alphabet $\Sigma_{n,d}$. An infinite path in a game graph with n nodes and edge priorities not exceeding d is *won* by Even if and only if the play is in $\text{LimsupEven}_{n,d}$.

A *positional strategy* for Even is a set of edges that go out of nodes she owns – exactly one such edge for each of her nodes. Even uses such a strategy by always – if the current

¹ A better complexity can be obtained by using one of the other QPT algorithms to solve the resulting game.

32:4 Parity Games: Another View on Lehtinen's Algorithm

node is owned by her – following the unique outgoing edge that is in the strategy. Note that when Even uses a positional strategy, her moves depend only on the current node – they are oblivious to what choices were made by the players so far. If Even wins the game by following such a strategy, no matter what edges her opponent Odd follows whenever it is her turn to move, then such a strategy is called *winning*. Analogously we define a positional (winning) strategy for Odd. A basic result for parity games that has notable implications is their *positional determinacy* [11, 26]: exactly one of the players has a positional winning strategy.

The *strategy subgraph* of a game graph G with respect to a positional strategy for Even is the subgraph of G that includes all outgoing edges from nodes owned by Odd and exactly those outgoing edges from nodes owned by Even that are in the positional strategy. Observe that the set of plays that arise from Even playing her positional strategy is exactly the set of all plays in the strategy subgraph.

Let $\text{PosEven}_{n,d}$ and $\text{PosOdd}_{n,d}$ be the sets of all plays that arise from positional winning strategies for Even and Odd, respectively, in some game graph with n nodes and priorities up to d . Clearly $\text{PosEven}_{n,d} \subseteq \text{LimsupEven}_{n,d}$ and $\text{PosOdd}_{n,d} \subseteq \text{LimsupOdd}_{n,d}$. The difference between $\text{PosEven}_{n,d}$ and $\text{LimsupEven}_{n,d}$ is not only in words that are not valid paths (where the target of some edge does not match the source of the next edge); in $\text{LimsupEven}_{n,d} \setminus \text{PosEven}_{n,d}$ we have for example the path $((1, 2, 1)(1, 1, 2)(2, 2, 2)(2, 1, 1))^\omega$ (if this path follows a positional strategy for Even in some game graph, then $((1, 1, 2)(2, 1, 1))^\omega$ follows such a strategy as well, but the latter path is won by Odd).

Parity and Safety Automata. We consider here only automata reading plays of parity games, so we assume that the input alphabet is $\Sigma_{n,d}$ for some n and d . We use d' to denote an upper bound for priorities emitted by parity automata. A non-deterministic *parity automaton* is a tuple $\mathcal{A} = (Q, s_I, \Delta)$, where Q is a finite set of *states*, $s_I \in Q$ is an *initial state*, and $\Delta \subseteq Q \times \Sigma_{n,d} \times \{1, 2, \dots, d'\} \times Q$ is a *transition relation*. Without loss of generality, we assume that the transition relation is *total*, that is, for every state s and letter e , there is some priority p and some state s' , such that the tuple (s, e, p, s') is in the transition relation.

Such a parity automaton can be seen as a directed graph, where $(s, e, p, s') \in \Delta$ is an edge labeled by a letter e and by a priority p . An infinite path in this graph, starting in the initial state, is called a *run* of \mathcal{A} . The word *read by* such a run (being a word over $\Sigma_{n,d}$) is obtained by projecting every edge of the run to its second component. A run is *accepting* if the largest priority that labels infinitely many edges of the run is even. If an accepting run reading a word w exists, we say that w is *accepted*, and we write $\mathcal{L}(\mathcal{A})$ for the set of all words accepted by \mathcal{A} .

A parity automaton is called a *safety automaton* if $d' = 2$, and there is a set of *rejecting states* such that

- if $(s, e, 1, s') \in \Delta$, then s' is rejecting, and
- if s is rejecting then all transitions $(s, e, p, s') \in \Delta$ are such that $p = 1$ and s' is rejecting.

We notice that a run of a safety automaton is accepting if it does not visit rejecting states.

3 Product Games and SFPG Separators

We first recall the notion of product games and separators considered in Czerwiński et al. [9].

► **Definition 3.1.** *Given a game graph $G = (V, V_\square, V_\Delta, v_I, E)$ with at most n nodes and priorities up to d , and a parity automaton $\mathcal{A} = (Q, s_I, \Delta)$ with input alphabet $\Sigma_{n,d}$, we define a game graph $G \times \mathcal{A}$, called a synchronized product of G and \mathcal{A} , in which*

- the set of nodes is $(V \cup E) \times Q$, and the starting node is (v_I, s_I) ;
- ownership of nodes in $V \times Q$ is inherited from the parity game G , and all nodes in $E \times Q$ belong to Even;
- for every edge $e = (u, p, v) \in E$ and every state $s \in Q$, there is an edge $((u, s), 1, (e, s))$;
- for every edge $e = (u, p, v) \in E$ and every transition $(s, e, p', s') \in \Delta$, there is an edge $((e, s), p', (v, s'))$;
- there are no other edges except those specified above.

In other words, the players of $G \times \mathcal{A}$ play in the parity game G , and the automaton \mathcal{A} is fed the edges corresponding to moves made by the players. After every move in G , Even resolves non-deterministic choices in \mathcal{A} . In order to win in $G \times \mathcal{A}$, Even has to ensure that the run of \mathcal{A} reading the play from G is accepting.

It is easy to see that if \mathcal{A} is deterministic, and $\mathcal{L}(\mathcal{A})$ equals $\text{LimsupEven}_{n,d}$ (i.e., the winning condition in G), then the games G and $G \times \mathcal{A}$ have the same winner. The crux of the QPT algorithms is that instead of an automaton recognizing $\text{LimsupEven}_{n,d}$, we can use a *PG separator*.

► **Definition 3.2.** Let \mathcal{A} be a parity automaton with input alphabet $\Sigma_{n,d}$. We say that \mathcal{A} is a parity games separator (PG separator) if it accepts all words from $\text{PosEven}_{n,d}$, and rejects all words from $\text{PosOdd}_{n,d}$. If it additionally rejects all words from $\text{LimsupOdd}_{n,d}$, it is a strong PG separator.

While for solving parity games (i.e., for the equivalence between G and $G \times \mathcal{A}$ described below) it is enough to have a PG separator, the separators corresponding to the QPT algorithms [6, 22, 14, 25] are actually strong PG separators (cf. [9, Section 4]).

If \mathcal{A} is a PG separator, and Odd can win in G , then she can also win in $G \times \mathcal{A}$: she can ensure that the play from G belongs to $\text{PosOdd}_{n,d}$, and such a play is rejected by \mathcal{A} . The same holds for Even, assuming that \mathcal{A} is deterministic. If \mathcal{A} is nondeterministic, however, it is possible that Even wins in G but Odd wins in $G \times \mathcal{A}$. Indeed, if Even wins in G , she can only ensure that the resulting play is accepted by \mathcal{A} . But in $G \times \mathcal{A}$ her task is more difficult: she has to resolve nondeterministic choices of \mathcal{A} as they arise, without knowing the whole play from G . The abilities of Even are described by transition strategies.

► **Definition 3.3.** A transition strategy for an automaton \mathcal{A} is a function $\sigma: \Sigma_{n,d}^* \times Q \times \Sigma_{n,d} \rightarrow \Delta$ such that $\sigma(w, s, e)$ is of the form (s, e, p, s') for all $(w, s, e) \in \Sigma_{n,d}^* \times Q \times \Sigma_{n,d}$. We use such a strategy to resolve non-deterministic choices: if the word read so far is w , the state of \mathcal{A} is s , and the next letter to be read is e , then we proceed using the transition $f(w, s, e)$. We say that a transition strategy σ is winning for a set of words $L \subseteq \mathcal{L}(\mathcal{A})$ if for every word $w \in L$, the run obtained by following σ while reading the word w is accepting.

Henzinger and Piterman [19] proposed a notion of good-for-games automata: an automaton \mathcal{A} is *good for games* (GFG) if in \mathcal{A} there exists a transition strategy that is winning for $\mathcal{L}(\mathcal{A})$. If \mathcal{A} is GFG, then Even can use a winning strategy from G and a transition strategy winning for $\mathcal{L}(\mathcal{A})$ to win in $G \times \mathcal{A}$. We observe, though, that it is not a problem for Even to have a transition strategy that depends on G , and on her winning strategy in G . This way we come to a more comprehensive definition of SFPG separators.

► **Definition 3.4.** A PG separator \mathcal{A} with input alphabet $\Sigma_{n,d}$ is suitable for parity games (SFPG) if for every game graph G with n nodes and priorities up to d , and for every positional winning strategy τ for Even in G , the automaton \mathcal{A} has a transition strategy σ winning for the set of all plays in G that arise from τ .

Notice that every deterministic automaton \mathcal{A} is good for games: the transition strategy that in every situation chooses the only available transition allows to accept all words from $\mathcal{L}(\mathcal{A})$. Moreover, every good-for-games PG separator \mathcal{A} is SFPG: for every G and τ as in Definition 3.4, all plays in G that arise from τ are accepted by \mathcal{A} (because \mathcal{A} is a PG separator), and thus the transition strategy that is winning for the whole $\mathcal{L}(\mathcal{A})$ (existing because \mathcal{A} is GFG) can be used for the set of these plays. In the next section we present the PG separator corresponding to Lehtinen's algorithm; it is neither deterministic nor good for games, but it is SFPG.

We now prove that by producing a parity game with an SFPG separator, we obtain an equivalent game.

► **Theorem 3.5.** *If G is a game graph with n nodes and priorities up to d , and \mathcal{A} is an SFPG separator with input alphabet $\Sigma_{n,d}$, then Even has a winning strategy in G if and only if she has a winning strategy in the synchronized product $G \times \mathcal{A}$.*

Proof. Suppose first that Even has a winning strategy in G . Then, by positional determinacy, she also has a positional winning strategy τ in G . Because the separator \mathcal{A} is SFPG, it has a transition strategy σ that is winning for the set of all plays in G that arise from τ . Using τ and σ we define an Even's strategy in $G \times \mathcal{A}$: she plays according to τ in the G component, and according to σ in the \mathcal{A} component. An infinite play of $G \times \mathcal{A}$ following this strategy is a pair: a play w in G following τ , and a run ρ of \mathcal{A} reading w and following σ . By assumption on σ , because w is a play in G that arises from τ , we obtain that ρ is accepting. This implies that the considered play of $G \times \mathcal{A}$ is won by Even, and thus the considered strategy is winning for Even.

Next, suppose that Even does not have a winning strategy in G . Then, by positional determinacy, Odd has a positional winning strategy τ in G . This strategy can be also used in $G \times \mathcal{A}$, as Odd takes decisions only in the G part of $G \times \mathcal{A}$. Consider a play of $G \times \mathcal{A}$ following this strategy; it consists of a play w in G following τ , and of a run ρ of \mathcal{A} reading w . Because τ is a positional winning strategy for Odd, we have $w \in \text{PosOdd}_{n,d}$, hence, because the PG separator \mathcal{A} rejects all words from $\text{PosOdd}_{n,d}$, the run ρ is rejecting; the play is won by Odd. This implies that Even does not have a winning strategy in $G \times \mathcal{A}$. ◀

Notice that the product game $G \times \mathcal{A}$ is larger, but potentially simpler, than G . For example, if \mathcal{A} is a safety automaton, out of a parity game we obtain a safety game; the latter can be solved in linear time.

We remark that Colcombet and Fijalkow in their recent work [7] define a similar notion of good-for-small-games automata: an automaton \mathcal{A} is good for (n, d) -parity games if it satisfies our Theorem 3.5, that is, if for every game graph G with n nodes and priorities up to d , the games G and $G \times \mathcal{A}$ are equivalent. Such a definition is purely semantical; it does not give any hint which automata are indeed good for (n, d) -parity games. Our definition of SFPG separators is more concrete: it specifies particular conditions on an automaton (what it should accept / reject, and in which way). In Theorem 3.5 we then prove that every SFPG separator can be indeed used to solve parity games (i.e., that it is good for (n, d) -parity games, in the terminology of Colcombet and Fijalkow).

4 Register Automata

In this section we express Lehtinen's construction [25] as an SFPG separator $\mathcal{R}_{n,d}$. Recall that out of a parity game G she constructs an equivalent parity game, which is essentially $G \times \mathcal{R}_{n,d}$.

The idea of the construction is to store some recently visited priorities in some number of registers. Let us denote $rn(n) = 1 + \lfloor \log_2 n \rfloor$; this is the number of registers needed to solve games with n nodes. In Lehtinen's work, $rn(n)$ is called a register index.²

For all positive numbers n and d , such that d is even, we define a *non-deterministic parity automaton* $\mathcal{R}_{n,d}$ in the following way.

- The set of states of $\mathcal{R}_{n,d}$ is the set of non-increasing $rn(n)$ -sequences $\langle r_{rn(n)}, \dots, r_2, r_1 \rangle$ of “registers” that hold numbers in $\{1, 2, \dots, d\}$. The initial state is $\langle 1, 1, \dots, 1 \rangle$.
- For every state $s = \langle r_{rn(n)}, \dots, r_2, r_1 \rangle$ and letter $e = (u, p, v) \in \Sigma_{n,d}$, we define the *update of s by e* to be the state $\langle r_{rn(n)}, \dots, r_{k+1}, p, \dots, p \rangle$, where k is the greatest index such that $r_1, \dots, r_k < p$.
- For every state $s = \langle r_{rn(n)}, \dots, r_2, r_1 \rangle$ and for every k , $1 \leq k \leq rn(n)$, we define the *k -reset of s* to be the state $\langle r_{rn(n)}, \dots, r_{k+1}, r_{k-1}, \dots, r_2, 1 \rangle$. We say that this k -reset is even (odd) if r_k is even (odd, respectively).
- For every state s and letter $e \in \Sigma_{n,d}$, if s' is the update of s by e , then in the transition relation there is a transition $(s, e, 1, s')$, called a *non-reset* transition.
- For every state s , letter $e \in \Sigma_{n,d}$, and for every k , $1 \leq k \leq rn(n)$, if s' is the update of s by e , and s'' is the even k -reset of s' , then in the transition relation there is a transition $(s, e, 2k, s'')$, called an *even reset of register k* .
- For every state s , letter $e \in \Sigma_{n,d}$, and for every k , $1 \leq k \leq rn(n)$, if s' is the update of s by e , and s'' is the odd k -reset of s' , then in the transition relation there is a transition $(s, e, 2k + 1, s'')$, called an *odd reset of register k* .
- There are no other transitions in $\mathcal{R}_{n,d}$ except those specified above.

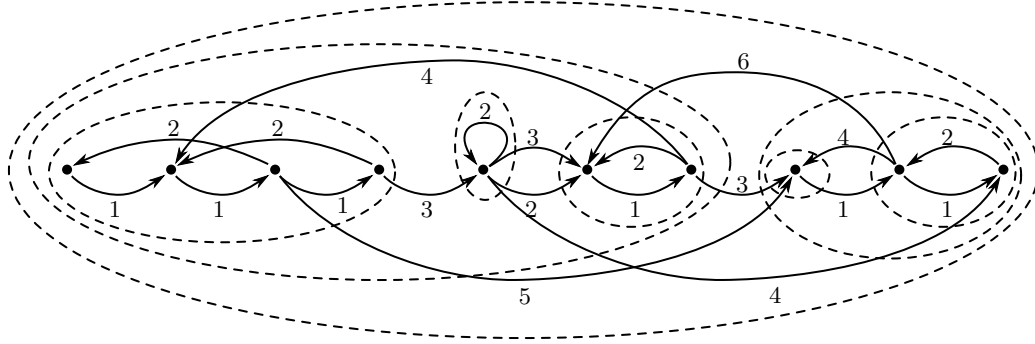
In Theorem 4.2 we prove that $\mathcal{R}_{n,d}$ is indeed an SFPG separator. Moreover, we prove that its runs are of a special form, as specified by Definition 4.1; this is useful in Section 5, where we argue that the product game $G \times \mathcal{R}_{n,d}$ can be solved faster than an arbitrary parity game.

► **Definition 4.1.** *Let ρ be a run of a parity automaton. We define $bad(\rho)$ to be the greatest number m such that in ρ there is an infix containing m transitions emitting some odd priority p and no transitions emitting higher priority.*

► **Theorem 4.2.** *The automaton $\mathcal{R}_{n,d}$ is a strong SFPG separator. Moreover, for every game graph G with n nodes and priorities up to d , for every Even's positional winning strategy τ in G , and for every run ρ of $\mathcal{R}_{n,d}$ that follows the transition strategy existing by Definition 3.4 and that reads a play in G arising from τ , it holds that $bad(\rho) \leq n - 1$.*

We now prove Theorem 4.2. We start with the easier part, saying that \mathcal{A} rejects all words from $\text{LimsupOdd}_{n,d}$. Consider thus a word $w \in \text{LimsupOdd}_{n,d}$, and a run ρ of $\mathcal{R}_{n,d}$ reading this word. If from some moment there are no more resets in this run, then indeed ρ is rejecting. Otherwise, consider the greatest (odd) priority p occurring in w infinitely often, and consider the greatest index k such that there are infinitely many resets of register k in ρ . From some moment on, in ρ no priority higher than p is read, and there is no reset of any register $l > k$. A little bit later, after k resets of register k , the value of register k is at most p for the rest of the run. Then, infinitely many times the priority p is read, it is stored to register k , never overwritten by anything larger, and then reset. This means that there are infinitely many odd resets of register k , emitting priority $2k + 1$, while no higher priorities are emitted (except in the finite prefix that we have skipped). In consequence, ρ is rejecting.

² While there exist games with n nodes that can be solved using less registers (i.e., games with a smaller register index), $1 + \lfloor \log_2 n \rfloor$ is the upper bound.



■ **Figure 1** An example of a strategy subgraph, together with the corresponding game tree. Dashed circles depict nodes of the game tree: the largest circle is the root $(3, S)$, inside it we have two children $(2, S_1), (2, S_2)$, ordered left to right, and so on; additionally, every node x of the graph also constitutes a node $(0, \{x\})$ (i.e., a leaf) of the game tree. Notice that edges with odd priorities can only go right. This is a strategy subgraph, so nodes belonging originally to Even have here only a single successor.

For the remaining part of the proof, fix a game graph G , and an Even’s positional winning strategy τ . Let G_τ be the strategy subgraph of G with respect to τ , and let V_τ be the set of those nodes of G_τ that are reachable from the starting node. For a priority p , and for $S \subseteq V_\tau$, let $G_{S,p}$ be the subgraph of G_τ that contains only nodes that belong to S and only edges of priority not larger than p .

We now define a *game tree* of G_τ in a top-down fashion. The root of this tree is $(\lceil d/2 \rceil, V_\tau)$. Let now (k, S) be an (already defined) node of this tree such that $k \geq 1$, and let S_1, S_2, \dots, S_m be (the sets of nodes of) all the strongly connected components of $G_{S,2k-1}$. We assume that S_1, S_2, \dots, S_m are sorted topologically, that is, that in $G_{S,2k-1}$ there are no edges to S_i from S_j when $i < j$ (if there are multiple such orders of S_1, S_2, \dots, S_m , we fix one of them). In such a case, $(k-1, S_1), (k-1, S_2), \dots, (k-1, S_m)$ are children of (k, S) , in this order. An example of a game tree is presented in Figure 1.

Notice that if S_i is a strongly connected component of $G_{S,2k-1}$, then it does not contain edges of priority $2k-1$. Indeed, if such an edge existed inside a strongly connected component, there would be a cycle in $G_{S,2k-1}$ (i.e., in G_τ) on which the maximal priority would be $2k-1$ (odd); by reaching such a cycle (recall that $S \subseteq V_\tau$ contains only nodes reachable in G_τ from the starting node) and repeating it forever, we would obtain a play won by Odd, while all plays in G_τ are, by assumption, won by Even. It follows that S_i is actually also a strongly connected component of $G_{S,2k-2}$. In other words, $G_{S_i,2k-2} = G_{S_i,2k-1}$.

For a node (k, S) of the game tree, and for $l < k$, let $fst_l(S) = S'$ for (l, S') being the leftmost descendant of (k, S) located on level l .

The following lemma states our thesis in a form suitable for induction. It uses a notion of a *partial run*, which is defined like a run, but it needs not to start in the initial state, and it needs not to be infinite.

► **Lemma 4.3.** *Let (k, S) be a node of the game tree of G_τ , let s be a state of $\mathcal{R}_{n,d}$, and let ξ be a nonempty (finite or infinite) path in $G_{S,2k}$ starting in a node v . Assume that if an odd number $2l+1$ (where $l \geq 1$) is contained in some of the registers $1, 2, \dots, rn(|S|)$ of s , then $l < k$ and $v \notin fst_l(S)$. Under these assumptions, there exists a partial run ρ from s reading ξ , such that*

1. *in ρ there are no resets of registers above $rn(|S|)$,*

2. if the register $rn(|S|)$ in s contains an even number not smaller than $2k$, then in ρ there are no odd resets of the register $rn(|S|)$,
3. $bad(\rho) \leq |S| - 1$, and
4. ρ follows a transition strategy (that may depend on G, τ, k, S, s): in every step, the non-determinism is resolved basing only on the prefix of ξ read so far and on the next edge of ξ that should be read.

In order to finish the proof of Theorem 4.2, we simply use Lemma 4.3 for $(k, S) = (\lceil d/2 \rceil, V_\tau)$, and for $s = \langle 1, 1, \dots, 1 \rangle$ (i.e., for the initial state of $\mathcal{R}_{n,d}$). Indeed, every play w in G that arises from the strategy τ is a path in $G_{V_\tau, 2\lceil d/2 \rceil}$; thus the lemma gives us a run ρ reading w . By Point 3, $bad(\rho)$ is finite, which implies that ρ is accepting. Because this holds for all G and τ , and because \mathcal{A} rejects all words from $\text{LimsupOdd}_{n,d}$ (as shown at the beginning), we already know that \mathcal{A} is a strong PG separator. Point 4 says that ρ is constructed following a transition strategy, so \mathcal{A} is SFPG. The condition $bad(\rho) \leq |V_\tau| - 1$ from Point 3 gives us the second part of the theorem's statement.

Proof of Lemma 4.3. We proceed by induction on k . If $k = 0$, then there is no nonempty path ξ in $G_{S,2k}$ (this graph has no edges), so the lemma trivially holds.

For the rest of the proof, suppose that $k \geq 1$. Let $(k-1, S_1), \dots, (k-1, S_m)$ be the children of (k, S) . By definition, S_1, \dots, S_m form a division of S . Obviously $|S_i| \leq |S|$, so $rn(|S_i|) \leq rn(|S|)$, for all $i \in \{1, \dots, m\}$.

Notice first that no matter how ρ is constructed, none of its last $rn(|S|)$ registers contains an odd number greater than $2k$, in all states of ρ – call this property (\spadesuit). This holds because the condition is satisfied in the first state s of ρ , and then only edges of priority up to $2k$ are read.

We construct a run ρ reading ξ by repeating the following steps:

- in the remaining part of ξ , let ξ' be the maximal prefix that stays in $G_{S_i, 2k-1}$ (i.e., in $G_{S_i, 2k-2}$) for some i (possibly $|\xi'| = 0$, i.e., already the first edge leaves $G_{S_i, 2k-1}$);
- if $i \geq 2$, then
 - let $j_1 < j_2 < \dots < j_r$ be the numbers of registers among $1, \dots, rn(|S_i|)$ which, in the current state, contain an odd priority higher than 1;
 - while reading the first $\min(r, |\xi'|)$ edges of ξ' , we perform resets of the registers $j_1, j_2, \dots, j_{\min(r, |\xi'|)}$, consecutively – call these transitions *preparatory transitions*;
- let ξ'' be the part of ξ' that remains to be read;
- if $|\xi''| > 0$, then we use the induction assumption with $k-1$ as k and with S_i as S to construct a fragment of a run that reads ξ'' (we prove below that the induction assumption can indeed be used) – call the fragment of ρ obtained this way a *block of local transitions*;
- we have now read the whole ξ' ;
- if ξ already ended, we stop the construction;
- otherwise, the next edge of ξ leads outside $G_{S_i, 2k-1}$;
- if this edge has priority $2k$, we reset the register $rn(|S|)$ while reading this edge – call this a *valuable transition*;
- otherwise, we perform a non-reset transition reading this edge – call this a *regressive transition*;
- we repeat the procedure from the beginning.

We have to prove that indeed the induction assumption can be used above. To this end, consider the state s' from which we are about to start a block of local transitions reading a path ξ'' in $G_{S_i, 2k-2}$, and let v' be the first node of this path. Suppose that some of the

32:10 Parity Games: Another View on Lehtinen's Algorithm

registers $1, 2, \dots, rn(|S_i|)$ of s' contains an odd number $2l + 1$, where $l \geq 1$. We have to prove that $l < k - 1$, and that $v' \notin fst_l(S_i)$. By Property (\spadesuit), $l < k$. There are three cases:

- Suppose that $i = 1$ and this is the first time when the loop is used. Then there are no preparatory transitions, so $s' = s$ and $v' = v$. By assumptions of the lemma, $v \notin fst_l(S)$. If $l = k - 1$, we would have $v \notin fst_l(S) = S_1$, while $v \in S_i = S_1$; thus $l < k - 1$. We then have $v' = v \notin fst_l(S) = fst_l(S_1)$.
- Suppose that $i = 1$ and ξ'' is preceded in ξ by some edge. This edge is not an edge of $G_{S_1, 2k-1}$, by maximality of the previous block of local transitions. By the definition of a game tree, there are no edges in $G_{S_1, 2k-1}$ coming to S_1 from $S \setminus S_1$ (S_1, \dots, S_m are topologically sorted strongly connected components of $G_{S, 2k-1}$). Thus, the edge preceding ξ'' has priority $2k$. After reading this edge, all registers contain value $2k$ or higher, or 1 (if there was a reset); they cannot contain $2l + 1$ with $1 \leq l < k$.
- Otherwise, $i \geq 2$. Then, we are just after preparatory transitions. All odd values (greater than 1) present before these transitions were reset to 1. Thus, priority $2l + 1$ appears in a register of s' because it was read during preparatory transitions, and later no edges with priority higher than $2l + 1$ were read. This already implies that $l < k - 1$, because only edges of priority up to $2k - 2$ are read during preparatory transitions. Edges of priority up to $2l + 1$ cannot lead to $fst_l(S_i)$ from $S_i \setminus fst_l(S_i)$: by the definition of the game tree, for every level j with $l \leq j \leq k - 2$, there are no edges of priority up to $2j + 1$ leading to $fst_j(S_i)$ from its (following) siblings. Moreover, there are no edges of priority $2l + 1$ inside $fst_l(S_i)$. Thus, after reading an edge of priority $2l + 1$, and then some edges of priority up to $2l + 1$, we cannot end inside $fst_l(S_i)$.

We now have to check Points 1-4 from the statement of the lemma. Point 1 is immediate: preparatory and valuable transitions reset only registers up to $rn(|S|)$, regressive transitions do not reset anything, and local transitions, by Point 1 of the induction assumption, also reset only registers up to $rn(|S|)$ (recall that $rn(|S_i|) \leq rn(|S|)$).

Point 4 is also immediate: by definition we create ρ in a deterministic way.

While proving Points 2-3 we assume that $|S| \geq 2$; the degenerate case of $|S| = 1$ is handled at the very end.

We now prove Point 2 saying that in ρ there are no odd resets of the register $rn(|S|)$ if this register in the first state of ρ contains an even number not smaller than $2k$. Simultaneously, we prove that odd resets of the register $rn(|S|)$ can appear in ρ only before the first valuable transition – call this property (\clubsuit). Notice first that when we visit some S_i such that $rn(|S_i|) < rn(|S|)$, then neither preparatory transitions, nor local transitions (by Point 1 of the induction assumption) reset the register $rn(|S|)$. On the other hand, $rn(|S_i|) = rn(|S|)$ implies that $|S_i| > |S|/2$, which is possible only for one component S_i ; call it S_{\max} . Regressive transitions do not reset anything.

It remains to handle valuable transitions, and transitions reading edges from $G_{S_{\max}, 2k-2}$ in the case of $rn(|S_{\max}|) = rn(|S|)$; these transitions may reset the register $rn(|S|)$. Recall that, in all states of ρ , none of the last $rn(|S|)$ registers can contain an odd number greater than $2k$ (Property (\spadesuit)). Consider a valuable transition. After the update by priority $2k$, the registers $rn(|S|)$ and $rn(|S|) - 1$ contain even numbers not smaller than $2k$ (we put there $2k$ during the update, unless a larger even priority is already there). Thus, when we reset the register $rn(|S|)$ during a valuable transition, its value is even. Moreover, after this transition, the register $rn(|S|)$ still contains an even number not smaller than $2k$, moved there from the register $rn(|S|) - 1$ (here it is important that $rn(|S|) \geq 2$, so that the register $rn(|S|) - 1$ indeed exists).

By the definition of a game tree, if we leave $G_{S_{\max}, 2k-1}$, then before entering $G_{S_{\max}, 2k-1}$ again there is an edge of priority $2k$, resulting in a valuable transition (edges of priority up to $2k-1$ cannot go to S_i from S_j when $i < j$). Assuming that the register $rn(|S|)$ of s (i.e., of the state from which we start ρ) contains an even number not smaller than $2k$, it follows that whenever we reach S_{\max} , the register $rn(|S|)$ contains an even number not smaller than $2k$ (either existing there from the beginning of ρ , or since the last valuable transition). Thus, the register $rn(|S|)$ is not reset during preparatory transitions, and by Point 2 of the induction assumption, there are no odd resets during the considered block of local transitions for S_{\max} ; we obtain Point 2.

For Property (\clubsuit), we do not have the assumption that at the very beginning the register $rn(|S|)$ contains an even number not smaller than $2k$. In consequence, there may be odd resets of the register $rn(|S|)$ while S_{\max} is visited for the first time, but later, after the first valuable transition, such resets are again impossible.

Next, concentrate on Point 3. We need to prove that:

- for every r , in every infix of ρ without resets of registers above r , there are at most $|S| - 1$ odd resets of the register r , and
- in every infix of ρ without any resets, there are at most $|S| - 1$ (non-reset) transitions.

For $r > rn(|S|)$ there are no r -resets at all (Point 1). Take some $r \leq rn(|S|)$, and consider an infix ρ' of ρ without any resets of registers above r ; let ξ' be the path read by ρ' . We are about to bound the number of odd resets of the register r in ρ' . If $r < rn(|S|)$, the infix ρ' does not contain valuable transitions, as they reset the register $rn(|S|)$, being above the register r . If $r = rn(|S|)$, we can also assume that ρ' does not contain valuable transitions, as anyway, by Property (\clubsuit), after the first valuable transition there are no more odd resets of the register $rn(|S|)$. In consequence, ξ' is a path in $G_{S, 2k-1}$. By the definition of a game tree, such a path can visit components S_1, S_2, \dots, S_m only in an ascending order; every $G_{S_i, 2k-1}$ is visited by ξ' at most once. By Point 3 of the induction assumption, in the block of local transitions in ρ' visiting S_i there are at most $|S_i| - 1$ odd resets of the register r without any resets of registers above r in between. Moreover, in every block of preparatory transitions, we reset the register r at most once, and there are $m - 1$ such blocks: before S_2, S_3, \dots, S_m , but not before S_1 . Together, there are at most $\sum_{i=1}^m (|S_i| - 1) + m - 1 = |S| - 1$ odd resets of the register r in ρ' , as wanted.

The situation is similar when we consider an infix ρ' of ρ without any resets, and we want to bound its length. Again, it visits every $G_{S_i, 2k-1}$ at most once. In every S_i there are at most $|S_i| - 1$ non-reset transitions in a row, by Point 3 of the induction assumption, and we have at most $m - 1$ regressive transitions.

This finishes the proof when $|S| \geq 2$. It remains to prove Points 2-3 in the degenerate case of $|S| = 1$, when $rn(|S|) = 1$. In this case, all edges in $G_{S, 2k}$ are loops around the only node in S . None of them can have an odd priority, because by reaching this node and then repeating this loop we would obtain a play won by Odd, while by assumption all plays in G_τ are won by Even. Moreover, by assumption, if the register 1 of s (i.e., of the state from which we start ρ) contained an odd number $2l + 1$, then $l < k$ and $v \notin fst_l(S)$. But, because $|S| = 1$, we have $fst_l(S) = S$, and $v \in S$. Thus, the register 1 of s contains an even number. In consequence, in all states of ρ the last register (the register number $rn(|S|)$) contains either an even number or 1; we then update it by an even number (so it cannot contain 1 after this update), and then we possibly reset it. This means that we can only have even resets of the register $rn(|S|)$, which gives Point 2. For Point 3, we also need to know that there are no non-reset transitions. But observe that for $|S| = 1$ there are no regressive transitions (all edges of priority up to $2k - 1$ stay inside $G_{S_1, 2k-1}$), and there are no non-reset transitions among local transitions, by Point 3 of the induction assumption. ◀

32:12 Parity Games: Another View on Lehtinen's Algorithm

We remark that the proof presented above is based on Lehtinen's work [25]. We only prove a slightly stronger property, and we expand some details that in Lehtinen's paper are treated in a quite sketchy way.

Because states of $\mathcal{R}_{n,d}$ consist of non-increasing $rn(n)$ -sequences of priorities in $\{1, \dots, d\}$, and because $rn(n) = 1 + \lceil \log_2 n \rceil$, the number of states of $\mathcal{R}_{n,d}$ is

$$\eta_{n,d} = \binom{rn(n) + d - 1}{rn(n)} = d^{O(\log n)} = n^{O(\log d)};$$

from every state the automaton has $rn(n) + 1$ transitions reading every letter $e \in \Sigma_{n,d}$ (a non-reset transition, and a reset transition for every register). In consequence, for a game graph G with n nodes, m edges, and priorities up to d , the product game $G \times \mathcal{R}_{n,d}$ has $(n + m) \cdot \eta_{n,d}$ nodes, $m \cdot \eta_{n,d} \cdot (rn(n) + 2)$ edges, and uses $2 \cdot rn(n) + 1$ priorities. Using a standard (i.e., not quasi-polynomial-time) algorithm to solve such a game, the number of priorities goes to the exponent, thus we obtain complexity $n^{O(\log d \cdot \log n)}$.

5 Safety Register Automata

In the final section we show that the property $bad(\rho) \leq n - 1$ obtained in Theorem 4.2 allows us to solve the product game $G \times \mathcal{R}_{n,d}$ faster: in time $n^{O(\log n)}$ instead of $n^{O(\log d \cdot \log n)}$. We could prove this directly, but instead we modify the parity automaton $\mathcal{R}_{n,d}$ into a *safety* automaton $\mathcal{S}_{n,d}$.

We define the safety automaton $\mathcal{S}_{n,d}$ in the following way:

- The set of states of $\mathcal{S}_{n,d}$ is the set of pairs: the first component is a state of the automaton $\mathcal{R}_{n,d}$ and the other component is an $(rn(n) + 1)$ -sequence $\langle c_{rn(n)}, \dots, c_1, c_0 \rangle$ of *counters* with values in $\{1, \dots, n\}$; additionally, in $\mathcal{S}_{n,d}$ we have a designated rejecting state *rej*.
- Throughout this definition, c always stands for the sequence $\langle c_{rn(n)}, \dots, c_1, c_0 \rangle$.
- The initial state is (s^0, c^0) , where s^0 is the initial state of $\mathcal{R}_{n,d}$ and $c^0 = \langle n, n, \dots, n \rangle$.
- For each transition $(s, e, 2k, s')$ in $\mathcal{R}_{n,d}$ that is an even reset of the register k , we have a transition $((s, c), e, 2, (s', c'))$ in $\mathcal{S}_{n,d}$, where $c' = \langle c_{rn(n)}, \dots, c_{k+1}, c_k, n, \dots, n \rangle$.
- For each transition $(s, e, 2k + 1, s')$ in $\mathcal{R}_{n,d}$ that has an odd priority (i.e., is a non-reset transition, or is an odd reset of the register k), we have a transition $((s, c), e, 2, (s', c'))$ in $\mathcal{S}_{n,d}$, where $c' = \langle c_{rn(n)}, \dots, c_{k+1}, c_k - 1, n, \dots, n \rangle$, if $c_k > 1$.
- For each transition $(s, e, 2k + 1, s')$ in $\mathcal{R}_{n,d}$ that has an odd priority we have a transition $((s, c), e, 1, \text{rej})$ in $\mathcal{S}_{n,d}$, where $c_k = 1$.
- Moreover, for every letter e , we have a transition $(\text{rej}, e, 1, \text{rej})$ in $\mathcal{S}_{n,d}$.
- There are no other transitions in $\mathcal{S}_{n,d}$ except those specified above.

► **Theorem 5.1.** *The automaton $\mathcal{S}_{n,d}$ is a strong SFPG separator.*

Proof. Consider first a word $w \in \text{LimsupOdd}_{n,d}$, and a run ρ_S of $\mathcal{S}_{n,d}$ reading this word; we have to prove that ρ_S is rejecting. While projecting every state of ρ_S to its first component, we obtain a run $\rho_{\mathcal{R}}$ of $\mathcal{R}_{n,d}$ also reading w . By Theorem 4.2, $\mathcal{R}_{n,d}$ is a strong SFPG separator, so it rejects all words from $\text{LimsupOdd}_{n,d}$ (cf. Definition 3.2); $\rho_{\mathcal{R}}$ is rejecting. Let p be the largest priority emitted by $\rho_{\mathcal{R}}$ infinitely often; p is odd. Consider the suffix of $\rho_{\mathcal{R}}$ in which no larger priority is emitted. Concentrate now on ρ_S . Emitting the priority p by $\rho_{\mathcal{R}}$ results in decreasing the counter $(p - 1)/2$ by 1, while emitting priorities lower than p leaves the counter $(p - 1)/2$ unchanged. Thus, after n transitions emitting the priority p the rejecting state is reached; ρ_S is rejecting.

Next, consider a game graph G with n nodes and priorities up to d , and an Even's positional winning strategy τ in G . Let $\sigma_{\mathcal{R}}$ be the transition strategy in $\mathcal{R}_{n,d}$ that is winning for the set of all plays in G that arise from τ , existing because $\mathcal{R}_{n,d}$ is SFPG (cf. Definition 3.4). We extend $\sigma_{\mathcal{R}}$ to a transition strategy $\sigma_{\mathcal{S}}$ for automaton $\mathcal{S}_{n,d}$: in $\sigma_{\mathcal{S}}$ we resolve nondeterministic choices in the same way as in $\sigma_{\mathcal{R}}$; the difference is only that in $\mathcal{S}_{n,d}$ we additionally update the counters (in a deterministic way). We have to prove that $\sigma_{\mathcal{S}}$ is winning for the set of all plays in G that arise from τ . To this end, consider such a play; let $\rho_{\mathcal{S}}$ be the run of $\mathcal{S}_{n,d}$ that follows $\sigma_{\mathcal{S}}$ and reads this play. Let $\rho_{\mathcal{R}}$ be the corresponding run of $\mathcal{R}_{n,d}$, obtained by projecting every state of $\rho_{\mathcal{S}}$ to its first component. By Theorem 4.2, $\text{bad}(\rho_{\mathcal{R}}) \leq n - 1$. Notice that when $\rho_{\mathcal{R}}$ emits an odd priority $2k + 1$, we decrease the counter k by 1, and when it emits any higher priority, we reset the counter k to n . Because priority $2k + 1$ is emitted at most $n - 1$ times without emitting any higher priority in between (by the condition $\text{bad}(\rho_{\mathcal{R}}) \leq n - 1$), we obtain that the rejecting state is not reached. In consequence $\rho_{\mathcal{S}}$ is accepting, which finishes the proof. \blacktriangleleft

We see that $\mathcal{S}_{n,d}$ has $\xi_{n,d} = \eta_{n,d} \cdot n^{rn(n)+1} + 1$ states, and that from every state it has at most $rn(n) + 1$ transitions reading every letter. Thus, for a game graph G with n nodes, m edges, and priorities up to d , the product game $G \times \mathcal{S}_{n,d}$ has $(n + m) \cdot \xi_{n,d}$ nodes and no more than $m \cdot \xi_{n,d} \cdot (rn(n) + 2)$ edges. This safety game can be solved in linear time. Without loss of generality we can assume that $d \leq n$, so the running time is of the form $n^{O(\log n)}$.

► **Remark 5.2.** Let us underline two aspects of the definition of SFPG separators that are important for the proofs of Theorems 4.2 and 5.1. First, the transition strategies that we create actually depend on G and τ (unlike in the definition of good-for-games automata). Second, in our transition strategies we choose a next transition basing not only on the priority of an edge to be read, but also basing on its target. For this reason, we use automata that read edges (i.e., triples: source, priority, target), not just priorities, nor pairs: source node of an edge, priority of the edge (as in Bojańczyk and Czerwiński [4, Section 3]).

We believe that it is possible to construct a transition strategy that does not depend on G and τ , and that chooses a next transition basing only on priorities (i.e., without knowing which nodes are visited). The proof of existence of such a transition strategy would be more involved than the proof presented above, however.

Nevertheless, $\mathcal{R}_{n,d}$ and $\mathcal{S}_{n,d}$ are not good for games, due to some words (not being in $\text{PosEven}_{n,d}$) that can be accepted by these automata, but not in a deterministic way. Interestingly, $\mathcal{R}_{n,d}$ accepts exactly $\text{LimsupEven}_{n,d}$, the set of winning plays (while $\mathcal{L}(\mathcal{S}_{n,d})$ is smaller).

References

- 1 Massimo Benerecetti, Daniele Dell'Erba, and Fabio Mogavero. Solving parity games via priority promotion. *Formal Methods in System Design*, 52(2):193–226, 2018. doi:10.1007/s10703-018-0315-1.
- 2 Julien Bernet, David Janin, and Igor Walukiewicz. Permissive strategies: From parity games to safety games. *ITA*, 36(3):261–275, 2002. doi:10.1051/ita:2002013.
- 3 Henrik Björklund and Sergei G. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Mathematics*, 155(2):210–229, 2007. doi:10.1016/j.dam.2006.04.029.
- 4 Mikołaj Bojańczyk and Wojciech Czerwiński. An Automata Toolbox, February 2018. URL: <https://www.mimuw.edu.pl/~bojan/papers/toolbox-reduced-feb6.pdf>.
- 5 Anca Browne, Edmund M. Clarke, Somesh Jha, David E. Long, and Wilfredo R. Marrero. An Improved Algorithm for the Evaluation of Fixpoint Expressions. *Theor. Comput. Sci.*, 178(1-2):237–255, 1997. doi:10.1016/S0304-3975(96)00228-9.

- 6 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.
- 7 Thomas Colcombet and Nathanaël Fijalkow. Universal Graphs and Good for Games Automata: New Tools for Infinite Duration Games. In Mikołaj Bojańczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11425 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2019. doi:10.1007/978-3-030-17127-8_1.
- 8 Anne Condon. The Complexity of Stochastic Games. *Inf. Comput.*, 96(2):203–224, 1992. doi:10.1016/0890-5401(92)90048-K.
- 9 Wojciech Czerwiński, Laure Daviaud, Nathanaël Fijalkow, Marcin Jurdziński, Ranko Lazić, and Paweł Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2333–2349. SIAM, 2019. doi:10.1137/1.9781611975482.142.
- 10 Constantinos Daskalakis and Christos H. Papadimitriou. Continuous Local Search. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 790–804. SIAM, 2011. doi:10.1137/1.9781611973082.62.
- 11 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.
- 12 E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. On model checking for the μ -calculus and its fragments. *Theor. Comput. Sci.*, 258(1-2):491–522, 2001. doi:10.1016/S0304-3975(00)00034-7.
- 13 John Fearnley. Exponential Lower Bounds for Policy Iteration. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 551–562. Springer, 2010. doi:10.1007/978-3-642-14162-1_46.
- 14 John Fearnley, Sanjay Jain, Sven Schewe, Frank Stephan, and Dominik Wojtczak. An ordered approach to solving parity games in quasi polynomial time and quasi linear space. In Hakan Erdogmus and Klaus Havelund, editors, *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA, USA, July 10-14, 2017*, pages 112–121. ACM, 2017. doi:10.1145/3092282.3092286.
- 15 Wladimir Fridman and Martin Zimmermann. Playing Pushdown Parity Games in a Hurry. In Marco Faella and Aniello Murano, editors, *Proceedings Third International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2012, Napoli, Italy, September 6-8, 2012.*, volume 96 of *EPTCS*, pages 183–196, 2012. doi:10.4204/EPTCS.96.14.
- 16 Oliver Friedmann, Thomas Dueholm Hansen, and Uri Zwick. Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 283–292. ACM, 2011. doi:10.1145/1993636.1993675.
- 17 Hugo Gimbert and Rasmus Ibsen-Jensen. A short proof of correctness of the quasi-polynomial time algorithm for parity games. *CoRR*, abs/1702.01953, 2017. arXiv:1702.01953.
- 18 Matthew Hague, Roland Meyer, Sebastian Muskalla, and Martin Zimmermann. Parity to Safety in Polynomial Time for Pushdown and Collapsible Pushdown Systems. In Igor Potapov,

- Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPICs*, pages 57:1–57:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.MFCS.2018.57.
- 19 Thomas A. Henzinger and Nir Piterman. Solving Games Without Determinization. In Zoltán Ésik, editor, *Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, pages 395–410, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. doi:10.1007/11874683_26.
 - 20 Marcin Jurdziński. Deciding the Winner in Parity Games is in $UP \cap co-UP$. *Inf. Process. Lett.*, 68(3):119–124, 1998. doi:10.1016/S0020-0190(98)00150-1.
 - 21 Marcin Jurdziński. Small Progress Measures for Solving Parity Games. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2000. doi:10.1007/3-540-46541-3_24.
 - 22 Marcin Jurdziński and Ranko Lazić. Succinct progress measures for solving parity games. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–9. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005092.
 - 23 Marcin Jurdziński, Mike Paterson, and Uri Zwick. A Deterministic Subexponential Algorithm for Solving Parity Games. *SIAM J. Comput.*, 38(4):1519–1532, 2008. doi:10.1137/070686652.
 - 24 Bakhadyr Khoussainov. A Brief Excursion to Parity Games. In Mizuho Hoshi and Shinnosuke Seki, editors, *Developments in Language Theory - 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings*, volume 11088 of *Lecture Notes in Computer Science*, pages 24–35. Springer, 2018. doi:10.1007/978-3-319-98654-8_3.
 - 25 Karoliina Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 639–648. ACM, 2018. doi:10.1145/3209108.3209115.
 - 26 Andrzej W. Mostowski. Games with forbidden positions. Technical Report 78, Uniwersytet Gdański, 1991.
 - 27 Paweł Parys. Parity Games: Zielonka’s Algorithm in Quasi-Polynomial Time. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science, MFCS 2019, August 26-30, 2019, Aachen, Germany*, volume 138 of *LIPICs*, pages 10:1–10:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.MFCS.2019.10.
 - 28 Michael Oser Rabin. *Automata on Infinite Objects and Church’s Problem*. American Mathematical Society, Boston, MA, USA, 1972.
 - 29 Sven Schewe. Solving parity games in big steps. *J. Comput. Syst. Sci.*, 84:243–262, 2017. doi:10.1016/j.jcss.2016.10.002.
 - 30 Helmut Seidl. Fast and Simple Nested Fixpoints. *Inf. Process. Lett.*, 59(6):303–308, 1996. doi:10.1016/0020-0190(96)00130-5.
 - 31 Jens Vöge and Marcin Jurdziński. A Discrete Strategy Improvement Algorithm for Solving Parity Games. In E. Allen Emerson and A. Prasad Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 202–215. Springer, 2000. doi:10.1007/10722167_18.
 - 32 Wiesław Zielonka. Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.
 - 33 Uri Zwick and Mike Paterson. The Complexity of Mean Payoff Games on Graphs. *Theor. Comput. Sci.*, 158(1&2):343–359, 1996. doi:10.1016/0304-3975(95)00188-3.

De Jongh’s Theorem for Intuitionistic Zermelo-Fraenkel Set Theory

Robert Passmann 

Institute for Logic, Language and Computation, University of Amsterdam, The Netherlands
<http://robertpassmann.github.io/>
robertpassmann@posteo.de

Abstract

We prove that the propositional logic of intuitionistic set theory IZF is intuitionistic propositional logic IPC. More generally, we show that IZF has the de Jongh property with respect to every intermediate logic that is complete with respect to a class of finite trees. The same results follow for constructive set theory CZF.

2012 ACM Subject Classification Theory of computation → Constructive mathematics; Theory of computation → Proof theory

Keywords and phrases Intuitionistic Logic, Intuitionistic Set Theory, Constructive Set Theory

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.33

Related Version The paper is also available at <https://arxiv.org/abs/1905.04972>.

Funding This research was partially supported by the *Studienstiftung des deutschen Volkes* (German Academic Scholarship Foundation).

Acknowledgements I would like to thank Lorenzo Galeotti, Joel Hamkins, Rosalie Iemhoff, Dick de Jongh, Yurii Khomskii, and Benedikt Löwe for helpful and inspiring discussions.

1 Introduction

Originally motivated by philosophical concerns about the meaning of logical symbols such as \forall and \exists , *intuitionistic logic* has been increasingly influential in computer science due to its constructive nature: in contexts of implementation, the abstract existence of a solution (roughly corresponding to the classical interpretation of \exists) is often less useful than the ability to construct such a solution (roughly corresponding to the intuitionistic interpretation of \exists).

As a consequence, we can see that computational systems used as automated theorem provers or proof assistants use constructive logic as their underlying logic (see, e.g., Wiedijk’s discussion in [28, 127–129]).

However, you cannot just add mathematical axioms to a constructive logic and expect that the resulting system remains constructive: the most famous example of this is the fact that even in the context of intuitionistic logic, the Axiom of Choice proves the law of excluded middle, thus giving full classical logic [6, 159–160].

It is therefore important to determine which axiomatic frameworks for mathematics preserve which constructive logical systems. More formally, if T is any mathematical theory, we let $\mathbf{L}(T)$ be the propositional logic consisting of all propositional formulas φ such that all substitution instances of φ with sentences of the appropriate language are theorems of T (i.e., $T \vdash \varphi^\sigma$ for all substitutions σ of propositional letters for T -sentences). We are interested in determining for well-known constructive foundational systems T whether $\mathbf{L}(T)$ is intuitionistic propositional logic **IPC** or not. This property is known as *de Jongh’s Theorem* for T .

The main result of this paper is that both intuitionistic Zermelo-Fraenkel set theory IZF and constructive Zermelo-Fraenkel set theory CZF satisfy de Jongh’s Theorem.



© Robert Passmann;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 33; pp. 33:1–33:16

Leibniz International Proceedings in Informatics



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

Background: de Jongh's theorem and the de Jongh property. The questions and basic concepts of this work originated in arithmetic; we give a brief historical overview. Heyting arithmetic HA , the intuitionistic counterpart to Peano arithmetic, is constructed on the basis of intuitionistic logic by adding certain arithmetical axioms and axiom schemes. Having defined the theory in this way, it follows immediately that the propositional validities of HA contain all of intuitionistic propositional logic IPC , i.e., $\text{IPC} \subseteq \mathbf{L}(\text{HA})$. De Jongh [7] proved that $\mathbf{L}(\text{HA}) = \text{IPC}$, i.e., that adding the axioms of HA does not entail any logical principle that goes beyond IPC .

That results like this are not obvious can be illustrated with an example of an arithmetical theory that does not satisfy de Jongh's theorem: Consider the theory $\text{HA} + \text{MP} + \text{ECT}_0$, i.e., Heyting arithmetic extended with *Markov's Principle* (MP) and *Extended Church's Thesis* (ECT_0). Even though these principles are generally considered constructive, one can show that the propositional logic of this theory contains principles that are not provable in IPC but it can also not prove all of classical logic CPC , i.e., $\text{IPC} \subsetneq \mathbf{L}(\text{HA} + \text{MP} + \text{ECT}_0) \subsetneq \text{CPC}$ (this follows from results of Rose [22] and McCarty [17]; for details see the discussion at the end of [9, Section 2]). The essence of this example is that – even though we construct the theory $\text{HA} + \text{MP} + \text{ECT}_0$ on the basis of intuitionistic logic – its propositional logic contains principles that are not intuitionistically valid. Hence, the theory $\text{HA} + \text{MP} + \text{ECT}_0$ does not satisfy *de Jongh's theorem*.

These arithmetical examples illustrate that de Jongh-style theorems are important as they guarantee that the logics of constructive systems are not strengthened by the mathematical axioms of the system. An in-depth history of *de Jongh's theorem* can be found in the paper [9] of de Jongh, Verbrugge and Visser.

Related Work. Starting with de Jongh's classical result [7] that the propositional logic of Heyting Arithmetic HA is intuitionistic logic IPC , there has been an intensive examination of this phenomenon in arithmetic. Many authors (see, e.g., [5, 8, 23, 26, 27]) have refined and generalised de Jongh's original work for more logics or stronger arithmetical theories.

The de Jongh property was introduced and analysed for Heyting arithmetic by de Jongh, Verbrugge and Visser [9]: It is an interesting generalisation of de Jongh's theorem. Given an intuitionistic theory T and a propositional logic J , we can obtain a strengthened system $T(J)$ by adding all substitution instances of the rules in J to T . We then say that T has the *de Jongh property* with respect to the intermediate logic J whenever $\mathbf{L}(T(J)) = J$. We can think of the theory $T(J)$ as being constructed on the basis of intuitionistic logic enriched with the propositional principles from J .

In this article, we shall investigate the propositional logics of constructive set theory CZF and intuitionistic set theory IZF . In particular Aczel's constructive set theory CZF [1, 2, 3] has the status of a standard theory for constructive mathematics, also due to its type-theoretic interpretation (see [4]). The metamathematical properties of CZF have also been investigated: Rathjen [21] proved that CZF possesses the disjunction property, the numerical existence property and other common metamathematical properties, however Swan [24] showed that CZF does not have the existence property (the definitions can be found in the respective papers).

The *blended Kripke models* that we construct for the purpose of this article are inspired by the constructions of Iemhoff [10] and Lubarsky [13, 14, 15, 16], and combine Kripke semantics with classical models of set theory.

Bounded constructive set theory BCZF is obtained from CZF by restricting the collection schemes to bounded formulas. The present author used Iemhoff's construction to prove that BCZF has the de Jongh property with respect to every Kripke-complete logic (see [20]).

With the techniques used there, it was (provably) not possible to extend the result to CZF. However, in the present work we will be able to derive a result for CZF as a corollary of the result for IZF.

Moreover, the author [19, Chapter 4] proved that the propositional logic of those of Lubarsky’s models that are based on a Kripke frame with leaves contains the intermediate logic **KC** (axiomatised by $\neg\varphi \vee \neg\neg\varphi$). Consequently, the Lubarsky models based on such a frame cannot be used to prove de Jongh properties with respect to logics weaker than **KC**, such as **IPC**.

The blended models have more flexibility than Lubarsky’s models and model a stronger set theory than Iemhoff’s models, and can therefore be used to prove de Jongh’s theorem for IZF and CZF. We will discuss the relation of Lubarsky’s models and the blended models at the end of Section 3.1.

Organisation of the Article. The main result of this article are de Jongh theorems for the set theories CZF and IZF. That is, $\mathbf{L}(\text{IZF}) = \mathbf{IPC}$ and $\mathbf{L}(\text{CZF}) = \mathbf{IPC}$. To prove these results, we introduce a new semantics for IZF, the so-called *blended Kripke models*, or *blended models* for short, that allow for controlling the logic of the set-theoretic Kripke model in a very precise way. To prove our results, it will be enough to refute one substitution instance of every propositional formula that is not intuitionistically valid. We will do so by imitating valuations on Kripke frames for propositional logic through set-theoretic sentences in a corresponding blended model.

Using our blended Kripke models, we show that intuitionistic set theory IZF has the de Jongh property with respect to every intermediate logic J that is characterised by a class of finite trees (see Definition 10 and Theorem 31). Examples of such logics are intuitionistic propositional logic **IPC**, Dummett’s logic **LC**, the Gabbay-de Jongh logics **T_n** and the logics **BD_n** of bounded depth n (see Example 7 for the definitions of these logics). As constructive set theory CZF is a subtheory of IZF, all of these results also apply for CZF (see Corollary 35).

Section 2 discusses the preliminaries for our work. We introduce blended Kripke models in Section 3 and prove that they satisfy intuitionistic set theory IZF. In Section 4, we consider the propositional logic of blended Kripke models and prove de Jongh’s theorem for IZF. We draw some conclusions and state a few questions for further research in Section 5.

2 Preliminaries

In this section, we will discuss the preliminaries for the later sections. After briefly discussing notation and intermediate logics in Section 2.1 and Section 2.2, respectively, we will introduce Kripke semantics for intuitionistic propositional logic in Section 2.3. We will then discuss the de Jongh property in Section 2.4.

2.1 Notation and Meta-Theory

We adopt the following notational policy: The symbol \Vdash will be used for the forcing relation of Kripke models. As usual, we will use \models for the classical modelling relation, and \vdash for the provability relation.

The meta-theory of this article is **ZFC** + “there is a countable transitive model of **ZFC**”, a theory that is strictly in strength between **ZFC** + **Cons**(**ZFC**) and **ZFC** + “there is an inaccessible cardinal”.

Note that a *countable transitive model of ZFC* is a countable set $M \models \mathbf{ZFC}$ (where \in is interpreted as usual set-membership) such that whenever $y \in x \in M$, then $y \in M$. The class of ordinals Ord^M of such a countable transitive model M of **ZFC** is a countable ordinal in the meta-universe, i.e., $\text{Ord}^M \in \text{Ord}$. We also refer to Ord^M as the *ordinal height* of M .

2.2 Intuitionistic and Intermediate Logics

We fix a countable set Prop of propositional variables for the scope of this article, and identify propositional logics J with the set of formulas they prove (i.e., $J \vdash \varphi$ if and only if $\varphi \in J$). As usual, we denote intuitionistic propositional logic by **IPC**, and classical propositional logic by **CPC**. We say that a logic J is an intermediate logic if $\text{IPC} \subseteq J \subseteq \text{CPC}$ (in particular, **IPC** and **CPC** are considered intermediate logics here). Intuitionistic predicate logic is called **IQC**.

2.3 Kripke Frames and Kripke Models

We will now introduce Kripke frames for intuitionistic logic. In particular, we will focus on Kripke frames that are trees.

► **Definition 1.** A Kripke frame (K, \leq) is a partial order. We call a Kripke frame (K, \leq) a tree if for every $v \in K$, the set $K^{\leq v} = \{w \in K \mid w \leq v\}$ is well-ordered by \leq , and moreover, if there is a node $r \in K$ such that $r \leq v$ for all $v \in K$ (i.e., K is rooted, and r is its root). A Kripke frame is called finite whenever K is finite.

All finite trees can be constructed recursively according to the following rules: First, every reflexive partial order with only one point is a finite tree. Second, given finitely many finite trees T_i with roots r_i , the partial order T obtained as the disjoint union of the T_i with an additional element r such that $r \leq x$ for all $x \in T$, is a tree. This recursive definition allows us to prove facts about trees by induction on construction complexity.

► **Definition 2.** Given a Kripke frame (K, \leq) , we say that a node e is a leaf if e is maximal with respect to \leq . We denote the set of leaves of (K, \leq) by E_K . A Kripke frame (K, \leq) with leaves is a Kripke frame such that for every $v \in K$ there is some $e \in E_K$ with $v \leq e$. Given a node $v \in K$, let E_v denote the set of all leaves $e \in K$ such that $v \leq e$.

The following combinatorial proposition will be useful later when we will determine the propositional logic of certain Kripke models. An up-set X in a Kripke frame (K, \leq) is a set $X \subseteq K$ such that $v \in X$ and $v \leq w$ implies $w \in X$. Given a finite tree (K, \leq) and a node $v \in K$, let U_v be the number of up-sets $X \subseteq K^{\geq v}$, where $K^{\geq v} = \{w \in K \mid w \geq v\}$.

► **Proposition 3.** In a finite tree (K, \leq) , every node v is uniquely determined by U_v and E_v .

Proof. This is an easy induction on the construction complexity of finite trees. ◀

A valuation on a Kripke frame (K, \leq) is a function $V : \text{Prop} \rightarrow \mathcal{P}(K)$ that is persistent, i.e., if $w \in V(p)$ and $w \leq v$, then $v \in V(p)$. A Kripke model for **IPC** is a triple (K, \leq, V) such that (K, \leq) is a Kripke frame. We can now define, by induction on propositional formulas, the forcing relation \Vdash for propositional logic at a node $v \in K$ in the following way:

1. $K, V, v \Vdash p$ if and only if $v \in V(p)$,
2. $K, V, v \Vdash \varphi \wedge \psi$ if and only if $K, V, v \Vdash \varphi$ and $K, V, v \Vdash \psi$,
3. $K, V, v \Vdash \varphi \vee \psi$ if and only if $K, V, v \Vdash \varphi$ or $K, V, v \Vdash \psi$,
4. $K, V, v \Vdash \varphi \rightarrow \psi$ if and only if for all $w \geq v$, $K, V, w \Vdash \varphi$ implies $K, V, w \Vdash \psi$,
5. $K, V, v \Vdash \perp$ never holds.

Sometimes we will write $v \Vdash \varphi$ instead of $K, V, v \Vdash \varphi$, and $K, V \Vdash \varphi$ if $K, V, v \Vdash \varphi$ holds for all $v \in K$. A formula φ is valid in K if $K, V, v \Vdash \varphi$ holds for all valuations V on K and $v \in K$, and φ is valid if it is valid in every Kripke frame K .

► **Proposition 4** (Persistence). *Let (K, \leq, V) be a Kripke model for **IPC**, $v \in K$, and φ be a propositional formula such that $K, v \Vdash \varphi$ holds. Then $K, w \Vdash \varphi$ holds for all $w \geq v$.*

Proof. By induction on formulas. The base case follows from the definition of a valuation, and the other cases follow easily. ◀

We can now define the logic of a Kripke frame and of a class of Kripke frames.

► **Definition 5.** *If (K, \leq) is a Kripke frame for **IPC**, we define the propositional logic $\mathbf{L}(K, \leq)$ to be the set of all propositional formulas that are valid in K . For a class \mathcal{K} of Kripke frames, we define the propositional logic $\mathbf{L}(\mathcal{K})$ to be the set of all propositional formulas that are valid in all Kripke frames (K, \leq) in \mathcal{K} . Given an intermediate logic J , we say that \mathcal{K} characterises J if $\mathbf{L}(\mathcal{K}) = J$.*

If \leq is clear from the context, we shall write $\mathbf{L}(K)$ for $\mathbf{L}(K, \leq)$. Let us conclude this section with a few examples of intermediate logics and some classes of Kripke frames that characterise them. For proofs we refer to the literature. The following important proposition is well-known.

► **Proposition 6** (e.g., [25, Theorem 6.12]). *Intuitionistic propositional logic **IPC** is characterised by the class of all finite trees.*

► **Example 7.** We present some examples of logics from the paper of de Jongh, Verbrugge and Visser [9] that are characterised by classes of finite trees.

Dummett's logic The logic **LC** is obtained by extending **IPC** with the axiom

$$(p \rightarrow q) \vee (q \rightarrow p).$$

The logic **LC** is characterised by the class of finite linear orders.

Gabbay-de Jongh Logics The logics \mathbf{T}_n , for $n \in \mathbb{N}$, are characterised by the class of finite trees which have splittings of exactly n , i.e., every node is either a leaf or has exactly n successors. \mathbf{T}_1 coincides with **LC**, and the logics \mathbf{T}_n are axiomatised by the following formulas:

$$\bigwedge_{k \leq n+1} \left(\left(\varphi_k \rightarrow \bigwedge_{j \neq k} \varphi_j \right) \rightarrow \bigwedge_{j \neq k} \varphi_j \right) \rightarrow \bigwedge_{k \leq n+1} \varphi_k.$$

Logics of Bounded Depth n The logics \mathbf{BD}_n , for $n \in \mathbb{N}$, are characterised by the finite trees of depth n . The logic of depth 1, \mathbf{BD}_1 is classical logic **CPC** axiomatised by Peirce's law,

$$\beta_1 = ((\varphi_1 \rightarrow \psi) \rightarrow \varphi_1) \rightarrow \varphi_1.$$

For every $n \in \mathbb{N}$, the logic \mathbf{BD}_n is axiomatised by β_n as obtained recursively via:

$$\beta_{n+1} = ((\varphi_{n+1} \rightarrow \beta_n) \rightarrow \varphi_{n+1}) \rightarrow \varphi_{n+1}.$$

2.4 The de Jongh Property

Let φ be a propositional formula and let $\sigma : \text{Prop} \rightarrow \mathcal{L}^{\text{sent}}$ an assignment of propositional variables to sentences in a language \mathcal{L} . By φ^σ we denote the \mathcal{L} -sentence obtained from φ by replacing each propositional variable p with the sentence $\sigma(p)$.

► **Theorem 8** (de Jongh, [7]). *Let φ be a formula of propositional logic. Then $\text{HA} \vdash \varphi^\sigma$ for all $\sigma : \text{Prop} \rightarrow \mathcal{L}_{\text{HA}}^{\text{sent}}$ if and only if $\text{IPC} \vdash \varphi$.*

Given a theory based on intuitionistic logic, we may consider its propositional logic, i.e., the set of propositional formulas that are derivable after substituting the propositional letters by arbitrary sentences in the language of the theory.

► **Definition 9.** *Let T be a theory in intuitionistic predicate logic, formulated in a language \mathcal{L} . A propositional formula φ will be called T -valid if and only if $T \vdash \varphi^\sigma$ for all $\sigma : \text{Prop} \rightarrow \mathcal{L}^{\text{sent}}$. The propositional logic $\mathbf{L}(T)$ is the set of all T -valid formulas.*

Given a theory T and an intermediate logic J , we denote by $T(J)$ the theory obtained by closing T under J .

► **Definition 10.** *We say that a theory T has the de Jongh property if $\mathbf{L}(T) = \text{IPC}$. The theory T has the de Jongh property with respect to an intermediate logic J if $\mathbf{L}(T(J)) = J$.*

De Jongh's theorem is equivalent to the assertion that Heyting arithmetic has the de Jongh property. As explained in the introduction, the theory $\text{HA} + \text{MP} + \text{ECT}_0$ does not have the de Jongh property.

3 Blended Models

This section introduces the new model construction for intuitionistic set theory IZF: the blended models. We will now construct blended Kripke models in Section 3.1, observe some of their basic properties in Section 3.2 and show that they satisfy intuitionistic Zermelo-Fraenkel set theory IZF in Section 3.3. Finally, Section 3.4 contains a simple example of a blended model.

3.1 Constructing Blended Models

For the sake of this construction, we fix a Kripke frame (K, \leq) with leaves. Transitive models of ZFC have an ordinal height Ω ; in our construction all models assigned will have the same ordinal height. To each leaf $e \in K$, we assign a transitive model $M_e \models \text{ZFC}$ of height Ω . Note that Ω denotes the same ordinal in the meta-universe for all $e \in E_K$; we can therefore refer to this ordinal by Ω without specifying a particular $e \in E_K$.

Before giving the technical details of the construction, let us spark the readers intuition. We need to define a collection \mathcal{D}_v of v -sets at every node $v \in K$ of the Kripke model. A v -set x will be a function that assigns to every node $w \geq v$ a collection of previously defined w -sets; $x(w)$ is the extension of x at the node w . Note that these assignments shall not be random but must happen in a coherent way: at every leaf e , the extension $x(e)$ must be a set of the transitive model M_e associated to the leaf e . Moreover, the extensions of x should be monotone along the \leq -relation of the Kripke frame to account for the persistence required in Kripke models for intuitionistic theories – once a member of x , always a member of x . More formally, we shall require for any $y \in x(v)$ that $y \upharpoonright K^{\geq w} \in x(w)$. The truncation of y to $y \upharpoonright K^{\geq w}$ is necessary to obtain the w -set $y \upharpoonright K^{\geq w}$ from the v -set y .

The formal construction of blended models is conducted in three steps. We begin by constructing the collection of domains $\langle \mathcal{D}_v \mid v \in K \rangle$: first the domains for the leaves and, secondly, for all remaining nodes of the Kripke frame. The third step is to define the semantics.

Step 1. Domains for leaves. Let $e \in E_K$ be a leaf, and M_e be the transitive model associated to it. Instead of directly assigning the transitive model M_e as the domain at the node e , we will transform this model into a domain \mathcal{D}_e of functions that is isomorphic to the original model. We define a function $f_e : M_e \rightarrow \text{ran}(f)$ by \in -recursion via $f_e(x) = (e, f_e[x])$.

Then define $\mathcal{D}_e = f_e[M_e]$. Hence, each \mathcal{D}_e is a set of functions $x : K^{\geq e} \rightarrow \text{ran}(x)$ (where $K^{\geq e} = \{e\}$). Moreover, for $\alpha \in \text{Ord}^M$, let $\mathcal{D}_e^\alpha = f_e[(V_\alpha)^{M_e}]$. Then $\mathcal{D}_e^0 = \emptyset$ and it holds that

$$\bigcup_{\alpha \in \text{Ord}^M} \mathcal{D}_e^\alpha = \mathcal{D}_e.$$

In Proposition 14 below, we will see that the domains of the leaves of a blended model are isomorphic to the classical model of set theory associated to the node (with respect to the equality and membership relations).

Step 2. Domains for all nodes. Now we are ready to define the domains at the remaining nodes. We do this simultaneously for all $v \in K \setminus E_K$ by induction on $\alpha \in \Omega$. Let \mathcal{D}_v^α consist of the functions $x : K^{\geq v} \rightarrow \text{ran}(x)$ such that the following properties hold:

- (i) for all leaves $e \geq v$, we have $x \upharpoonright \{e\} \in \mathcal{D}_e^\alpha$,
- (ii) for all non-leaves $w \geq v$, we have $x(w) \subseteq \bigcup_{\beta < \alpha} \mathcal{D}_w^\beta$, and
- (iii) for all nodes $u \geq w \geq v$ we have that $\{y \upharpoonright K^{\geq u} \mid y \in x(w)\} \subseteq x(u)$.

We define the domain \mathcal{D}_v at the node v to be the set

$$\mathcal{D}_v = \bigcup_{\alpha \in \text{Ord}^M} \mathcal{D}_v^\alpha.$$

For completing the definition of the domains of our model, we still require *transition functions* $f_{vw} : \mathcal{D}_v \rightarrow \mathcal{D}_w$ such that $f_{wu} \circ f_{vw} = f_{vu}$. The transition functions explain how the elements at a node v should be interpreted at a later node $w \geq v$ (see also step 3). For this purpose, we use the restriction maps f_{vw} with $x \mapsto x \upharpoonright K^{\geq w}$ as transition functions. Note that these maps are well-defined by the definition of the domains.

Moreover, by the definition of the maps f_{vw} , it is clear that condition (i) is just a special case of condition (iii). We state it separately as it requires special attention when working with blended models.

Step 3. Defining the semantics. We define, by induction on \mathcal{L}_\in -formulas, the forcing relation at every node of the Kripke model in the following way, where φ and ψ are formulas with all free variables shown, and, moreover, $\bar{y} = y_0, \dots, y_{n-1}$ are elements of \mathcal{D}_v for the node v considered on the left side:

1. $(K, \leq, \mathcal{D}), v \Vdash \perp$ never holds,
2. $(K, \leq, \mathcal{D}), v \Vdash \varphi(\bar{y}) \wedge \psi(\bar{y})$ if and only if $(K, \leq, \mathcal{D}), v \Vdash \varphi(\bar{y})$ and $(K, \leq, \mathcal{D}), v \Vdash \psi(\bar{y})$,
3. $(K, \leq, \mathcal{D}), v \Vdash \varphi(\bar{y}) \vee \psi(\bar{y})$ if and only if $(K, \leq, \mathcal{D}), v \Vdash \varphi(\bar{y})$ or $(K, \leq, \mathcal{D}), v \Vdash \psi(\bar{y})$,
4. $(K, \leq, \mathcal{D}), v \Vdash \varphi(\bar{y}) \rightarrow \psi(\bar{y})$ if and only if for all $w \geq v$, $(K, \leq, \mathcal{D}), w \Vdash \varphi(f_{vw}(\bar{y}))$ implies $(K, \leq, \mathcal{D}), w \Vdash \psi(f_{vw}(\bar{y}))$,
5. $(K, \leq, \mathcal{D}), v \Vdash x \in y$ if and only if $x \in y(v)$,
6. $(K, \leq, \mathcal{D}), v \Vdash a = b$ if and only if $a = b$,
7. $(K, \leq, \mathcal{D}), v \Vdash \exists x \varphi(x, \bar{y})$ if and only if there is some $a \in D_v$ with $(K, \leq, \mathcal{D}), v \Vdash \varphi(a, \bar{y})$,
8. $(K, \leq, \mathcal{D}), v \Vdash \forall x \varphi(x, \bar{y})$ if and only if for all $w \geq v$ and $a \in D_w$, $w \Vdash \varphi(a, f_{vw}(\bar{y}))$.

The negation $\neg\varphi$ is an abbreviation for $\varphi \rightarrow \perp$.

► **Definition 11.** We call (K, \leq, \mathcal{D}) the blended Kripke model obtained from $\langle M_e \mid e \in E_K \rangle$.

This finishes the definition of the blended models. If the collection $\langle M_e \mid e \in E_K \rangle$ is either clear from the context, or if it does not matter, we will also say that (K, \leq, \mathcal{D}) is a *blended Kripke model*. We will usually say *blended model* instead of blended Kripke model. Moreover, we might refer to an element $x \in \mathcal{D}_v$ as a *v-set*, and to $x(w)$ as the *extension of x at w*.

An \mathcal{L}_\in -formula φ is *valid in* (K, \leq, \mathcal{D}) if $v \Vdash \varphi$ holds for all $v \in K$, and φ is *valid* if it is valid in every Kripke frame K . We will call (K, \leq) the *underlying Kripke frame* of (K, \leq, \mathcal{D}) , or the frame that (K, \leq, \mathcal{D}) is based on. Moreover, let us call $\llbracket \varphi \rrbracket^{(K, \leq, \mathcal{D})} = \{v \in K \mid v \Vdash \varphi\}$ the *truth set* of a sentence φ in the language of set theory in a blended model (K, \leq, \mathcal{D}) . When the model is clear from the context, we will also write $\llbracket \varphi \rrbracket^K$ or just $\llbracket \varphi \rrbracket$.

Before we continue with some basic properties of the blended models, let us briefly discuss this construction in comparison to Lubarsky's Kripke models [13, 14, 15, 16], which are constructed in a similar way. The crucial difference, however, is that our models are constructed in a top-down manner that allows to choose any (finite) collection of classical models of set theory of the same ordinal height at the leaves, whereas Lubarsky's bottom-up construction requires elementary equivalence of the models involved.

3.2 Basic Properties

We will now observe some basic properties of the blended models.

► **Proposition 12 (Persistence).** Let (K, \leq, \mathcal{D}) be a blended model and φ a formula in the language of set theory. If $v \Vdash \varphi(a_0, \dots, a_{n-1})$ and $w \geq v$, then $w \Vdash \varphi(f_{vw}(a_0), \dots, f_{vw}(a_{n-1}))$.

Proof. This is proved by induction on \mathcal{L}_\in -formulas. ◀

► **Proposition 13.** The blended models are sound with respect to IQC.

Proof. This follows from the more general soundness result for Kripke models for predicate logics with respect to IQC. See, for example, [25, Theorem 6.6]. ◀

We will now make the essential observation that the domains at the leaves are isomorphic to the models they were obtained from.

► **Proposition 14.** Let (K, \leq, \mathcal{D}) be a blended model, and $e \in E_K$ a leaf. Then $(K, \leq, \mathcal{D}), e \Vdash \varphi(f_e(a_0), \dots, f_e(a_{n-1}))$ if and only if $M_e \models \varphi(a_0, \dots, a_{n-1})$ for all elements $a_0, \dots, a_{n-1} \in M_e$.

Proof. Let us first argue that the function $f_e : M_e \rightarrow \mathcal{D}_e$ as introduced in Step 1 is a bijection. Define g by \in -recursion with $(e, x) \mapsto g[x]$. It follows by induction that $g \circ f_e = \text{id}_{M_e}$ and $f_e \circ g = \text{id}_{\mathcal{D}_e}$. Hence, f_e is a bijection.

It suffices to prove the claim for the atomic cases: equality and set-membership. The case for equality follows from the definition of the semantics and the fact that f is bijective. For set-membership observe that if $M_e \models x \in y$, then $f_e(x) \in f_e(y)(e)$ and hence $e \Vdash f_e(x) \in f_e(y)$. Conversely, if $e \Vdash f_e(x) \in f_e(y)$, then $f_e(x) \in f_e(y)(e)$ and hence $x = g(f_e(x)) \in g(f_e(y)) = y$. The other cases follow trivially as the intuitionistic interpretation of the logical symbols in a leaf coincides with the classical interpretation in the model M_e . ◀

Extensionality $\forall a \forall b (\forall x (x \in a \leftrightarrow x \in b) \rightarrow a = b)$

Empty set $\exists a \forall x \in a \perp$

Pairing $\forall a \forall b \exists y \forall x (x \in y \leftrightarrow (x = a \vee x = b))$

Union $\forall a \exists y \forall x (x \in y \leftrightarrow \exists u (u \in a \wedge x \in u))$

Power set $\forall a \exists y \forall x (x \in y \leftrightarrow x \subseteq a)$

Infinity $\exists a (\exists x x \in a \wedge \forall x \in a \exists y \in a x \in y)$

Set Induction $\forall a (\forall x \in a \varphi(x) \rightarrow \varphi(a)) \rightarrow \forall a \varphi(a)$, for all formulas $\varphi(x)$.

Separation $\forall a \exists y \forall x (x \in y \leftrightarrow (x \in a \wedge \varphi(x)))$, for all formulas $\varphi(x)$.

Collection $\forall a (\forall x \in a \exists y \varphi(x, y) \rightarrow \exists b \forall x \in a \exists y \in b \varphi(x, y))$, for all formulas $\varphi(x, y)$, where b is not free in $\varphi(x, y)$.

■ **Figure 1** The axioms of IZF. Note that the formulas $\varphi(x)$ appearing in the axiom schemes are allowed to have parameters.

3.3 Intuitionistic Set Theory Holds in Blended Models

In this section, we will show that the axioms of IZF (see Figure 1) hold in blended models. For the sake of this section, let (K, \leq, \mathcal{D}) be a blended model obtained from $\langle M_e \mid e \in E_K \rangle$.

Intuitionistic set theory IZF is classically equivalent to ZF set theory. With Proposition 14 we note that IZF holds true at every leaf because the models associated with the leaves are models of ZF set theory and classical logic holds in the leaves.

▷ **Claim 15.** The model (K, \leq, \mathcal{D}) satisfies the axiom of extensionality.

Proof. Let $v \in K$ and $a, b \in \mathcal{D}_v$. We have to show that

$$v \Vdash \forall x (x \in a \leftrightarrow x \in b) \rightarrow a = b.$$

So assume that $w \Vdash \forall x (x \in a \leftrightarrow x \in b)$ for all $w \geq v$, i.e., $a(w) = b(w)$ for all $w \geq v$. Hence, a and b are equal as functions with domain $K^{\geq v}$, and so they are equal. ◁

▷ **Claim 16.** The model (K, \leq, \mathcal{D}) satisfies the axiom of pairing.

Proof. Let $v \in K$ and $a, b \in \mathcal{D}_v$. Let c be the function with $c(w) = \{f_{vw}(a), f_{vw}(b)\}$ for all $w \geq v$.

Let us first show that $c \in \mathcal{D}_v$. For condition (i), let $e \geq v$ be a leaf. As $a, b \in \mathcal{D}_v$ it follows from the definition that $f_{ve}(a), f_{ve}(b) \in \mathcal{D}_e$. Hence, by pairing in M_e , we have that $c \upharpoonright \{e\} \in \mathcal{D}_e$, where $c(e) = \{f_{ve}(a), f_{ve}(b)\}$. Conditions (ii) and (iii) of the definition of \mathcal{D}_v follow directly from the definition of c .

Now it is straightforward to check that c constitutes a witness for the axiom of pairing for a and b at the node v . ◁

▷ **Claim 17.** The model (K, \leq, \mathcal{D}) satisfies the axiom of union.

Proof. Let $v \in K$ and $a \in \mathcal{D}_v$. Define a function b with domain $K^{\geq v}$ with $b(w) = \bigcup_{c \in a(w)} c(w)$ for all $w \geq v$.

Again, we need to show that $b \in \mathcal{D}_v$. For condition (i), observe that $f_{ve}(a) \in \mathcal{D}_e$ for every leaf $e \geq v$. As the axiom of union holds in M_e , it follows that there is a witness $b' \in \mathcal{D}_e$. By transitivity of M_e , it must then hold that $b \upharpoonright \{e\} = b' \in \mathcal{D}_e$. As in the previous proposition, conditions (ii) and (iii) follow directly from the definition of b . Then b witnesses the axiom of union for a . ◁

33:10 De Jongh's Theorem for IZF and CZF

▷ **Claim 18.** The model (K, \leq, \mathcal{D}) satisfies the axiom of empty set.

Proof. For every $v \in K$ consider the function 0_v with domain $K^{\geq v}$ such that $0_v(w) = \emptyset$ for all $w \geq v$. This is an element of \mathcal{D}_v and witnesses the axiom of empty set. ◁

▷ **Claim 19.** The model (K, \leq, \mathcal{D}) satisfies the axiom of infinity.

Proof. By recursion on natural numbers, we will define elements $n_v \in \mathcal{D}_v$ simultaneously for every $v \in K$. Let 0_v be the empty set as defined in the proof of Claim 18. Then, if m_v has been defined for all $m < n$, let n_v be the function with $n_v(w) = \{0_w, \dots, (n-1)_w\}$ for all $w \geq v$. This finishes the recursive definition. It follows inductively that every $n_v \in \mathcal{D}_v$, again paying special attention at the leaves: the sets n_e correspond to the finite ordinal $n \in M_e$.

Finally, let $\omega_v(w) = \{n_w \mid n < \omega\}$ for all $w \geq v$. To see that $\omega_v \in \mathcal{D}_v$ note that, for every leaf $e \geq v$, $f_{ve}(\omega_v) = \omega_e \in \mathcal{D}_e$ as M_e satisfies the axiom of infinity.

It follows that ω_v is a witness for the axiom of infinity at the node v . ◁

▷ **Claim 20.** The model (K, \leq, \mathcal{D}) satisfies the axiom scheme of separation.

Proof. Let $\varphi(x, y_0, \dots, y_n)$ be a formula with all free variables shown. Let $v \in K$, $a \in \mathcal{D}_v$ and $b_0, \dots, b_n \in \mathcal{D}_v$. Define c to be the function with domain $K^{\geq v}$ such that

$$c(w) = \{d \in a(w) \mid w \Vdash \varphi(d, b_0, \dots, b_n)\}$$

holds for all $w \geq v$. We have that $c \in \mathcal{D}_v$ by the definition of the domains \mathcal{D}_v . Again, property (i) follows from the fact that separation holds in M_e for every leaf model M_e . Moreover, property (iii) follows by persistence. Finally, c witnesses separation from a by φ with parameters b_i . ◁

▷ **Claim 21.** If K is finite, then the model (K, \leq, \mathcal{D}) satisfies the axiom scheme of collection.

Proof. Let $v \in K$, $\varphi(x, y)$ be a formula (possibly with parameters), and $a \in \mathcal{D}_v$. We need to show that:

$$v \Vdash \forall x \in a \exists y \varphi(x, y) \rightarrow \exists b \forall x \in a \exists y \in b \varphi(x, y).$$

Without loss of generality, assume that $v \Vdash \forall x \in a \exists y \varphi(x, y)$. In particular, by persistence, for every $w \geq v$ and every $x \in a(w)$ there exists some $y \in \mathcal{D}_w$ such that $w \Vdash \varphi(x, y)$. Let α be the minimal ordinal such that for every $w \geq v$ and $x \in a(w)$, there is some $y \in \mathcal{D}_w^\alpha$ with $w \Vdash \varphi(x, y)$. Note that $\alpha < \Omega$ as K is finite. Define b to be the function with domain $K^{\geq v}$ such that $b(w) = \mathcal{D}_w^\alpha$. It follows that $b \in \mathcal{D}_v$, where the case for leaves e follows from the fact that $(V_\alpha)^{M_e}$ is a set in M_e . Hence, b is a witness for the above instance of the collection scheme. ◁

▷ **Claim 22.** The model (K, \leq, \mathcal{D}) satisfies the powerset axiom.

Proof. Let $v \in K$ and $a \in \mathcal{D}_v$. Define a function b with domain $K^{\geq v}$ such that

$$b(w) = \{c \in \mathcal{D}_w \mid w \Vdash c \subseteq f_{vw}(a)\}$$

for all $w \geq v$. We have to show that $b \in \mathcal{D}_v$. Observe that for every leaf $e \geq v$, $f_{ve}(b)$ corresponds to $(\mathcal{P}(a))^{M_e}$, and hence condition (i) is satisfied. Conditions (ii) and (iii) follow easily. ◁

▷ **Claim 23.** The model (K, \leq, \mathcal{D}) satisfies the axiom scheme of set induction.

Proof. We shall show that the set-induction scheme holds for all $v \in K$, i.e., that

$$v \Vdash \forall a (\forall x \in a \varphi(x) \rightarrow \varphi(a)) \rightarrow \forall a \varphi(a),$$

holds for all formulas $\varphi(x)$ and $v \in K$. So assume that $v \Vdash \forall a (\forall x \in a \varphi(x) \rightarrow \varphi(a))$. We have to show that $v \Vdash \forall a \varphi(a)$, i.e., for all $a \in \mathcal{D}_v$ we have that $v \Vdash \varphi(a)$. To do so, we will proceed by a simultaneous induction for all $v \in K$ on the rank of $a \in \mathcal{D}_v$, i.e., the minimal $\alpha < \Omega$ such that $a \in \mathcal{D}_v^{\alpha+1} \setminus \mathcal{D}_v^\alpha$.

The only v -set of rank 0 is the function x that assigns the empty set to every node $w \geq v$, so the assumption of set-induction applies and we have $v \Vdash \varphi(x)$. For the induction step, observe that the members of a v -set x of rank α are w -sets (for some $w \geq v$) of lower rank. Hence, the induction hypothesis applies and it follows by using the assumption of set-induction that $v \Vdash \varphi(x)$. This finishes the induction, and the proof of the claim. \triangleleft

Let us summarise the results of this section in the following theorem.

► **Theorem 24.** *If K is finite, then the model (K, \leq, \mathcal{D}) satisfies IZF. For arbitrary K , the model (K, \leq, \mathcal{D}) satisfies IZF – Collection.*

We do not know whether there is an example of an infinite Kripke frame K and a model (K, \leq, \mathcal{D}) based on K that does not satisfy the collection scheme.

3.4 An Example

To illustrate our construction above, we will construct a Kripke model (K, \leq, \mathcal{D}) such that $(K, \leq, \mathcal{D}) \not\Vdash \text{CH} \vee \neg\text{CH}$, where CH is the continuum hypothesis. Take (K, \leq) to be the three element Kripke frame (K, \leq) with $K = \{v, e_0, e_1\}$ with \leq being the reflexive closure of the relation defined by $v \leq e_0$ and $v \leq e_1$.

Now, let M be any countable transitive model of ZFC + CH, and take G to be generic for Cohen forcing over M . Then we associate the model M with node e_0 , and $M[G]$ with e_1 , i.e., $M_{e_0} = M$ and $M_{e_1} = M[G]$. By our construction above and Proposition 14, we know that $(K, \leq, \mathcal{D}), e_0 \Vdash \text{CH}$ and $(K, \leq, \mathcal{D}), e_1 \Vdash \neg\text{CH}$. Hence, persistence implies that $(K, \leq, \mathcal{D}), v \not\Vdash \text{CH} \vee \neg\text{CH}$.

In particular, observe that $\llbracket \text{CH} \rrbracket = \{e_0\}$, $\llbracket \neg\text{CH} \rrbracket = \{e_1\}$, $\llbracket \text{CH} \vee \neg\text{CH} \rrbracket = \{e_0, e_1\}$ and $\llbracket \top \rrbracket = K$. Hence, every up-set and therefore any valuation on K can be imitated with sentences in the language of set-theory evaluated in the blended model.

Moreover, this example also shows that IZF $\not\Vdash \text{CH} \vee \neg\text{CH}$, i.e., the law of excluded middle does not hold for assertions regarding the continuum. One can easily generalise the above argument to obtain the following proposition.

► **Proposition 25.** *If φ is a sentence in the language of set theory such that there are models M and N of ZFC with the same ordinals such that $M \models \varphi$ and $N \models \neg\varphi$, then IZF $\not\Vdash \varphi \vee \neg\varphi$.*

Of course, this result also follows from the fact that IZF is a subtheory of ZFC having the disjunction property (see [18, Corollary 1]).

4 The Propositional Logic of Blended Models

In this section, we will analyse the propositional logic of blended models and prove the de Jongh property for IZF with respect to intermediate logics that are characterised by a class of finite trees.

4.1 Faithful Blended Models

The aim of this section is to show that we can find a blended model based on every finite tree Kripke frame (K, \leq) that allows us to imitate every valuation on (K, \leq) . Let us begin with a definition and several useful observations.

► **Definition 26.** *A blended model (K, \leq, \mathcal{D}) is called faithful if for every valuation V on the Kripke frame (K, \leq) and every propositional letter $p \in \text{Prop}$, there is an \mathcal{L}_\in -formula φ_p such that $\llbracket \varphi_p \rrbracket^{(K, \leq, \mathcal{D})} = V(p)$.*

This notion was first introduced in [19]. For further discussion and connections to the de Jongh property, see also [20].

Given a natural number n , let Γ_n be the following sentence¹ in the language of set theory:

$$\forall x_0, \dots, x_{n-1} \left(\bigwedge_{i < n} (\forall y \in x_i \forall z \in y \perp) \rightarrow \bigvee_{i < j < n} x_i = x_j \right).$$

Informally, this sentence asserts that given n subsets of $1 = \{\emptyset\}$, at least 2 of them are equal. The power set of 1 is crucial for distinguishing models of non-classical set theories; it is consistent with CZF that the power set of 1 is a proper class (see [13]). Note that Γ_1 is inconsistent and Γ_2 is a theorem of ZF set theory. If Γ_2 is not a theorem, then classical logic does not hold.

Recall that we defined U_v in Section 2.3 to be the number of up-sets $X \subseteq K^{\geq v}$. The following proposition holds for all Kripke frames with leaves and not only for finite trees. We also do not need to assume that U_v is finite.

► **Proposition 27.** *Let (K, \leq) be a Kripke frame with leaves, (K, \leq, \mathcal{D}) be a blended model, and $v \in K$. For every natural number n , we have that $v \Vdash \Gamma_{n+1}$ if and only if $n \geq U_v$.*

Proof. Given any up-set $X \subseteq K^{\geq v}$, we define the element 1_X^v to be the function

$$K^{\geq v} \rightarrow \bigcup_{w \geq v} \mathcal{D}_w, \quad w \mapsto \begin{cases} \{0_w\}, & \text{if } w \in X, \\ \emptyset, & \text{otherwise.} \end{cases}$$

Observe that $1_X^v \in \mathcal{D}_v$ as it is monotone because X is an up-set. Further, we have $1_X^v \neq 1_Y^v$ for up-sets $X \neq Y$ and therefore, $v \not\Vdash 1_X^v = 1_Y^v$. It follows that $v \Vdash \forall y \in 1_X^v \forall z \in y \perp$ for all up-sets X because $1_X^v(w)$ is either empty or contains the empty set for $w \geq v$. We conclude that $v \not\Vdash \Gamma_{n+1}$ for $n < U_v$ taking the 1_X^v as witnesses.

Conversely, assume that $n \geq U_v$. We will first show that whenever $v \Vdash \forall y \in x \forall z \in y \perp$ for some $x \in \mathcal{D}_v$, then x is actually of the form 1_X^v for some up-set $X \subseteq K^{\geq v}$. For contradiction, assume that x was not of the form 1_X^v for some up-set X . Then there is a node $w \geq v$ such that $x(w)$ contains an element y different from 0_w . But then there must be a node $u \geq w$ such that $y(u)$ is non-empty. This is a contradiction to $v \Vdash \forall y \in x \forall z \in y \perp$, and hence, every element $x \in \mathcal{D}_v$ satisfying the above formula must be of the form 1_X^v . As there are only U_v -many elements 1_X^v , we know that the conclusion of Γ_{n+1} must be true at the node v . Hence, $v \Vdash \Gamma_{n+1}$. ◀

¹ The sentences Γ_n were also used in yet unpublished joint work with Lorenzo Galeotti and Benedikt Löwe on the logics of algebra-valued models of set theory; see also the discussion after Theorem 13 of [12]. We adapt them here for the case of Kripke semantics.

The following proposition is a special case of a more general proposition for Kripke models of predicate logic.

► **Proposition 28.** *Let (K, \leq) be a Kripke frame with leaves, (K, \leq, \mathcal{D}) be a blended model and $v \in K$. If $e \not\Vdash \varphi$ for all leaves $e \geq v$, then $v \Vdash \neg\varphi$.*

Proof. By the definition of our semantics, we know that $v \Vdash \neg\varphi$ if and only if $w \not\Vdash \varphi$ for all $w \geq v$. Assume that there was a node $w \geq v$ such that $w \Vdash \varphi$. By persistence we can conclude that $e \Vdash \varphi$ for every leaf $e \geq w$. Hence, $w \not\Vdash \varphi$ for all $w \geq v$, so $v \Vdash \neg\varphi$. ◀

► **Theorem 29.** *Let (K, \leq, \mathcal{D}) be a blended model based on a finite tree (K, \leq) with leaves e_0, \dots, e_{n-1} . If there is a collection of \in -sentences φ_i for $i < n$ such that $e_j \Vdash \varphi_i$ if and only if $i = j$, then (K, \leq, \mathcal{D}) is faithful.*

Proof. Let (K, \leq, \mathcal{D}) be a blended model based on a finite tree (K, \leq) with leaves e_0, \dots, e_{n-1} such that there is a collection of \in -sentences φ_i for $i < n$ such that $e_j \Vdash \varphi_i$ if and only if $i = j$.

As (K, \leq) is a finite tree, we know by Proposition 3 that every node $v \in K$ is uniquely determined by U_v and the set of leaves $e \geq v$.

Let V be a valuation on (K, \leq) . For every $p \in \text{Prop}$, we need to find a sentence ρ_p in the language of set theory such that $\llbracket \rho_p \rrbracket^{(K, \leq, \mathcal{D})} = V(p)$. Due to the finiteness of K , it suffices to consider up-sets of the form $K^{\geq v}$ for some $v \in K$ because general up-sets can be constructed by finitely many disjunctions.

We will now prove for every $v \in K$ that there is a sentence χ_v in the language of set theory such that $(K, \leq, \mathcal{D}), w \Vdash \chi_v$ if and only if $w \geq v$ (i.e., $w \in K^{\geq v}$). Let χ_v be the following sentence, where $n = U_v + 1$:

$$\Gamma_n \wedge \bigwedge_{v \not\leq e_i} \neg\varphi_i$$

By Proposition 27 and Proposition 28 it is clear that $w \Vdash \chi_v$ for all $w \geq v$. For the converse direction, let $w \in K$ such that $w \not\geq v$. There are two cases.

First, if $w < v$, then $U_w > U_v = n$ and hence $w \not\Vdash \Gamma_n$ by Proposition 27. Hence, it follows that $w \not\Vdash \chi_v$.

Second, if $w \not\leq v$, then there must be a leaf $e_i \geq w$ such that $e_i \not\geq v$. By assumption $e_i \Vdash \varphi_i$ and hence, $w \not\Vdash \neg\varphi_i$. But this means that $w \not\Vdash \chi_v$.

This concludes the proof of the theorem. ◀

► **Theorem 30.** *Let (K, \leq) be a finite tree. Then there is a faithful blended model (K, \leq, \mathcal{D}) based on (K, \leq) .*

Proof. Let e_0, \dots, e_{n-1} be the set of leaves of (K, \leq) . Let M be a countable transitive model of ZFC set theory. By set-theoretic forcing, we can obtain generic extensions $M[G_i]$ of M such that $M[G_i] \models 2^{\aleph_0} = \aleph_{i+1}$ for every $i < n$ (see, e.g., [11, Theorem 6.17] for details). Let $M_{e_i} = M[G_i]$, and (K, \leq, \mathcal{D}) be the blended model obtained from $\langle M_i \mid i < n \rangle$. Clearly, $M_{e_i} \models 2^{\aleph_0} = \aleph_{j+1}$ if and only if $i = j$. This implies, by Proposition 14, that $e_i \Vdash 2^{\aleph_0} = \aleph_{j+1}$ if and only if $i = j$. In this situation, we can apply Theorem 29 to conclude that (K, \leq, \mathcal{D}) is faithful. ◀

4.2 The de Jongh Property for IZF and CZF

In this section, we will draw conclusions regarding the de Jongh property for IZF and CZF from the main result of the previous section.

► **Theorem 31.** *Intuitionistic set theory IZF has the de Jongh property with respect to every intermediate logic J that is characterised by a class of finite trees.*

Proof. Let J be an intermediate logic with $\mathbf{L}(\mathcal{K}) = J$, where \mathcal{K} is a class of finite trees. We have to show that $\mathbf{L}(\text{IZF}(J)) = J$, i.e., for every propositional formula, we have that:

$$J \vdash \varphi \text{ if and only if } \text{IZF}(J) \vdash \varphi^\sigma \text{ for all substitutions } \sigma : \text{Prop} \rightarrow \mathcal{L}_\varepsilon^{\text{sent}}.$$

The direction from left to right is immediate from the definition of $\text{IZF}(J)$. We will prove the converse direction by contraposition.

Assume that there is φ such that $J \not\vdash \varphi$. As J is characterised by \mathcal{K} , there is a frame $(K, \leq) \in \mathcal{K}$ and a valuation V such that $(K, \leq), V \not\models \varphi$. By Theorem 30 and the assumption that \mathcal{K} consists of finite trees, we can find a faithful blended model (K, \leq, \mathcal{D}) based on (K, \leq) . For every propositional letter $p \in \text{Prop}$, let ψ_p be a sentence in the language of set theory such that $\llbracket \psi_p \rrbracket^{(K, \leq, \mathcal{D})} = V(p)$. Define an assignment $\sigma : \text{Prop} \rightarrow \mathcal{L}_\varepsilon^{\text{sent}}$ by $\sigma(p) = \psi_p$.

We prove by induction on propositional formulas χ , simultaneously for all $v \in K$ that:

$$(K, \leq), v \Vdash \chi \text{ if and only if } (K, \leq, \mathcal{D}), v \Vdash \chi^\sigma.$$

The base case for propositional letters follows directly from the definition of σ . Furthermore, the induction cases for the connectives \rightarrow , \wedge and \vee follow directly from the fact that their semantics coincide in Kripke models for **IPC** and in blended models. This finishes the induction.

Hence, it follows from the induction that $(K, \leq, \mathcal{D}) \not\models \varphi^\sigma$, and therefore, $\varphi \notin \mathbf{L}(\text{IZF}(J))$. This finishes the proof of the theorem. ◀

► **Corollary 32.** *Intuitionistic set theory IZF has the de Jongh property.*

Proof. By Proposition 6, we know that **IPC** is complete with respect to the class of all finite trees, i.e., this class characterises **IPC**. By the previous Theorem 31, this implies that IZF has the de Jongh property. ◀

More examples of logics that are characterised by classes of finite trees are Gödel-Dummett logic **LC**, the Gabbay-de Jongh logics **T_n**, and the logics of bounded depth **BD_n** (see Example 7).

► **Corollary 33.** *Intuitionistic set theory IZF has the de Jongh property with respect to the logics **LC**, **T_n** and **BD_n**.*

► **Lemma 34.** *If a theory T has the de Jongh property with respect to a logic J , then any theory $S \subseteq T$ has the de Jongh property with respect to J .*

Proof. We have to show that $J \vdash \varphi$ if and only if $S(J) \vdash \varphi^\sigma$ for all $\sigma : \text{Prop} \rightarrow \mathcal{L}_\varepsilon^{\text{sent}}$. The implication from left to right is trivial. We prove the other direction by contraposition. So assume that $J \not\vdash \varphi$. By assumption, T has the de Jongh property with respect to J and hence there is some σ such that $T(J) \not\vdash \varphi^\sigma$. As $S \subseteq T$, it follows that $S(J) \not\vdash \varphi^\sigma$. ◀

► **Corollary 35.** *Constructive set theory CZF has the de Jongh property with respect to every intermediate logic J that is characterised by a class of finite trees. In particular, CZF has the de Jongh property with respect to the logics **IPC**, **LC**, **T_n** and **BD_n**.*

In fact, Lemma 34 implies that Corollary 35 holds for any set theory $T \subseteq \text{IZF}$ based on intuitionistic logic. Indeed, any set theory T that is weaker than IZF has the de Jongh property with respect to every intermediate logic J that is characterised by a class of finite trees.

5 Open Questions

In this paper, we defined a class of Kripke models for intuitionistic set theory IZF, the blended Kripke models. We then used these models to prove a range of de Jongh properties for IZF and CZF. It would certainly be interesting to find a constructive proof of the results presented in this paper.

► **Question 36.** *Does intuitionistic set theory IZF have the de Jongh property with respect to every intermediate logic?*

Lubarsky used his Kripke models for independence results for CZF and IZF. For example, he proved that it is consistent with CZF that the power set of 1 is a proper class (see [13]; for more results in this area see, e.g., [14, 15, 16]). We wonder whether our blended models can be used for similar purposes.

► **Question 37.** *Can we obtain independence results for IZF with blended models?*

► **Question 38.** *Is it possible to vary the construction of blended models to provide proper models of CZF (i.e., models of CZF that are not also models of IZF)?*

References

- 1 Peter Aczel. The type theoretic interpretation of constructive set theory. In Leszek Pacholski Angus Macintyre and Jeff Paris, editors, *Proceedings of the Colloquium held in Wrocław, August 1-12, 1977*), volume 96 of *Studies in Logic and the Foundations of Mathematics*, pages 55–66. North-Holland Publishing Co., Amsterdam-New York, 1978.
- 2 Peter Aczel. The type theoretic interpretation of constructive set theory: choice principles. In A. S. Troelstra and D. van Dalen, editors, *The L. E. J. Brouwer Centenary Symposium. Proceedings of the Symposium held in Noordwijkerhout, June 8-13, 1981*, volume 110 of *Studies in Logic and the Foundations of Mathematics*, pages 1–40. North-Holland Publishing Co., Amsterdam-New York, 1982.
- 3 Peter Aczel. The type theoretic interpretation of constructive set theory: inductive definitions. In Georg J. W. Dorn Ruth Barcan Marcus and Paul Weingartner, editors, *Logic, methodology and philosophy of science. VII. Proceedings of the seventh international congress held at the University of Salzburg, Salzburg, July 11-16, 1983*, volume 114 of *Studies in Logic and the Foundations of Mathematics*, pages 17–49. North-Holland Publishing Co., Amsterdam, 1986.
- 4 Peter Aczel and Michael Rathjen. Notes on Constructive Set Theory. Draft, 2010.
- 5 Mohammad Ardeshtir and S. Mojtaba Mojtahedi. The de Jongh property for Basic Arithmetic. *Archive for Mathematical Logic*, 53(7-8):881–895, 2014.
- 6 John L Bell. *Set theory: Boolean-valued models and independence proofs*, volume 47. Oxford University Press, 2011.
- 7 Dick de Jongh. The maximality of the intuitionistic predicate calculus with respect to Heyting’s arithmetic. *The Journal of Symbolic Logic*, 35(4):606, 1970.
- 8 Dick de Jongh and Craig Smoryński. Kripke models and the intuitionistic theory of species. *Annals of Mathematical Logic*, 9(1-2):157–186, 1976.
- 9 Dick de Jongh, Rineke Verbrugge, and Albert Visser. Intermediate Logics and the de Jongh property. *Archive for Mathematical Logic*, 50(1):197–213, 2011.

- 10 Rosalie Iemhoff. Kripke models for subtheories of CZF. *Archive for Mathematical Logic*, 49(2):147–167, 2010.
- 11 Kenneth Kunen. *Set theory: An introduction to independence proofs*, volume 102 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., Amsterdam-New York, 1980.
- 12 Benedikt Löwe, Robert Passmann, and Sourav Tarafder. Constructing illoyal algebra-valued models of set theory, 2018. ILLC Prepublication Series PP-2018-15.
- 13 Robert S. Lubarsky. Independence results around constructive ZF. *Annals of Pure and Applied Logic*, 132(2-3):209–225, 2005.
- 14 Robert S. Lubarsky. Separating the Fan Theorem and Its Weakenings II. In Sergei N. Artëmov and Anil Nerode, editors, *Logical Foundations of Computer Science—International Symposium, LFCS 2018, Deerfield Beach, FL, USA, January 8-11, 2018, Proceedings*, volume 10703 of *Lecture Notes in Computer Science*, pages 242–255. Springer, 2018.
- 15 Robert S. Lubarsky and Hannes Diener. Separating the Fan Theorem and its weakenings. *The Journal of Symbolic Logic*, 79(3):792–813, 2014.
- 16 Robert S. Lubarsky and Michael Rathjen. On the constructive Dedekind reals. *Logic & Analysis*, 1(2):131–152, 2008.
- 17 David Charles McCarty. Incompleteness in intuitionistic metamathematics. *Notre Dame Journal of Formal Logic*, 32(3):323–358, 1991.
- 18 John Myhill. Some properties of intuitionistic Zermelo-Fraenkel set theory. In A. R. D. Mathias and H. Rogers, editors, *Cambridge Summer School in Mathematical Logic (held in Cambridge, England, August 1-21, 1971)*, volume 337 of *Lecture Notes in Mathematics*, pages 206–231. Springer-Verlag, Berlin-New York, 1973.
- 19 Robert Passmann. Loyalty and Faithfulness of Model Constructions for Constructive Set Theory. Master's thesis, ILLC, University of Amsterdam, 2018. Master of Logic Thesis (MoL) Series MoL-2018-03.
- 20 Robert Passmann. The de Jongh property for bounded constructive Zermelo-Fraenkel set theory. ILLC Prepublication Series PP-2019-05, 2019.
- 21 Michael Rathjen. The disjunction and related properties for constructive Zermelo-Fraenkel set theory. *The Journal of Symbolic Logic*, 70(4):1232–1254, 2005.
- 22 Gene F. Rose. Propositional calculus and realizability. *Transactions of the American Mathematical Society*, 75:1–19, 1953.
- 23 Craig Smoryński. Applications of Kripke models. In A. S. Troelstra, editor, *Metamathematical investigation of intuitionistic arithmetic and analysis*, volume 344 of *Lecture Notes in Mathematics*, pages 324–391. Springer, Berlin, 1973.
- 24 Andrew Swan. CZF does not have the existence property. *Annals of Pure and Applied Logic*, 165(5):1115–1147, 2014.
- 25 Anne Troelstra and Dirk van Dalen. *Constructivism in mathematics. Vol. I*, volume 121 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., Amsterdam, 1988.
- 26 Jaap van Oosten. A semantical proof of de Jongh's theorem. *Archive for Mathematical Logic*, 31(2):105–114, 1991.
- 27 Albert Visser. Rules and arithmetics. *Notre Dame Journal of Formal Logic*, 40(1):116–140, 1999.
- 28 Freek Wiedijk. The QED manifesto revisited. *Studies in Logic, Grammar and Rhetoric*, 10(23):121–133, 2007.

Computing Haar Measures

Arno Pauly 

Swansea University, UK

<http://www.cs.swan.ac.uk/~cspauly/>

arno.m.pauly@gmail.com

Dongseong Seon

KAIST, Daejeon, Republic of Korea

instigation@kaist.ac.kr

Martin Ziegler

KAIST, Daejeon, Republic of Korea

<http://ziegler.theoryofcomputation.asia/>

ziegler@kaist.ac.kr

Abstract

According to Haar's Theorem, every compact topological group G admits a unique (regular, right and) left-invariant Borel probability measure μ_G . Let the *Haar integral* (of G) denote the functional $\int_G : \mathcal{C}(G) \ni f \mapsto \int f d\mu_G$ integrating any continuous function $f : G \rightarrow \mathbb{R}$ with respect to μ_G . This generalizes, and recovers for the additive group $G = [0; 1) \bmod 1$, the usual Riemann integral: computable (cmp. Weihrauch 2000, Theorem 6.4.1), and of computational cost characterizing complexity class $\#P_1$ (cmp. Ko 1991, Theorem 5.32).

We establish that in fact, every computably compact computable metric group renders the Haar measure/integral computable: once using an elegant synthetic argument, exploiting uniqueness in a computably compact space of probability measures; and once presenting and analyzing an explicit, imperative algorithm based on "maximum packings" with rigorous error bounds and guaranteed convergence. Regarding computational complexity, for the groups $\mathcal{SO}(3)$ and $\mathcal{SU}(2)$, we reduce the Haar integral to and from Euclidean/Riemann integration. In particular both also characterize $\#P_1$.


2012 ACM Subject Classification Theory of computation \rightarrow Constructive mathematics; Mathematics of computing \rightarrow Continuous mathematics

Keywords and phrases Computable analysis, topological groups, exact real arithmetic, Haar measure

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.34

Related Version A full version of this work (including additional results and omitted proofs) is available at <http://arxiv.org/abs/1906.12220>.

Supplement Material The source code of the implementation in Section 6.1 is available at <http://github.com/realcomputation/irramplus/tree/master/HAAR>

Funding Supported by the National Research Foundation of Korea (grant NRF-2017R1E1A1A03071032), by the International Research & Development Program of the Korean Ministry of Science and ICT (grant NRF-2016K1A3A7A03950702), and by the  European Union's Horizon 2020 MSCA IRSES project 731143.

1 Motivation and Overview

Complementing empirical approaches, heuristics, and recipes [26, 28], Computable Analysis [35] provides a rigorous algorithmic foundation to Numerics, as well as a way of formally measuring the constructive contents of theorems in classical Calculus. Haar's Theorem is such an example, of particular beauty combining three categories: compact metric spaces, algebraic groups, and measure spaces:



© Arno Pauly, Dongseong Seon, and Martin Ziegler;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 34; pp. 34:1–34:17

Leibniz International Proceedings in Informatics



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

34:2 Computing Haar Measures

► **Fact 1.** Let $(G, e, \circ, \cdot^{-1})$ denote a group and (G, d) a compact metric space such that the group operation \circ and inverse operation \cdot^{-1} are continuous with respect to d (that is, form a topological group). There exists a unique left-invariant Borel probability measure μ_G , called Haar measure, on G . Moreover, μ_G is right-invariant and regular.

We refrain from expanding on generalizations to locally-compact Hausdorff spaces. Recall that a left-invariant measure satisfies $\mu(U) = \mu(g \circ U)$ for every $g \in G$ and every measurable $U \subseteq G$. For the additive group $[0; 1] \pmod 1$, its Haar measure recovers the standard Lebesgue measure λ , corresponding to the angular measure divided by 2π on the complex unit circle group $\mathcal{U}(1) \cong \mathcal{SO}(2)$.

Each of the categories involved in Fact 1 has a standard computable strengthening, cmp. [34, 30, 5]; and our first main result establishes them to combine nicely:

► **Theorem 2.** *Let \mathbf{X} be a computably compact computable metric space with a computable group operation $\circ : \mathbf{X} \times \mathbf{X} \rightarrow \mathbf{X}$. Then the corresponding Haar measure μ is computable.*

That the Haar measure is computable means that we can approximate the measure of any given open subset of \mathbf{X} from below, and implies that we can compute the integral of any given continuous function from \mathbf{X} into \mathbb{R} .

In contrast, recall that other classical results in Calculus, such as Brouwer’s Fixed Point Theorem [19, 2] or Peano’s Theorem [27], do not carry over to computability that nicely. And also common classical “constructive” existence proofs of the Haar measure [11, §58] do employ limits without rate of convergence, well-known since Specker [31, 32] to possibly leave the computable realm:

► **Fact 3.** For non-empty $A, B \subseteq X$ let $[A : B]$ denote the least number of left translates of B that cover A . Then $\mu(A) = \lim_B \frac{[A : B]}{[X : B]}$ holds for every compact $A \subseteq X$, where the limit exists in the sense of a net of open neighborhoods B of e .

In addition to the possibly uncomputable limit, the *least integer* defining $[A : B]$ depends discontinuously and uncomputably on the underlying data A, B .

We establish Theorem 2 with elegant arguments following the “synthetic” (i.e. implicit, functional) approach to Computable Analysis developed in [21]. It follows the following general strategy [7] (also explained in [21, Section 9]) for proving computability of some object Ω living in an admissibly represented space by three steps:

- I) Obtain a definition of Ω as the element of a computably closed set.
- II) Obtain a computably compact set containing Ω .
- III) Find a classical proof that (I) and (II) uniquely determine Ω .

As warm-up let us illustrate this approach to assert computability of the group unit e from the hypothesis of Theorem 2: For any fixed computable element $a \in G$, (I) e belongs to the computably closed set $\{y : a \circ y = a\}$ and (II) to the compact set $\Omega = X$ and (III) is uniquely determined by (I) and (II). Similarly, for every $x \in G$, its inverse x^{-1} (I) belongs to $\{y : x \circ y = e\}$ and (II) to the compact set $\Omega = X$ and (III) is uniquely defined by $x \circ x^{-1} = e$. Note that this proof does not immediately yield an algorithm computing e or $x \mapsto x^{-1}$.

In this spirit, Section 3 establishes Theorem 2. The challenge consists in (I) obtaining a computable definition of the Haar measure μ : The inequality $\tilde{\mu}(U) \neq \tilde{\mu}(xU)$ expressing a candidate measure $\tilde{\mu}$ to violate invariance is not even recognizable, since $\tilde{\mu}(U)$ is in general only a lower real. Subsection 3.1 avoids that by allowing to consider pairs of sets in Lemma 12. Section 4 complements Section 3 by devising and analyzing an explicit, imperative algorithm for computing Haar integrals $\mathcal{C}(f) \mapsto \int_X f d\mu$. It is based on “maximal packings”: finite sets $T_n \subseteq X$ of points with pairwise distance $> 2^{-n}$. Intuitively, the ratio $|T_n \cap A|/|T_n|$ of those

points contained in a given set A should approximate its measure $\mu(A)$; however, rigorously, this is wrong – and counting is uncomputable anyway. Subsections 4.1 and 4.2 describe a combination of mathematical and algorithmic approaches that avoid these obstacles. The superficially different hypotheses to Sections 4 and 3 are compared in Section 5. There we also give some examples showing that these requirements are not dispensable; and analyze which information of a compact metric group determines its Haar measure.

Having thus asserted computability, the natural next question is for efficiency. We consider here the non-uniform computational cost of the *Haar integral* functional

$$\int_G : \mathcal{C}(G) \ni f \mapsto \int f d\mu_G \in \mathbb{R} \quad (1)$$

integrating continuous real functions $f : G \rightarrow \mathbb{R}$. For the arguably most important additive groups $G = [0; 1)^d \bmod 1$ with Lebesgue measure λ^d , this amounts to Euclidean/Riemann integration – whose complexity had been shown to characterize the discrete class $\#P_1$ [14, Theorem 5.32] cmp. [8, 33]: indicating that standard quadrature methods, although taking runtime exponential in n to achieve guaranteed absolute output error 2^{-n} , are likely optimal. And Section 6 extends this numerical characterization of $\#P_1$ to the arguably next-most important compact metric groups:

► **Theorem 4.** *Let G denote any of the following compact groups, considered as subsets of Euclidean space and equipped with the intrinsic/path metric:*

- i) $\mathcal{SO}(3) \subseteq \mathbb{R}^9$ of orthogonal real 3×3 matrices of determinant 1,
- ii) $\mathcal{O}(3) \subseteq \mathbb{R}^9$ of orthogonal real 3×3 matrices,
- iii) $SU(2) \subseteq \mathbb{R}^8$ of unitary complex 2×2 matrices of determinant 1,
- iv) $U(2) \subseteq \mathbb{R}^8$ of unitary complex 2×2 matrices.

- a) For every polynomial-time computable $f \in \mathcal{C}(G)$, $\int_G f \in \mathbb{R}$ is computable in polynomial space (and exponential time).
- b) If $FP_1 = \#P_1$ and $f \in \mathcal{C}(G)$ is polynomial-time computable, then so is $\int_G f \in \mathbb{R}$.
- c) There exists a polynomial-time computable $f \in \mathcal{C}(G)$ such that polynomial-time computability of $\int_G f \in \mathbb{R}$ implies $FP_1 = \#P_1$.

The proof of this result proceeds by mutual polynomial-time continuous (i.e. Weihrauch) reduction from and to Euclidean/Riemann integration. Subsection 6.1 describes our implementation and empirical evaluation of rigorous integration on $SU(2)$ in the `iRRAM C++` library.

2 Background

In the following, we give a brief introduction to the key notions from computable analysis we need. For a formal treatment, we refer to [21]. Further standard references for Computable Analysis are [35, 1].

Computable analysis is concerned with *represented spaces*, which equip a set with a notion of computability by coding its elements as infinite binary sequences. We have various constructions of new represented spaces available, and use in particular the derived spaces $\mathcal{O}(\mathbf{X})$ of open subsets of \mathbf{X} (characterized by making membership recognizable) and $\mathcal{C}(\mathbf{X}, \mathbf{Y})$ of continuous functions from \mathbf{X} to \mathbf{Y} , characterized by making function evaluation computable.

Computable compactness and computable overtiness of a space are characterized by making universal and existential quantification preserve computable open predicates. We also use that admissibility of a space means that from a compact singleton we can extract the

34:4 Computing Haar Measures

point [29]. A space is computably Hausdorff, if inequality is semidecidable. It is computably separable, if it has a computable dense sequence. Being computably separable implies being computably overt.

A particular convenient class of represented space are the computable metric spaces (CMS). We can start with a designated dense sequence on which the metric is computable (given indices), and then represent arbitrary points as limits of fast converging sequences. CMSs are in particular computably Hausdorff and computably separable. The prototypic example of a CMS are the reals \mathbb{R} . We write *CCCMS* for *computably compact computable metric space*.

There is a further relevant represented space with the reals as underlying set, namely the space of lower reals \mathbb{R} . Here a real is represented as the supremum of a sequence of rational numbers (without any limitation on convergence rates). This space is relevant for us as we can introduce the computability structure of the space of (probability) measures on an arbitrary represented space \mathbf{X} by considering them as the subspace of $\mathcal{C}(\mathcal{O}(\mathbf{X}), \mathbb{R}_{<})$ of functions satisfying the properties of a (probability) measure. More precisely, these correspond to continuous valuations. Let $\mathcal{PM}(\mathbf{X})$ denotes the space of probability measures on \mathbf{X} .

A useful theorem is that for a CCCMS \mathbf{X} , also $\mathcal{PM}(\mathbf{X})$ is a CCCMS. Here we can use the *Wasserstein-Kantorovich-Rubinstein* metric

$$W(\mu, \nu) = \sup \left\{ \left| \int f d\mu - \int f d\nu \right| : f : X \rightarrow \mathbb{R}, \forall x, y \in X : |f(x) - f(y)| \leq d(x, y) \right\}$$

If \mathbf{X} a complete metric space, $\mathcal{PM}(\mathbf{X})$ is again a complete metric space; and convergence w.r.t. W is equivalent to weak convergence. For an introduction to computable probability theory, see [4]. Some further results are found in [24].

Regarding computational complexity of real numbers and real functions on compact metric spaces, we refer to [14] and [13].

Recall that $\#P_1$ is the class of all integer functions $\varphi : \{0\}^* \rightarrow \mathbb{N}$ with unary arguments counting the number of witnesses

$$\varphi(0^n) = \text{Card} \{ \vec{w} \in \{0, 1\}^{\text{poly}(n)} : 0^n \mathbf{1} \vec{w} \in P \}$$

to a polynomial-time decidable predicate $P \subseteq \{0, 1\}^*$; a class commonly conjectured to lie strictly between (the integer function versions of) NP_1 and $PSPACE$ [20, §18].

3 The Haar measure is computable

In this section we shall establish Theorem 2 using the approach to computable analysis via synthetic topology [6] outlined in [21]. To this end, we first obtain a more technical result stating that left-invariance of a Radon probability measure for some continuous binary operation constitutes a computably closed predicate:

► **Theorem 5.** *Let \mathbf{X} be a computable metric space. For $\mu \in \mathcal{PM}(\mathbf{X})$ and $g \in \mathcal{C}(\mathbf{X} \times \mathbf{X}, \mathbf{X})$ the following predicate is computably closed:*

$$\forall U \in \mathcal{O}(\mathbf{X}), \forall x \in \mathbf{X} \mu(U) = \mu(\{y \in \mathbf{X} \ g(x, y) \in U\})$$

In view of the general strategy for computability proofs from Section 1, this establishes (I). Regarding (II) recall [9, §2.5] that, if \mathbf{X} is a computably compact computable metric space, then so is $\mathcal{PM}(\mathbf{X})$. Finally, uniqueness in Haar's theorem takes care of Condition (III).

3.1 Disjoint pairs of open sets

Prima facie, the condition in Theorem 5 appears to be complicated. As measures of open sets are only available as lower reals, we cannot even recognize inequality. The workaround consists in considering pairs of disjoint open sets rather than individual open sets. We shall see that quantification over such pairs is unproblematic for the spaces we are interested in here.

Given a represented space \mathbf{X} , we define the space $\text{DPO}(\mathbf{X})$ as the subspace $\{(U, V) \mid U \cap V = \emptyset\} \subseteq \mathcal{O}(\mathbf{X}) \times \mathcal{O}(\mathbf{X})$.

► **Observation 6.** \mathbf{X} is computably overt iff $\text{DPO}(\mathbf{X})$ is a computable element of $\mathcal{A}(\mathcal{O}(\mathbf{X}) \times \mathcal{O}(\mathbf{X}))$.

Proof. If \mathbf{X} is computably overt, then $U \cap V \neq \emptyset$ is a recognizable property given $(U, V) \in \mathcal{O}(\mathbf{X}) \times \mathcal{O}(\mathbf{X})$. Conversely, we find that $(U, X) \notin \text{DPO}(\mathbf{X})$ iff $U \neq \emptyset$. ◀

► **Corollary 7.** If \mathbf{X} is computably overt, then $\text{DPO}(\mathbf{X})$ is computably compact.

Proof. The space $\mathcal{O}(\mathbf{X})$ is computably compact, as it contains \emptyset as a computable bottom element. Then $\mathcal{O}(\mathbf{X}) \times \mathcal{O}(\mathbf{X})$ is computably compact as a product, and finally the claim follows by noting that a computably closed subspace of a computably compact space is computably compact and invoking Observation 6. ◀

► **Lemma 8.** If \mathbf{X} is computably separable, effectively countably based and computably Hausdorff, then $\text{DPO}(\mathbf{X})$ is a computable element of $\mathcal{V}(\mathcal{O}(\mathbf{X}) \times \mathcal{O}(\mathbf{X}))$.

Proof. It is shown in [25] that under the given conditions, we can obtain an adequate formal disjointness notion on basic open sets. We can then obtain a dense sequence in $\text{DPO}(\mathbf{X})$ by constructing pairs of finite unions of basic open sets with the additional requirements that each basic open set is formally disjoint from all basic open sets listed in the opposite finite union. ◀

► **Corollary 9.** If \mathbf{X} is computably separable, effectively countably based and computably Hausdorff, then $\text{DPO}(\mathbf{X})$ is computably compact and computably overt.

► **Definition 10.** Given $f \in \mathcal{C}(\mathbf{X}, \mathbf{X})$, $(U, V) \in \text{DPO}(\mathbf{X})$ and $\mu \in \mathcal{PM}(\mathbf{X})$, we say that (U, V) is μ -invariant under f , iff:

$$\mu(U) + \mu(f^{-1}(V)) \leq 1$$

► **Observation 11.** (U, V) being μ -invariant under f is a computably closed property.

► **Lemma 12.** Let \mathbf{X} be computably separable, effectively countably based and computably Hausdorff. Then “all pairs from $\text{DPO}(\mathbf{X})$ are μ -invariant under f ” is a computably closed property in $\mu \in \mathcal{PM}(\mathbf{X})$ and $f \in \mathcal{C}(\mathbf{X}, \mathbf{X})$.

Proof. Computably closed properties are closed under universal quantification over computably overt sets. So we just combine Observation 11 and Corollary 9. ◀

3.2 Proof of Theorem 5

To be able to invoke the results of the previous subsection we need to relate invariance of disjoint pairs of open sets to invariance of individual open sets.

► **Lemma 13.** *For a computable metric space \mathbf{X} , $\mu \in \mathcal{PM}(\mathbf{X})$ and $f \in \mathcal{C}(\mathbf{X}, \mathbf{X})$ the following are equivalent:*

1. *All pairs from $\text{DPO}(\mathbf{X})$ are μ -invariant under f .*
2. *For all $U \in \mathcal{O}(\mathbf{X})$ it holds that $\mu(U) = \mu(f^{-1}(U))$.*

Proof. 2. implies

$$\mu(U) + \mu(f^{-1}(V)) = \mu(f^{-1}(U)) + \mu(f^{-1}(V)) \stackrel{(*)}{=} \mu(f^{-1}(U) \cup f^{-1}(V)) \leq 1$$

with (*) since $f^{-1}(U)$ and $f^{-1}(V)$ are disjoint.

For the converse, assume that U witnesses that f is not invariant, i.e. $\mu(U) \neq \mu(f^{-1}(U))$. We shall argue that this implies the existence of a disjoint pair of open sets which is not μ -invariant under f . Let $\delta = \frac{1}{3}|\mu(U) - \mu(f^{-1}(U))|$. Consider the sets $B_{-\varepsilon}(U) = \{x \in \mathbf{X} \mid d(x, U^C) > \varepsilon\}$. Since $U = \bigcup_{\varepsilon > 0} B_{-\varepsilon}(U)$ is a nested union and f is continuous, we find that $\mu(U) = \sup_{\varepsilon > 0} \mu(B_{-\varepsilon}(U))$ and $\mu(f^{-1}(U)) = \sup_{\varepsilon > 0} \mu(f^{-1}(B_{-\varepsilon}(U)))$. Consequently, there exists some ε_0 such that for all $\varepsilon < \varepsilon_0$ it holds that $|\mu(U) - \mu(B_{-\varepsilon}(U))| < \delta$ and $|\mu(f^{-1}(U)) - \mu(f^{-1}(B_{-\varepsilon}(U)))| < \delta$.

Next, consider the sets $D_{-\varepsilon}(U) := \{x \in \mathbf{X} \mid d(x, U^C) = \varepsilon\}$. Since for different ε these sets are disjoint, we know that for only countably many ε can it hold that $\mu(D_{-\varepsilon}(U)) > 0$. The sets $f^{-1}(D_{-\varepsilon}(U))$ are disjoint, too, and thus the same argument applies. We can thus select some $\varepsilon_1 < \varepsilon_0$ such that $\mu(D_{-\varepsilon_1}(U)) = \mu(f^{-1}(D_{-\varepsilon_1}(U))) = 0$. This ensures that $\mu(B_{-\varepsilon_1}(U)) + \mu((B_{-\varepsilon_1}(U)^C)^\circ) = 1$ and $\mu(f^{-1}(B_{-\varepsilon_1}(U))) + \mu(f^{-1}((B_{-\varepsilon_1}(U)^C)^\circ)) = 1$. Moreover, we know that $|\mu(B_{-\varepsilon_1}(U)) - \mu(f^{-1}(B_{-\varepsilon_1}(U)))| > \delta$ from $\varepsilon_1 < \varepsilon_0$, so depending on the sign of the difference, either $(B_{-\varepsilon_1}(U), (B_{-\varepsilon_1}(U)^C)^\circ)$ or $((B_{-\varepsilon_1}(U)^C)^\circ, B_{-\varepsilon_1}(U))$ is not μ -invariant under f . ◀

Proof of Theorem 5. By Lemma 13 we can replace the invariance for open sets by invariance for disjoint pairs of open sets. By Lemma 12, this is a computably closed property for each fixed choice of continuous function $y \mapsto g(x, y)$. The additional universal quantification over the computably overt space \mathbf{X} preserves being a computably closed predicate. ◀

4 Explicit computation of the Haar measure

The synthetic arguments from Section 3 establishing computability (Theorem 2) do not immediately exhibit an actual algorithm. To this end, the present section takes a more explicit approach. Its assumptions superficially differ but will be shown equivalent (in a sense to be formalized) in Section 5. Among others, we suppose computability of the *size of maximum packings*. This is a notion asymptotically related to, yet in detail (maximum packing vs. minimum covering, open vs. closed balls) subtly different from, Kolmogorov's metric entropy [16], to the *separation bound* from [36, Definition 6.2], and to the *capacity* from [13, Definition 12]. All three notions can be regarded as integer Skolemizations (i.e. *moduli*) of total boundedness [15, Def 17.106].

► **Definition 14.** *For any compact metric space (X, d) and its subset $U \subseteq X$,*

1. *$T \subseteq U$ is called an n -packing of U if $\forall x, y \in T (x \neq y) \rightarrow d(x, y) > 2^{-n}$.*
2. *An n -packing T is maximum if $|T| \geq |S|$ for every n -packing S of U .*

3. $\{T_n\}_{n=1}^{\infty}$ is a sequence of maximum packings if each T_n is a maximum n -packing.
4. $\kappa_U : \mathbb{N} \rightarrow \mathbb{N}$ is the size of maximum packings of U if $\kappa_U(n) = |T_n|$ where T_n is a maximum n -packing.

If $U = X$, the term “of U ” is omitted.

Our definition features strict inequality of pairwise distances: this asserts that a maximum n -packing T_n can be found algorithmically by exhaustive search, provided that its size is given/computable.

► **Theorem 15.** *Let (X, d) be a computable metric space and $(X, e, \circ, \cdot^{-1})$ a compact topological group. Suppose that the metric d is bi-invariant:*

$$\forall a, b, c \in X : d(a \circ c, b \circ c) = d(a, b) = d(c \circ a, c \circ b)$$

And suppose that the size of maximum packings $\kappa_X : \mathbb{N} \rightarrow \mathbb{N}$ is computable. Then the Haar integral $\mathcal{C}(X) \ni f \mapsto \int_X f d\mu$ is computable.

Recall [35, §8.1] that a computable metric space (X, d) comes with a dense sequence $\xi : \mathbb{N} \rightarrow X$ such that the real double sequence $d : \mathbb{N} \times \mathbb{N} \ni (a, b) \mapsto d(\xi(a), \xi(b))$ is computable. Note that, as opposed to Theorem 2, we do not suppose the group operation \circ (nor neutral element nor inversion) to be computable but instead require the metric to be bi-invariant. See Section 5 for a comparison between the different hypotheses.

4.1 Mathematical Estimates of Haar Measures

Invariance of both metric d and Haar measure μ implies that the content $\mu(B)$ of an open ball $B = B_r(c)$ depends only on its radius r , but not on its center c . Intuitively, for a sufficiently large maximum packing T , said volume should be approximated by the ratio of points in B to the total number of points (Definition 18). If $B_r(c)$ contains significantly smaller a fraction, then by double counting some other $B_r(c')$ would need to “compensate” with a larger fraction, hence invariance suggests that more points can be added to T at $B(r, c)$ as well, contradicting maximality. Lemma 17 below formalizes this idea both in its statement and proof.

► **Definition 16.** *For a metric space (X, d) and its subset $U \subseteq X$, we introduce the outer generalized closed ball as $\overline{B}_r(U) := \{x \in X \mid d(x, U) \leq r\}$. Similarly, the inner generalized closed ball is introduced as $\overline{B}_{-r}(U) = \{x \in X : d(x, U^c) \geq r\}$.*

For $0 \leq r, s$ it holds

$$\overline{B}_{+r}(\overline{B}_{-r}(U)) \subseteq \overline{U} \subseteq \overline{B}_{-r}(\overline{B}_{+r}(U)), \quad \overline{B}_{+r}(\overline{B}_{+s}(U)) \subseteq \overline{B}_{+r+s}(U) \quad (2)$$

► **Lemma 17.** *Suppose (X, d, \circ) is a compact topological group with bi-invariant metric d and a maximum n -packing T_n of size $\kappa_X(n)$. Then for any $x \in X$ and measurable $U \subseteq X$ it holds:*

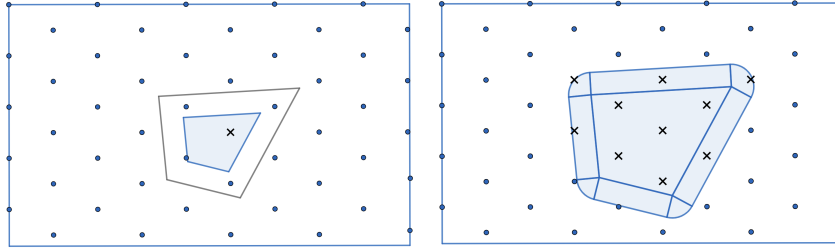
$$\kappa_{\overline{B}_{-2-n}(U)}(n) \leq |T_n \cap xU| \leq \kappa_{\overline{B}_{2-n}(U)}(n)$$

► **Definition 18.** *Abbreviate $\mu_T := \frac{1}{|T|} \sum_{p \in T} \delta_p$ where δ_p denotes the Dirac measure.*

► **Lemma 19.** *Let (X, d, \circ) be a compact topological group with bi-invariant metric d and Haar measure μ , and T_n a maximum n -packing. Then for any $U \subseteq X$:*

$$\mu(\overline{B}_{-2-n+2}(U)) \leq \mu_{T_n}(\overline{B}_{-2-n+1}(U)) \leq \mu(U) \leq \mu_{T_n}(\overline{B}_{2-n+1}(U)) \leq \mu(\overline{B}_{2-n+2}(U))$$

For the illustration of Lemma 19, see Figure 1.



■ **Figure 1** Illustration of Lemma 19. A blue rectangle represents the space. Blue points represent the maximum n -packing. A black shape represents U . Blue colored shapes represent inner and outer generalized balls. Counting cross-marked points and dividing it by the number of (any) points gives $\mu_{T_n}(\overline{B}_{2^{-n+1}}(U))$ and $\mu_{T_n}(B_{2^{-n+1}}(U))$.

4.2 Algorithmic Approximation of Haar Measures

Our strategy to compute $\mu(U)$ with Lemma 19 is to compute μ_{T_n} and assert that the sequence of intervals $\{ [\mu_{T_n}(\overline{B}_{2^{-n+1}}(U)), \mu_{T_n}(B_{2^{-n+1}}(U))] \}_{n=1}^\infty$ include $\mu(U)$, and that its length converges to zero. However, there are two obstacles:

1. μ_T is discrete (i.e. the value of $\mu_T(U)$ can jump even by a small perturbation to U), which makes it uncomputable.
2. If $\lim_{r \rightarrow 0} \mu(\overline{B}_r(U)) = \mu(U)$, then Lemma 19 guarantees that the length of the interval converges. However, the hypothesis may not hold in general.

This section works around these obstacles and gives an algorithm that can compute the measure of sufficiently rich a class of sets to perform the integration.

The first thing to address is μ_T . It is not computable, but procedure `pseudoCount` below can bound its measure on closed sets whose distances to any given points is computable. The latter condition is known as being (Turing-) located [10]:

► **Definition 20.** A closed subset S of a computable metric space (X, d) is located if the continuous function $X \ni p \mapsto d(p, S) \in \mathbb{R}$ is computable.

Located sets are sometimes called computably closed sets, but being located is different from being a computable element of $\mathcal{A}(X)$.

Our workaround to the second obstacle is, instead of trying to compute the measure of every closed set, to effectively “approximate” the given set by those satisfying the convergence condition and to compute their measure. Let us define such sets first:

► **Definition 21.** On a topological space (X, τ) with a Borel measure μ , call a measurable set U co-inner regular iff

$$\mu(U) = \sup \{ \mu(V) \mid V \subseteq U \text{ open and measurable} \} .$$

On a compact metric group (X, d, \circ) with the Haar measure μ where d is bi-invariant, a real number $r > 0$ is a co-inner regular radius iff for some/all $p \in X$, the ball $\overline{B}_r(p)$ is co-inner regular.

Indeed, invariance of d and μ implies that $\overline{B}_r(p)$ is co-inner regular iff $\overline{B}_r(q)$ is. Note that since Haar measures are regular, on a compact metric group with a bi-invariant metric and a Haar measure, if a set U is co-inner regular, then $\mu(\partial U) = 0$, giving $\lim_{r \rightarrow 0} \mu(\overline{B}_r(U)) = \mu(U)$.

► **Lemma 22.** Let (X, d, \circ) be a compact topological group with bi-invariant metric d , Haar measure μ , and computable size κ_X of maximum packings. If the closure of U is located and co-inner regular, then the procedure `computeMeasure` computes its measure $\mu(U)$.

Procedure `computeMeasure`($U, \{T_m\}_{m=1}^{\infty}, n$).

<p>Data: \bar{U} located co-inner regular set, $\{T_m\}_{m=1}^{\infty}$ computable sequence of maximum packings, n target precision</p> <p>Result: A rational number q s.t. $q - \mu(S) \leq 2^{-n}$.</p> <p>error $\leftarrow \infty$;</p> <p>$m \leftarrow 0$;</p> <p>while error $> 2^{-n}$ do</p> <p style="padding-left: 2em;">$r \leftarrow 2^{-m}$;</p> <p style="padding-left: 2em;">$a \leftarrow \text{pseudoCount}(\bar{B}_{-r}(\bar{U}), T_m, m + 1)$;</p> <p style="padding-left: 2em;">$b \leftarrow \text{pseudoCount}(\bar{B}_{r/2}(\bar{U}), T_m, m + 1)$;</p> <p style="padding-left: 2em;">error $\leftarrow b - a$;</p> <p style="padding-left: 2em;">$m \leftarrow m + 1$;</p> <p>end</p> <p>return any $p \in \text{interval}$</p>

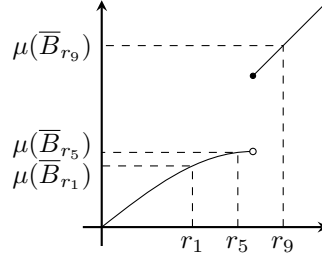
Procedure `pseudoCount`(S, T, n).

<p>Data: S a located set, T a finite set of points, n error parameter, <code>dist</code>(p, S, m) approximate distance between $p \in T$ and S up to 2^{-m}.</p> <p>Result: A rational q where $\mu_T(S) \leq q \leq \mu_T(\bar{B}_{2^{-n}}(S))$</p> <p>count $\leftarrow 0$;</p> <p>foreach $p \in T$ do</p> <p style="padding-left: 2em;">if <code>dist</code>($p, S, n + 2$) $< 2^{-n-1}$ then</p> <p style="padding-left: 4em;">count $\leftarrow \text{count} + 1$;</p> <p style="padding-left: 2em;">end</p> <p>end</p> <p>return $\frac{\text{count}}{ T }$</p>

Note that `computeMeasure` in turn calls `pseudoCount`(p, S, n).

Not every closed ball is co-inner regular, but “sufficiently” many are: Co-inner regular radii can be effectively found to compute Haar measures in the form of the Haar integral by `findCoInnerRegularRadius`. Figure 2 illustrates the procedure `findCoInnerRegularRadius`. $\lambda x. \text{findCoInnerRegularRadius}(a, b, \{T_m\}_{m=1}^{\infty}, x)$ is a nested sequence of intervals that converges to a co-inner regular radius. `findCoInnerRegularRadius` achieves this by recursively dividing and outputting the interval. That is, `findCoInnerRegularRadius`($a, b, \{T_m\}_{m=1}^{\infty}, n$) first computes the $(n - 1)$ -th interval (`findCoInnerRegularRadius`($a, b, \{T_m\}_{m=1}^{\infty}, n - 1$)) and outputs the n -th interval by dividing it. The procedure divides the $(n - 1)$ -th interval into two parts $[r_1, r_5]$ and $[r_5, r_9]$, computes corresponding measures $\mu(\bar{B}_{r_1}), \mu(\bar{B}_{r_5}), \mu(\bar{B}_{r_9})$, and picks the interval which has smaller difference of measures. In this case, since $\mu(\bar{B}_{r_5}) - \mu(\bar{B}_{r_1}) \leq \mu(\bar{B}_{r_9}) - \mu(\bar{B}_{r_5})$, $[r_1, r_5]$ is picked. This strategy makes the difference of measures converges to zero since it is always, at least, halved on each iterations. This gives a co-inner regular radius, because in fact co-inner regular radii are continuity points of the function $r \mapsto \mu(\bar{B}_r)$.

► **Lemma 23.** *Procedure `findCoInnerRegularRadius` computes a co-inner regular radius in the form of $\lambda x. \text{findCoInnerRegularRadius}(a, b, \{T_m\}_{m=1}^{\infty}, x)$.*



■ **Figure 2** Illustration of procedure `findCoInnerRegularRadius`. Here, an example graph of the discontinuous function $r \mapsto \mu(\overline{B}_r)$ is shown. Note that $\mu(\overline{B}_r(p)) = \mu(\overline{B}_r(q))$ because of the invariance of the metric and the Haar measure.

Procedure `findCoInnerRegularRadius`($a, b, \{T_m\}_{m=1}^\infty, n$).

Data: $a < b$ rational bounds between which to look for a co-inner regular radius, $\{T_m\}_{m=1}^\infty$ sequence of maximum packings, n target precision.

Result: Rational bounds a_n, b_n s.t. $(a < a_{n-1} < a_n < b_n < b_{n-1} < b) \wedge (b_n - a_n \leq 2^{-n}) \wedge |\mu(\overline{B}_{a_n}(p)) - \mu(\overline{B}_{b_n}(p))| \leq 2^{-n}$ for any/all $p \in X$.

$(a_{n-1}, b_{n-1}) \leftarrow \text{findCoInnerRegularRadius}(a, b, \{T_m\}_{m=1}^\infty, n-1)$;

$r_1, r_5, r_9 \leftarrow \frac{9a_{n-1}+1b_{n-1}}{10}, \frac{5a_{n-1}+5b_{n-1}}{10}, \frac{1a_{n-1}+9b_{n-1}}{10}$;

Pick sufficiently large N s.t. $2^{-N+2} \leq \frac{b_{n-1}-a_{n-1}}{10}$;

Compute an element $p \in X$ using the fact that X is a computable metric space;

$m_1, m_5, m_9 \leftarrow \text{pseudoCount}(\overline{B}_{r_1}(p), T_N, N), \text{pseudoCount}(\overline{B}_{r_5}(p), T_N, N),$

$\text{pseudoCount}(\overline{B}_{r_9}(p), T_N, N)$;

$\epsilon \leftarrow \frac{b_{n-1}-a_{n-1}}{10}$;

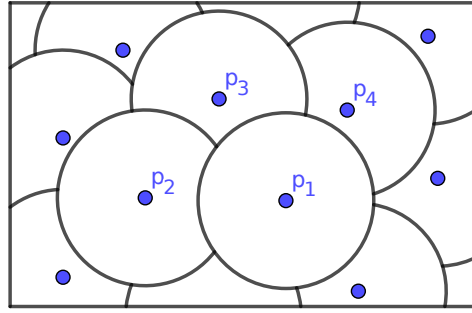
if $m_9 - m_5 \leq m_5 - m_1$ **then return** $[r_1 + \epsilon, r_5 - \epsilon]$ **else return** $[r_5 + \epsilon, r_9 - \epsilon]$;

Proof. $\lambda x. \text{findCoInnerRegularRadius}(a, b, \{T_m\}_{m=1}^\infty, x)$ represents $r := \lim_{n \rightarrow \infty} a_n$, where a_n is the first element of the interval that `findCoInnerRegularRadius` outputs. r is a co-inner regular radius because the fact $r \in (a_n, b_n)$ makes $\partial \overline{B}_r(p) \subseteq \overline{B}_{b_n}(p) \setminus \overline{B}_{a_n}(p)$, which leads to $\mu(\partial \overline{B}_r(p)) \leq |\mu(\overline{B}_{b_n}(p)) - \mu(\overline{B}_{a_n}(p))| \leq 2^{-n}$ for any n . This implies $\mu(\partial \overline{B}_r(p)) = 0$.

Now it is sufficient to prove the postconditions. Let us only prove $\mu(\overline{B}_{a_n}(p)) - \mu(\overline{B}_{b_n}(p)) \leq 2^{-n}$, since others are straightforward. Let $r_i := \frac{ia_{n-1}+(10-i)b_{n-1}}{10}$ and $m_i := \mu(\overline{B}_{r_i}(p))$. Because of Lemma 19 and the fact that N is sufficiently large, $\mu(\overline{B}_{r_i}(p)) \leq \mu_{T_N}(\overline{B}_{r_{i+1}}(p)) \leq m_{i+1} \leq \mu_{T_N}(\overline{B}_{r_{i+1}+2^{-N}}(p)) \leq \mu(\overline{B}_{r_{i+2}}(p))$. Then since $2^{-n+1} \geq |\mu(\overline{B}_{a_{n-1}}(p)) - \mu(\overline{B}_{b_{n-1}}(p))| \geq |m_9 - m_1| \geq |m_9 - m_5| + |m_5 - m_1|$, WLOG $|m_5 - m_1| \leq 2^{-n}$. Then $|\mu(\overline{B}_{r_5-\epsilon}(p)) - \mu(\overline{B}_{r_1+\epsilon}(p))| \leq |m_5 - m_1| \leq 2^{-n}$. ◀

4.3 Main Algorithm for Haar Integration

Explicit algorithm of Theorem 15. The procedure `computeIntegral` computes the Haar integral $\int_X f d\mu$. Generalizing classical Riemann sums, it partitions X into subsets $U_i, i \leq N$, of sufficiently small diameter (see Figure 3): given by a modulus of continuity such that f on each U_i varies by at most 2^{-n} . Then it sums those values of $f|_{U_i}$, each weighted by the measure of U_i . In order to invoke `computeMeasure`, we want the U_i to be located and co-inner regular: as provided by `findNicePartition`. Specifically, each U_i will be of the form $U_i = \overline{B}_R(p_i) \setminus \bigcup_{j < i} \overline{B}_R(p_j)$ for $p_1, \dots, p_N \in T_m$ and real $R > 0$ provided by `findCoInnerRegularRadius`.



■ **Figure 3** Consider the whole rectangle as the whole space. Then this is how the procedure `computeIntegral` partitions the space. For example, the subset containing p_1 is $\overline{B}_R(p_1)$. Similarly, the subset containing p_2 is $\overline{B}_R(p_2) \setminus \overline{B}_R(p_1)$, the subset containing p_3 is $\overline{B}_R(p_3) \setminus (\overline{B}_R(p_1) \cup \overline{B}_R(p_2))$, and so on. Then the subsets of the partition are of the form $\overline{B}_R(p_i) \setminus \bigcup_{j < i} \overline{B}_R(p_j)$.

Procedure `computeIntegral`($f, \{T_m\}_{m=1}^{\infty}, n$).

<p>Data: real function f, sequence of maximum packings $\{T_m\}_{m=1}^{\infty}$, target precision n</p> <p>Result: a rational number q s.t. $q - \int_X f d\mu \leq 2^{-n}$</p> <p>$m_f \leftarrow \text{modulus}(f, n + 1)$; // modulus is from [14, Definition 2.12]</p> <p>$\{U_i\}_{i=1}^N \leftarrow \text{findNicePartition}(\{T_m\}_{m=1}^{\infty}, m_f)$;</p> <p>$M \leftarrow \text{bound}(f)$;</p> <p>foreach U_i <i>in</i> $\{U_i\}_{i=1}^N$ do</p> <p style="padding-left: 20px;">$p_i \leftarrow \text{center}(U_i)$;</p> <p style="padding-left: 20px;">$m_i \leftarrow \text{computeMeasure}(U_i, \{T_m\}_{m=1}^{\infty}, n + 1 + i + \log M)$;</p> <p>end</p> <p>return $\sum_{i=1}^N m_i f(p_i)$</p>
--

Recall the comment after Definition 14 that we may suppose a sequence T_m of maximum packings is given. ◀

5 Discussion of Hypotheses

While the requirements of Theorem 2 and Theorem 15 appear to be very different, it turns out that actually, both theorems are applicable in the very same cases.

For one direction, suppose we have a computably compact computable metric space (\mathbf{X}, d) with a computable group operation \circ . Then¹

$$d'(a, b) := \sup_{x \in \mathbf{X}} \sup_{y \in \mathbf{X}} d(x \circ a \circ y, x \circ b \circ y)$$

constitutes a topologically equivalent and also computable, but now bi-invariant, metric. The size of maximum packings may be non-computable for a CCCMS. However, for any CCCMS there is a computable sequence of radii converging to zero for which we can compute the maximum packings. It is straightforward to see that this suffices for Theorem 15. As such, we see that the requirements for Theorem 15 are implied by those of Theorem 2.

¹ We appreciate relevant discussion on MathOverflow [22].

34:12 Computing Haar Measures

Procedure findNicePartition($\{T_m\}_{m=1}^\infty, n$).

Data: $\{T_m\}_{m=1}^\infty$ is a sequence of maximum packings, n is the target precision
Result: A partition $P = \{U_i\}_{i=1}^N$ s.t. each $\overline{U_i}$ is a located co-inner regular set of the form $U_i = \overline{B}_R(p_i) \setminus \bigcup_{j < i} \overline{B}_R(p_j)$.
 $P \leftarrow \{\}$;
 $R \leftarrow \lambda x. \text{findCoInnerRegularRadius}((2^{-n-1}, 2^{-n}), \{T_m\}_{m=1}^\infty, x)$;
foreach p_i **in** T_{n+1} **do**
 $U_i \leftarrow \overline{B}_R(p_i) \setminus \bigcup_{U \in P} U$;
 $P \leftarrow P \cup \{U_i\}$;
end
return P

For the converse direction, note that Theorem 15 does not suppose the group operation \circ (nor neutral element nor inversion) to be computable. Indeed, a group operation on a CCCMS can have a computable bi-invariant metric but fail to be computable itself. This is due to the potential for many different group operations to have the same bi-invariant metric:

► **Example 24.** Fix some $A \subseteq \mathbb{N}$. Let $G_n := IZ_{p_n^2}$ if $n \in A$, and let $G_n := \mathbb{Z}_{p_n} \times \mathbb{Z}_{p_n}$ if $n \notin A$, where p_n is the n -th prime. Note that both have cardinality p_n^2 but are not isomorphic as additive groups yet both have the same Haar measure under the bi-invariant discrete metric. Now let $G_A := \prod_{n \in A} G_n$, equipped with the Baire space metric. For $A \neq B$ we find that G_A and G_B are not homeomorphic. The group operation on G_A is computable iff A is decidable. However, both the bi-invariant metric structure on G_A and the Haar measure are all independent of A , and computable.

Interestingly, the Haar measure on a compact group is determined already by an invariant metric and independent of the potentially many different underlying group operations:

► **Corollary 25.** *Consider a compact metric space (X, d) with two group operations \circ and \circ' both rendering d left-invariant. Then (X, \circ) and (X, \circ') induce the same Haar measure.*

Proof. In the metric case, the net of neighborhoods B of e from Fact 3 becomes a sequence of open balls $B_{1/2^n}(e)$. Left-invariance implies that all translates $q \cdot B_r(e) = B_r(q)$ have the same measure; and by the group property, every open ball $B_r(q)$ is a translate of $B_r(e)$: for both \circ and \circ' . In particular, $[X : B_{1/2^n}] = \kappa_X(n)$. ◀

On the other hand the collection of different group operations \circ to a given bi-invariant metric d is “tame”:

► **Lemma 26.** *Let (\mathbf{X}, d) be a CCCMS. The set $O \subseteq \mathcal{C}(\mathbf{X} \times \mathbf{X}, \mathbf{X})$ of group operations rendering d bi-invariant is a computably compact set.*

Proof. If d is bi-invariant for $\circ \in \mathcal{C}(\mathbf{X} \times \mathbf{X}, \mathbf{X})$, the triangle inequality gives

$$d(a \circ x, b \circ y) \leq d(a \circ x, b \circ x) + d(b \circ x, b \circ y) = d(a, b) + d(x, y) \leq 2d((a, x), (b, y)) \quad ;$$

rendering \circ 2-Lipschitz with respect to the maximum metric on $\mathbf{X} \times \mathbf{X}$. By the effective Arzelà-Ascoli theorem, the subset of 2-Lipschitz $f \in \mathcal{C}(\mathbf{X} \times \mathbf{X}, \mathbf{X})$ is computably compact.

Within this set, we are interested in those satisfying the bi-invariance and group axioms:

$$\begin{aligned} \forall a, b, c : \quad & d(f(c, a), f(c, b)) = d(a, b) = d(f(a, c), f(b, c)) \\ \exists e, a' : \quad & f(a, e) = a = f(e, a) \wedge f(f(a, b), c) = f(a, f(b, c)) \wedge f(a, a') = e = f(a', a) \end{aligned}$$

These are computably closed predicates since the quantification is over the computably compact and computably overt space \mathbf{X} . This ends the proof, since a computably closed subset of a computably compact set is computably compact. \blacktriangleleft

We can combine Corollary 25 and Lemma 26 to see that Theorem 5 also implies that from a CCCMS (X, d) such that some group operation is bi-invariant for d we can compute the Haar measure for any such group operation.

To conclude this section, we shall consider a family of examples that show that we need more computability requirements than that of the metric and of the group operation. We consider the closed subgroups of $(\mathbf{2}^{\mathbb{N}}, \oplus)$, where \oplus denotes the componentwise exclusive or. These subgroups are of the form

$$G_A := \{p \in \mathbf{2}^{\mathbb{N}} \mid \forall n \in A \ p(n) = 0\}$$

for some $A \subseteq \mathbb{N}$. Each G_A inherits compactness, computable metrizable and the computability of the group operation from $(\mathbf{2}^{\mathbb{N}}, \oplus)$.

G_A is computably compact iff A is c.e., and effectively separable (and thus a computable metric space) iff A is co-c.e. Now if we have the Haar measure μ_A on G_A , we can recover A since $\mu_A(\{p \in G_A \mid p(n) = 1\}) = \frac{1}{2}$ iff $n \notin A$ and $\mu_A(\{p \in G_A \mid p(n) = 1\}) = 0$ iff $n \in A$. We thus see that G_A is a CCCMS iff μ_A is computable – so neither computable compactness or computable separability are dispensable for the computability of the Haar measure.

If we already have a bi-invariant metric, computable compactness is even necessary:

► Theorem 27. *Let (\mathbf{X}, d) be a computable metric space with computable probability measure μ such that $\mu(B_r(x))$ depends only on r but not on x . Then \mathbf{X} is computably compact.*

6 Computational Complexity of the Haar Integral

We now move beyond mere computability of the Haar measure, and consider the computational complexity of this task for the groups $G = \mathcal{SO}(3)$, $G = \mathcal{O}(3)$, $G = \mathcal{SU}(2)$, and $G = \mathcal{U}(2)$. In each case, the complexity turns out to be closely related to the complexity class $\#\mathcal{P}_1$. We prove Theorem 4, namely

- a) For every polynomial-time computable $f \in \mathcal{C}(G)$, $\int_G f \in \mathbb{R}$ is computable in polynomial space (and exponential time).
- b) If $\mathcal{FP}_1 = \#\mathcal{P}_1$ and $f \in \mathcal{C}(G)$ is polynomial-time computable, then so is $\int_G f \in \mathbb{R}$.
- c) There exists a polynomial-time computable $f \in \mathcal{C}(G)$ such that polynomial-time computability of $\int_G f \in \mathbb{R}$ implies $\mathcal{FP}_1 = \#\mathcal{P}_1$.

To this end recall [14, Theorem 5.32] that (a), (b), and (c) are known for definite Riemann integration

$$\mathcal{C}[0; 1] \ni \tilde{f} \mapsto \int_0^1 \tilde{f}(t) dt \in \mathbb{R} .$$

Moreover, Item (c) remains true for $\tilde{f} \in \mathcal{C}_0^\infty[0; 1]$: the class of smooth (infinitely often differentiable) $\tilde{f} : [0; 1] \rightarrow \mathbb{R}$ such that $\tilde{f}(0) = 0 = \tilde{f}(1)$; cmp. [8, 33].

34:14 Computing Haar Measures

Before proceeding to the groups $\mathcal{SO}(3)$, $\mathcal{O}(3)$, $SU(2)$, $\mathcal{U}(2)$, recall the argument for the case $\mathcal{U}(1) = \{ \exp(2\pi it) : 0 \leq t \leq 1 \}$ equipped with complex multiplication and the Haar integral

$$\mathcal{C}(\mathcal{U}(1)) \ni f \mapsto \int_0^1 f(\exp(2\pi it)) dt \quad :$$

So to see (c), consider the polynomial-time computable embedding

$$\mathcal{C}_0[0; 1] \ni \tilde{f} \mapsto (\exp(2\pi it) \mapsto \tilde{f}(t)) \in \mathcal{C}(\mathcal{U}(1)) \quad .$$

And to see (a) and (b) for $G = \mathcal{U}(1)$, consider the polynomial-time computable embedding

$$\mathcal{C}(\mathcal{U}(1)) \ni f \mapsto (t \mapsto f(\exp(2\pi it))) \in \mathcal{C}[0; 1] \quad .$$

This also covers $\mathcal{SO}(2) \cong \mathcal{U}(1)$; and integration over $\mathcal{O}(2) \cong \mathcal{SO}(2) \times \{\pm 1\}$ amounts to two integrals over $\mathcal{SO}(2)$.

Let $\mathbb{H} = \{ \alpha + i\beta + j\gamma + k\delta : \alpha, \beta, \gamma, \delta \in \mathbb{R} \}$ denote the quaternions, parameterized as real quadruples with respect to units $1, i, j, k$. The group $SU(2)$ is well-known, and easily verified to be, isomorphic to the multiplicative group \mathbb{H}_1 of quaternions of norm 1 (aka *versors*) via isomorphism

$$\mathbb{H}_1 \ni \alpha + i\beta + j\gamma + k\delta \mapsto \begin{pmatrix} \alpha + i\beta & -\gamma + i\delta \\ \gamma + i\delta & \alpha - i\beta \end{pmatrix} \in SU(2) \quad (3)$$

with $|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$. Reparameterize \mathbb{H}_1 in generalized spherical coordinates

$$\begin{aligned} [0; \pi) \times [0; \pi) \times [0; 2\pi) &\ni (\eta, \vartheta, \varphi) \mapsto \Psi(\eta, \vartheta, \varphi) := \\ &\cos(\eta) + i \sin(\eta) \cos(\vartheta) + j \sin(\eta) \sin(\vartheta) \cos(\varphi) + k \sin(\eta) \sin(\vartheta) \sin(\varphi) \in \mathbb{H}_1 \end{aligned}$$

with Jacobian determinant $|\det(\Psi'(\eta, \vartheta, \varphi))| = \sin^2(\eta) \sin(\vartheta)$, and verify that integration by change-of-variables

$$\mathcal{C}(\mathbb{H}_1) \ni f \mapsto \int_0^{2\pi} \int_0^\pi \int_0^\pi f(\Psi(\eta, \vartheta, \varphi)) \cdot |\det \Psi'(\eta, \vartheta, \varphi)| d\eta d\vartheta d\varphi \quad (4)$$

is left-invariant, hence must coincide with the Haar integral on $SU(2)$. Items (a) and (b) thus follow by polynomial-time reduction to Euclidean/Riemann integration according to Equation (4). And Item (c) follows by polynomial-time embedding

$$\begin{aligned} \mathcal{C}(\mathcal{U}(1)) \ni f \mapsto \tilde{f} \in \mathcal{C}(\mathbb{H}_1) \cong \mathcal{C}(SU(2)), \quad \text{where} \\ \tilde{f} : \mathbb{H}_1 \ni \alpha + i\beta + j\gamma + k\delta \mapsto f\left(\frac{\alpha + i\beta}{\sqrt{\alpha^2 + \beta^2}} \cdot \sqrt{\alpha^2 + \beta^2}\right) \in \mathbb{R} \quad . \end{aligned}$$

Since continuous f on compact \mathbb{H}_1 is bounded, $f\left(\frac{\alpha + i\beta}{\sqrt{\alpha^2 + \beta^2}} \cdot \sqrt{\alpha^2 + \beta^2}\right) \rightarrow 0$ as $\alpha^2 + \beta^2 \searrow 0$. Hence \tilde{f} is indeed well-defined and remains polynomial-time computable by continuous extension also for $\alpha^2 + \beta^2 = 0$.

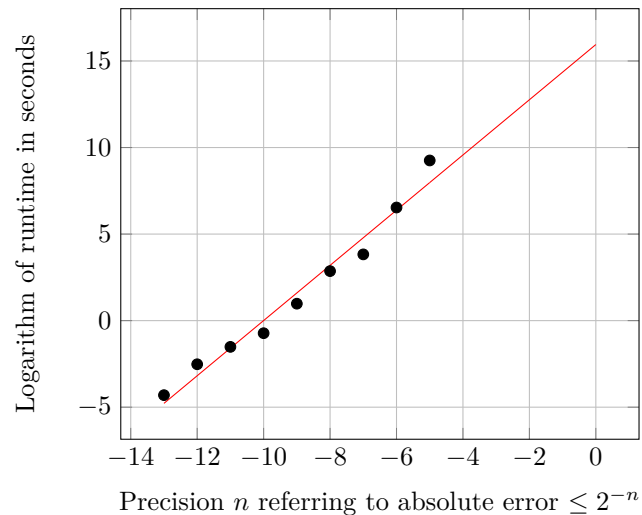
$\mathcal{SO}(3)$ is doubly-covered by \mathbb{H}_1 , identifying $q \in \mathbb{H}_1$ with special orthogonal linear map

$$\mathbb{R}^3 \ni (\beta, \gamma, \delta) \mapsto (\beta', \gamma', \delta') : i\beta' + j\gamma' + k\delta' \stackrel{!}{=} (q \cdot (i\beta + j\gamma + k\delta) \cdot q^{-1}) \quad .$$

Moreover $\mathcal{O}(3) \cong \mathcal{SO}(3) \times \{\pm 1\}$ and $\mathcal{U}(2) \cong SU(2) \times \mathcal{U}(1)$. ◀

6.1 Implementation and Evaluation

Based on the above reduction to ordinary Riemann integration, we have implemented integration on $SU(2)$ in the `iRRAM C++` library [18]. The source code is available at <http://github.com/realcomputation/irramplus/tree/master/HAAR>. Its empirical evaluation produced the following timing results:



■ **Figure 4** Empirical evaluation of rigorous integration on $SU(2)$.

Specifically, we chose the Lipschitz-continuous function $\mathbb{H}_1 \ni w + xi + yj + zk \mapsto |w| + |x| + |y| + |z| \in \mathbb{R}$ to integrate *without* letting the algorithm exploit its particular symbolic form and symmetry. Time measurements were performed on the virtual machine that has Ubuntu 64-bit with 4 cores and 8GB RAM by VMware Workstation 15 Player. The underlying computer has Intel(R) Core(TM) i7-7700K CPU 4.20GHz and 16GB RAM. Execution time for each precision is the average execution time of 5 executions.

Note that the y-axis records the *logarithm* of the execution time in seconds. This time is confirmed to grow exponentially with the output precision parameter n : as expected for a $\#P_1$ -complete problem.

7 Conclusion and Future Work

We have devised a computable version of Haar's Theorem: proven once using the elegant synthetic (implicit) approach and once developing and analyzing an explicit, imperative algorithm. And we have established the computational complexity of the Haar integral to characterize $\#P_1$ for each of the compact groups $U(1), U(2), O(2), O(3), SU(2), SO(3)$. Moreover, we implemented the algorithm for $SU(2)$ in *Exact Real Computation* [3] and confirmed that the experiment coincides with the complexity theorem. In fact, our proof shows them mutually second-order polynomial-time Weihrauch reducible [12].

Future work will generalize the above complexity considerations to $SO(4)$, to $SO(d)$, and to further classes of compact metric groups; and improve the implementation to achieve practical performance.

On the abstract side of our work, an immediate question is whether we can generalize from compact groups to locally compact groups (as was done for the classical Haar's theorem).


The price to pay for this generalization in the classic setting is that we no longer obtain a unique probability measure, but merely a locally finite measure identified up to a constant scaling factor. A notion of effective local compactness is available (see [23]), but any such generalization seems to require new proof techniques beyond those employed in this article. Recently, Davorin Lešnik has shown that this one can be done in synthetic topology, provided that one is willing to relax the requirement that measures take values in the lower reals to values in the Borel reals [17].

References

- 1 Vasco Brattka, Peter Hertling, and Klaus Weihrauch. A Tutorial on Computable Analysis. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *New Computational Paradigms: Changing Conceptions of What is Computable*, pages 425–491. Springer, New York, 2008. doi:10.1007/978-0-387-68546-5_18.
- 2 Vasco Brattka, Joseph Miller, Stéphane Le Roux, and Arno Pauly. Connected Choice and Brouwer’s Fixed Point Theorem. *Journal for Mathematical Logic*, 2019. doi:10.1142/S0219061319500041.
- 3 Franz Brauße, Pieter Collins, Johannes Kanig, SunYoung Kim, Michal Konecny, Gyesik Lee, Norbert Müller, Eike Neumann, Sewon Park, Norbert Preining, and Martin Ziegler. Semantics, Logic, and Verification of "Exact Real Computation". *CoRR*, abs/1608.05787, 2016. arXiv:1608.05787.
- 4 Pieter Collins. Computable Stochastic Processes. *CoRR*, abs/1409.4667, 2014. arXiv:1409.4667.
- 5 Abbas Edalat. A computable approach to measure and integration theory. *Information and Computation*, 207(5):642–659, 2009. doi:10.1016/j.ic.2008.05.003.
- 6 Martín Escardó. Synthetic Topology of Data Types and Classical Spaces. *Electronic Notes in Theoretical Computer Science*, 87:21–165, 2004. doi:10.1016/j.entcs.2004.09.017.
- 7 Martín Escardó. Algorithmic solution of higher type equations. *J. Logic Comput.*, 23(4):839–854, 2013. doi:10.1093/logcom/exr048.
- 8 Hugo Férée and Martin Ziegler. On the Computational Complexity of Positive Linear Functionals on $C[0;1]$. In *Mathematical Aspects of Computer and Information Sciences - 6th International Conference, MACIS 2015, Berlin, Germany, November 11-13, 2015, Revised Selected Papers*, pages 489–504, 2015. doi:10.1007/978-3-319-32859-1_42.
- 9 Stefano Galatolo, Mathieu Hoyrup, and Cristobal Rojas. Dynamics and abstract computability: computing invariant measures. *Discrete Contin. Dyn. Syst.*, 29(1):193–212, 2011. doi:10.3934/dcds.2011.29.193.
- 10 Xiaolin Ge and Anil Nerode. On extreme points of convex compact Turing located set. In Anil Nerode and Yu. V. Matiyasevich, editors, *Logical Foundations of Computer Science*, pages 114–128, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.
- 11 Paul R. Halmos. *Measure Theory*, volume 018 of *GTM*. Springer, 1950. doi:10.1007/978-1-4684-9440-2.
- 12 Akitoshi Kawamura and Stephen A. Cook. Complexity Theory for Operators in Analysis. *TOCT*, 4(2):5:1–5:24, 2012. doi:10.1145/2189778.2189780.
- 13 Akitoshi Kawamura, Florian Steinberg, and Martin Ziegler. Complexity Theory of (Functions on) Compact Metric Spaces. In *Proc. 31st Ann. Symposium on Logic in Computer Science, LICS*, pages 837–846. ACM, 2016. doi:10.1145/2933575.2935311.
- 14 Ker-I Ko. *Complexity Theory of Real Functions*. Progress in Theoretical Computer Science. Birkhäuser, Boston, 1991. doi:10.1007/978-1-4684-6802-1.
- 15 Ulrich Kohlenbach. *Applied Proof Theory: Proof Interpretations and their Use in Mathematics*. Springer, Berlin, 2008. doi:10.1007/978-3-540-77533-1.
- 16 A.N. Kolmogorov and V.M. Tikhomirov. \mathcal{E} -Entropy and \mathcal{E} -Capacity of Sets in Functional Spaces. *Uspekhi Mat. Nauk*, 14(2):3–86, 1959. doi:10.1007/978-94-017-2973-4_7.

- 17 Davorin Lešnik. Haar Measure in Synthetic Topology. presentation at CCC 2019, 2019.
- 18 Norbert Th. Müller. The iRRAM: Exact Arithmetic in C++. In Jens Blanck, Vasco Brattka, and Peter Hertling, editors, *Computability and Complexity in Analysis*, volume 2064 of *Lecture Notes in Computer Science*, pages 222–252, Berlin, 2001. Springer. 4th International Workshop, CCA 2000, Swansea, UK, September 2000. doi:10.1007/3-540-45335-0_14.
- 19 Vladimir P. Orevkov. A constructive mapping of the square onto itself displacing every constructive point (Russian). *Doklady Akademii Nauk*, 152:55–58, 1963. translated in: Soviet Math. - Dokl., 4 (1963) 1253–1256.
- 20 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
- 21 Arno Pauly. On the topological aspects of the theory of represented spaces. *Computability*, 5(2):159–180, 2016. doi:10.3233/COM-150049.
- 22 Arno Pauly. Bi-invariant metrics on compact Polish group. MathOverflow, February 2019. URL: <https://mathoverflow.net/q/322668>.
- 23 Arno Pauly. Effective local compactness and the hyperspace of located sets. *CoRR*, abs/1903.05490, 2019. arXiv:1903.05490.
- 24 Arno Pauly and Willem Fouché. How constructive is constructing measures? *Journal of Logic & Analysis*, 9, 2017. doi:10.4115/jla.2017.9.c3.
- 25 Arno Pauly and Hideki Tsuiki. T^ω -representations of compact sets. *CoRR*, abs/1604.00258, 2016. arXiv:1604.00258.
- 26 Florian Pausinger. Uniformly distributed sequences in the orthogonal group and on the Grassmannian manifold. *Mathematics and Computers in Simulation*, 160:13–22, 2019. doi:10.1016/j.matcom.2018.11.022.
- 27 Marian B. Pour-El and J. Ian Richards. A computable ordinary differential equation which possesses no computable solution. *Annals Math. Logic*, 17:61–90, 1979. doi:10.1016/0003-4843(79)90021-4.
- 28 William H Press. *Numerical recipes: The art of scientific computing*. Cambridge university press, 2007.
- 29 Matthias Schröder. Admissible Representations in Computable Analysis. In A. Beckmann, U. Berger, B. Löwe, and J.V. Tucker, editors, *Logical Approaches to Computational Barriers*, volume 3988 of *Lecture Notes in Computer Science*, pages 471–480, Berlin, 2006. Springer. Second Conference on Computability in Europe, CiE 2006, Swansea, UK, June 30-July 5, 2006. doi:10.1007/11780342_48.
- 30 Matthias Schröder and Alex Simpson. Representing probability measures using probabilistic processes. *Journal of Complexity*, 22(6):768–782, 2006. doi:10.1016/j.jco.2006.05.003.
- 31 Ernst Specker. Nicht konstruktiv beweisbare Sätze der Analysis. *The Journal of Symbolic Logic*, 14(3):145–158, 1949. doi:10.2307/2268459.
- 32 Ernst Specker. Der Satz vom Maximum in der rekursiven Analysis. In A. Heyting, editor, *Constructivity in mathematics*, Studies in Logic and the Foundations of Mathematics, pages 254–265, Amsterdam, 1959. North-Holland. Proc. Colloq., Amsterdam, Aug. 26–31, 1957. doi:10.1007/978-3-0348-9259-9_12.
- 33 Florian Steinberg. Complexity theory for spaces of integrable functions. *Logical Methods in Computer Science*, Volume 13, Issue 3, September 2017. doi:10.23638/LMCS-13(3:21)2017.
- 34 Klaus Weihrauch. Computability on computable metric spaces. *Theoretical Computer Science*, 113:191–210, 1993. Fundamental Study. doi:10.1016/0304-3975(93)90001-A.
- 35 Klaus Weihrauch. *Computable Analysis*. Springer, Berlin, 2000. doi:10.1007/978-3-642-56999-9.
- 36 Klaus Weihrauch. Computational complexity on computable metric spaces. *Mathematical Logic Quarterly*, 49(1):3–21, 2003. doi:10.1002/ma1q.200310001.

The Call-By-Value Lambda-Calculus with Generalized Applications

José Espírito Santo 

Centre of Mathematics, University of Minho, Portugal
jes@math.uminho.pt

Abstract

The lambda-calculus with generalized applications is the Curry-Howard counterpart to the system of natural deduction with generalized elimination rules for intuitionistic implicational logic. In this paper we identify a call-by-value variant of the system and prove confluence, strong normalization, and standardization. In the end, we show that the cbn and cbv variants of the system simulate each other via mappings based on extensions of the “protecting-by-a-lambda” compilation technique.

2012 ACM Subject Classification Theory of computation → Proof theory; Theory of computation → Lambda calculus

Keywords and phrases Generalized applications, Natural deduction, Strong normalization, Standardization, Call-by-name, Call-by-value, Protecting-by-a-lambda

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.35

Funding The author was supported by Fundação para a Ciência e a Tecnologia (FCT) through project UID/MAT/00013/2013.

1 Introduction

The λ -calculus with generalized applications [8], named system ΛJ , or λJ -calculus, corresponds, under the Curry-Howard isomorphism, to the system of natural deduction with generalized elimination rules [10, 16], in the setting of intuitionistic implicational logic. As a variant of the λ -calculus, ΛJ can be qualified naively as being a call-by-name (cbn) system, simply because its β -rule prescribes that, in functional application, functions are called without prior evaluation of arguments. In this paper we propose a call-by-value (cbv) variant of system ΛJ and prove some of its properties. With this, we develop the cbv side of natural deduction.

The novel system is named system ΛJ_v , or the λJ_v -calculus. Notably, the syntax of proof terms remains the same as that of system ΛJ - and our purpose is to define cbv reduction rules appropriate for this syntax. Moreover, the reduction rules of ΛJ_v will look like those of ΛJ : there is a β -rule (corresponding to a “detour” conversion rule) and a π (corresponding to a commuting conversion rule), the latter in fact unchanged w.r.t. ΛJ ; the notion of β -redex is also unchanged; the only thing that will change is the concept of substitution.

Plotkin [11] sets a criterion for a calculus to be qualified as call-by-value: it should enjoy a standardization theorem, and the notion of standard reduction sequence should be based on a notion of call-by-value evaluation. We will prove a standardization theorem for ΛJ_v that makes an explicit link to a notion of cbv evaluation. We also prove the main rewriting-theoretic properties: confluence and strong normalization.

Plotkin [11] shows that cbn and cbv calculi based on the syntax of ordinary λ -terms simulate each other via cps translations. The need to resort to cps translations is justified in [11] by the fact that the “protecting-by-a-lambda” compilation technique (which easily gives a simulation of cbn by cbv) does not extend to a simulation in the opposite direction. Here we show that, when the syntax allows generalized applications, cps translations are not needed to obtain simulation in both ways, as both simulations can be based on “protecting-by-a-lambda”.



© José Espírito Santo;

licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 35; pp. 35:1–35:12

Leibniz International Proceedings in Informatics



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

Plan of the paper. Section 2 reviews Λ , Plotkin's Λ_v , and Λ_J . Section 3 introduces Λ_{J_v} . Sections 4 and 5 prove strong normalization and confluence, respectively. Section 6 proves standardization. Section 7 simulates the cbn and cbv variants into each other. Section 8 summarizes our contributions and concludes.

2 Background

System Λ . The ordinary λ -calculus is denoted Λ . The λ -terms are given by:

$$(\text{Terms}) \ M, N, P, Q ::= V \mid MN \quad (\text{Values}) \ V, W ::= x \mid \lambda x.M$$

We work modulo α -equivalence and assume silent renaming of bound variables as needed. We write $x \notin M$ to say that x does not occur free in M . Substitution is denoted $[N/x]M$.

Λ is equipped with the β -rule $(\lambda x.M)N \rightarrow [N/x]M$, which generates the relation \rightarrow_β - the compatible closure of β . We use the common notations \rightarrow_β^- , \rightarrow_β^+ , \rightarrow_β^* , and $=_\beta$ for, respectively, the reflexive, transitive, reflexive-transitive, and equivalence closures of \rightarrow_β (and similarly for any other reduction rule in any other calculus in this paper).

An important role is played by λ -terms with *holes*. A hole, denoted $[\cdot]$, is a special place-holder that can be *filled* with a λ -term. We will only consider terms with a single hole, and call them *contexts*. If \mathcal{C} is a context, the $\mathcal{C}[M]$ denotes the λ -term resulting from filling the single hole of \mathcal{C} with M . Notice contexts allow an alternative definition of \rightarrow_β : $P \rightarrow_\beta Q$ iff $P = \mathcal{C}[(\lambda x.M)N]$ and $Q = \mathcal{C}[[N/x]M]$, for some \mathcal{C} .

We consider the familiar, Curry-style typing system, for assigning simple types to λ -terms. Types, ranged over by A, B, C , etc. are formulas of implicational logic, which can either be an atom p or an implication $A \supset B$. A sequent is an expression $\Gamma \vdash M : A$, where Γ and M are called the *base* and the *subject* of the sequent, respectively. A base Γ is a set of assignments $x : A$ of types to variables, so that no variable is assigned two different types. The familiar typing rules, which we refrain to repeat, determine the *derivable* sequents, and define a natural deduction system for intuitionistic implicational logic. A λ -term M is *typable* if there is a derivable sequent with M as subject.

System Λ_v . Plotkin's cbv λ -calculus [11] is here named Λ_v . The terms of Λ_v are the same λ -terms of Λ , but now the system is equipped with a variant of the β -rule, named β_v : $(\lambda x.M)V \rightarrow [V/x]M$. Here, for the function $\lambda x.M$ to be called, the argument is required to be a value. Again, the compatible closure \rightarrow_{β_v} of β_v may be defined by $\mathcal{C}[(\lambda x.M)V] \rightarrow_{\beta_v} \mathcal{C}[[V/x]M]$. For the typed version of the system, we employ the same typing system as for Λ .

Several authors [13, 6] have proposed extra reduction principles for Λ_v . Two of them will be central in the present paper:

$$\begin{array}{ll} (\rho_1) & (\lambda x.M)NQ \rightarrow (\lambda x.MQ)N \\ (\rho_2) & (\lambda y.P)((\lambda x.M)N) \rightarrow (\lambda x.(\lambda y.P)M)N \end{array}$$

The first is one of Regnier's σ -rules [12]. The author has studied these rules in [3, 4], allowing the second one in a more general form: $P((\lambda x.M)N) \rightarrow (\lambda x.PM)N$. A common idea to ρ_1 and ρ_2 is to rearrange the term to reveal (potential) redexes: MQ in the former case, $(\lambda y.P)M$ in the latter case¹. We let $\rho := \rho_1 \cup \rho_2$.

¹ The name ρ intends to be mnemonic of this action of rearranging to reveal redexes. We find ρ preferable to σ for two reasons: first, σ is a name one wishes to reserve to substitution rules; second, ρ_2 is not one

System ΛJ . The λ -calculus with generalized applications [8] is here named system ΛJ , or the λJ -calculus. The λJ -terms are given by:

$$(\text{Terms}) \ M, N, P, Q ::= V \mid M(N, x.P) \quad (\text{Values}) \ V, W ::= x \mid \lambda x.M$$

The constructor $M(N, x.P)$ is called *generalized application*; in it, “ x .” is a binder of x , so x is bound in P . Consistent with the generalized application terminology is to call $(N, x.P)$ the *generalized argument* of that application.

ΛJ is equipped with two rules:

$$\begin{aligned} (\beta) \quad & (\lambda x.M)(N, y.P) \rightarrow [[N/x]M/y]P \\ (\pi) \quad & M(N, x.P)(N', y.P') \rightarrow M(N, x.P(N', y.P')) \end{aligned}$$

The version of π adopted is the same as in [8]. One could have considered an “eager” version, with *contractum* $M(N, x.P@(N', y.P'))$, where operator $@$ is defined by

$$\begin{aligned} V@(N', y.P') &= V(N', y.P') \\ M(N, x.P)@(N', y.P') &= M(N, x.P@(N', y.P')) \end{aligned}$$

In this version of the rule, the generalized argument $(N', y.P')$ is eagerly pushed in, until a value is found. Operator $@$ will be used again in Section 5.

The typing system of ΛJ is obtained from the one for Λ by adopting the following rule for typing generalized applications:

$$\frac{\Gamma \vdash M : A \supset B \quad \Gamma \vdash N : A \quad \Gamma, x : B \vdash P : C}{\Gamma \vdash M(N, x.P) : C} \text{GE } \supset$$

Such typing system defines a system of natural deduction with generalized elimination rules [16]. Rule π is a “commutative conversion” caused by the repetition of formula C in $GE \supset$.

3 The call-by-value variant

This section is dedicated to the call-by-value variant of ΛJ we introduce, named ΛJ_v . We first motivate the system. In the second part of the section, we define the system.

3.1 Motivation

Consider the β -redex $(\lambda x.M)N$. We will define a new *contractum* for this redex, making use of a syntax where application is generalized. The *contractum* is again a substitution, but one whose definition will express call-by-value - let us denote it by $[N \setminus x]M$.

If $N = V$, no doubt we want to substitute ordinarily, as in Plotkin’s Λ_v , so we put

$$[V \setminus x]M = [V/x]M \tag{1}$$

But if $N = M'N'$, we want to postpone the call of $\lambda x.M$ and evaluate N first. Making use of generalized application, we rewrite the original β -redex as $M'(N', x.M)$. Notice $M'N'$ is actually $M'(N', z.z)$, so we want

$$[M'(N', z.z) \setminus x]M = M'(N', x.M) \tag{2}$$

of Regnier’s σ -rules. In [3, 4] these rules were called π_1 and π_2 . There is some point in that choice of names, even knowing that π is the name used in [8] for one of the reduction rules of ΛJ - a convention we will maintain here. Still, we found it preferable to abandon π_1 and π_2 and give the rules a fresh and useful name for the present paper.

How to complete the definition of $[N \setminus x]M$, for the case $N = M'(N', z.P')$? Just put

$$[M'(N', z.P') \setminus x]M = M'(N', z.[P' \setminus x]M) \quad (3)$$

Then we get (2) from (3) and (1).

Finally, how about starting off with $(\lambda x.M)(N, y.P)$? As in ΛJ , the *contractum* consists of two substitutions, but both of the new kind we have just defined.

3.2 System ΛJ_v

We now define system ΛJ_v , or λJ_v -calculus. The λJ_v -terms are the same as the λJ -terms. Ordinary substitution will be used only in the form $[V/x]M$, with the actual parameter a value V . We introduce **left substitution**, denoted $[N \setminus x]M$, defined by recursion on N as follows:

$$\begin{aligned} [V \setminus x]M &= [V/x]M \\ [N(Q, y.P) \setminus x]M &= N(Q, y.[P \setminus x]M) \end{aligned}$$

ΛJ_v has two reduction rules:

$$\begin{aligned} (\beta_v) \quad & (\lambda x.M)(N, y.P) \rightarrow [[N \setminus x]M \setminus y]P \\ (\pi) \quad & M(N, x.P)(N', y.P') \rightarrow M(N, x.P(N', y.P')) \end{aligned}$$

Rule π is the same as ΛJ , rule β_v is new. However, there is a formal similarity with rule β from ΛJ - the only difference is in the substitution operator employed.

The typing system for ΛJ_v is the same as the typing system for ΛJ , with left substitution being typed by the same admissible rule that types ordinary substitution. A λJ_v -term is typable if it is the subject of some derivable sequent.

A routine result is subject reduction, already known for \rightarrow_π , and which also extends to \rightarrow_{β_v} . Perhaps more important is to recast rule β_v as a proof normalization rule in natural deduction with generalized elimination rules - this is done in Fig. 1.

We finish this section with some technical lemmas.

► **Lemma 1** (Substitution lemma). *In ΛJ_v :*

1. $[V/x][N \setminus y]M = [[V/x]N \setminus y][V/x]M$, provided $y \notin V$.
2. $[N \setminus x][N' \setminus y]M = [[N \setminus x]N' \setminus y]M$, provided $x \notin M, y \notin N$.

Proof. Routine. ◀

► **Lemma 2** (Parallelism). *The following rules are admissible:*

$$\frac{V \rightarrow_\pi^* V' \quad M \rightarrow_\pi^* M'}{[V/x]M \rightarrow_\pi^* [V'/x]M'} \quad (i) \quad \frac{M \rightarrow_\pi^* M'}{[N \setminus x]M \rightarrow_\pi^* [N \setminus x]M'} \quad (ii) \quad \frac{N \rightarrow_\pi^* N' \quad M \rightarrow_\pi^* M'}{[N \setminus x]M \rightarrow_\pi^* [N' \setminus x]M'} \quad (iii)$$

Proof. (i) Known. (ii) Proved by induction on N and uses (i). (iii) Follows easily from the version of (iii) where the first premiss is $N \rightarrow_\pi N'$, and the latter is proved by induction on $N \rightarrow_\pi N'$, using (i) and (ii). ◀

4 Strong normalization

We define a map from λJ -terms to λ -terms

$$x^\# = x \quad (\lambda x.M)^\# = \lambda x.M^\# \quad M(N, x.P)^\# = (\lambda x.P^\#)(M^\#N^\#)$$

This map is to be promoted to a homomorphism $\Lambda J_v \rightarrow \Lambda_v$. First, the technical lemma:

Proof. Both items by induction on $M_1 \rightarrow M_2$. The base case of the first item uses Lemma 3 twice. The base case of the second item is proved by a direct calculation. The inductive cases are routine. ◀

► **Theorem 5 (SN).** *If $M \in \Lambda J_v$ is typable, then M is $\beta_v\pi$ -SN.*

Proof. Suppose M is typable. It is easy to see that M^\sharp has to be typable. By strong normalization for Λ , M^\sharp is β -SN. By the main result in [4], M^\sharp is $\beta\rho$ -SN, and so is $\beta_v\rho$ -SN. Given the strict simulation obtained in Proposition 4 ($M_1 \rightarrow_{\beta_v\pi} M_2$ implies $M_1^\sharp \rightarrow_{\beta_v\rho}^+ M_2^\sharp$), we conclude that M is $\beta_v\pi$ -SN. ◀

5 Confluence

In this section we prove that $\rightarrow_{\beta_v\pi}$ is confluent. We follow the approach in [8], pointing out where the differences are. In this section, we abbreviate $\rightarrow_{\beta_v\pi}$ as \rightarrow . So \rightarrow^* denotes $\rightarrow_{\beta_v\pi}^*$.

Given a binary relation \rightsquigarrow on ΛJ -terms and a function f from ΛJ -terms to ΛJ -terms, we say \rightsquigarrow and f satisfy the **triangle property** [15, 8] if $M \rightsquigarrow M'$ implies $M' \rightsquigarrow f(M)$. Hence, every 1-step \rightsquigarrow -reduct of M does one \rightsquigarrow -step to a common term that depends on M solely; and therefore, \rightsquigarrow satisfies the diamond property.

Given a binary relation \rightsquigarrow on ΛJ -terms, we say \rightsquigarrow is a **$\beta_v\pi$ -development**, or just a **development**, if $\rightarrow \subseteq \rightsquigarrow \subseteq \rightarrow^*$ and \rightsquigarrow and f satisfy the triangle property, for some f (which is then called a **complete $\beta_v\pi$ -development**, or just a **development**). Notice that, if there is a development \rightsquigarrow , then \rightarrow is confluent. Proof: \rightsquigarrow satisfies the diamond property, hence so does \rightarrow^* . But $\rightsquigarrow^* = \rightarrow^*$ (because $\rightarrow \subseteq \rightsquigarrow \subseteq \rightarrow^*$). Therefore \rightarrow^* satisfies the diamond property, that is, \rightarrow is confluent.

For M a ΛJ -term, the π -normal form M^π is defined² by recursion on M as follows:

$$\begin{aligned} x^\pi &= x \\ (\lambda x.M)^\pi &= \lambda x.M^\pi \\ (M(N, x.P))^\pi &= M^\pi @ (N^\pi, x.P^\pi) \end{aligned}$$

► **Lemma 6.** \rightarrow_π^* and $(_)^\pi$ satisfy the triangle property.

Proof. In [8]. ◀

Given that $\rightarrow_\pi \subseteq \rightarrow_\pi^* = \rightarrow_\pi^*$, we may call $(_)^\pi$ a complete π -development.

Next we introduce a new pair that will satisfy the triangle property, containing a new binary relation and a complete β_v -development. The binary relation \Rightarrow_v on ΛJ is defined in Fig. 2. It is immediate to show that: (i) \Rightarrow_v is reflexive; (ii) $\rightarrow_{\beta_v} \subseteq \Rightarrow_v$; (iii) $\Rightarrow_v \subseteq \rightarrow_{\beta_v}^*$.

► **Lemma 7 (Parallelism).** *The following rules are admissible:*

$$\frac{V \Rightarrow_v V' \quad M \Rightarrow_v M'}{[V/x]M \Rightarrow_v [V'/x]M'} \quad (i) \qquad \frac{N \Rightarrow_v N' \quad M \Rightarrow_v M'}{[N \setminus x]M \Rightarrow_v [N' \setminus x]M'} \quad (ii)$$

Proof. The proof of (i) is by induction on $M \Rightarrow_v M'$. Case BETA uses item 1 of Lemma 1. The proof of (ii) is by induction on $N \Rightarrow_v N'$. Cases VAR and ABS use (i). Case BETA uses item 2 of Lemma 1. ◀

² The operator @ we are employing in this paper is slightly different from the one employed in [8], but the function M^π is the same.

$$\begin{array}{c}
\frac{}{x \Rightarrow_v x} \text{VAR} \quad \frac{M \Rightarrow_v M'}{\lambda x.M \Rightarrow_v \lambda x.M'} \text{ABS} \quad \frac{M \Rightarrow_v M' \quad N \Rightarrow_v N' \quad P \Rightarrow_v P'}{M(N, x.P) \Rightarrow_v M'(N', x.P')} \text{APL} \\
\\
\frac{M \Rightarrow_v M' \quad N \Rightarrow_v N' \quad P \Rightarrow_v P'}{(\lambda x.M)(N, y.P) \Rightarrow_v [[N' \setminus x]M' \setminus y]P'} \text{BETA}
\end{array}$$

■ **Figure 2** β_v -development.

For M a ΛJ -term, define M^{β_v} by recursion on M as follows:

$$\begin{array}{l}
x^{\beta_v} = x \\
(\lambda x.M)^{\beta_v} = \lambda x.M^{\beta_v} \\
(M(N, y.P))^{\beta_v} = \begin{cases} [[N^{\beta_v} \setminus x]M_0^{\beta_v} \setminus y]P^{\beta_v} & \text{if } M = \lambda x.M_0 \\ M^{\beta_v}(N^{\beta_v}, y.P^{\beta_v}) & \text{otherwise} \end{cases}
\end{array}$$

► **Lemma 8.** \Rightarrow_v and $(_)^\pi$ satisfy the triangle property.

Proof. Once we have the parallelism property of \Rightarrow_v w.r.t. left substitution (item (ii) of Lemma 7), we can repeat the proof in [8]. ◀

Given that $\rightarrow_{\beta_v} \subseteq \Rightarrow_v \rightarrow_{\beta_v}^*$, we may call $(_)^\beta$ a complete β_v -development.

► **Lemma 9 (Commutation).** *If $M \Rightarrow_v N_1$ and $M \rightarrow_{\pi}^* N_2$ then there is P such that $N_1 \rightarrow_{\pi}^* P$ and $N_2 \Rightarrow_v P$.*

Proof. We can repeat the proof in [8], since \rightarrow_{π}^* is also parallel in the sense of Lemma 2. ◀

► **Theorem 10.** $\rightarrow_{\beta_v \pi}$ is confluent.

Proof. The two triangle properties (Lemmas 6 and 8) and the commutation property (Lemma 9) imply that the relation $\rightarrow_{\pi}^* \circ \Rightarrow_v$ and the function $(_)^\pi \circ (_)^\beta$ have the triangle property (this composition of triangle properties is easily proved - see [8]). Moreover, it is obvious that $\rightarrow_{\beta_v \pi} \subseteq \rightarrow_{\pi}^* \circ \Rightarrow_v \subseteq \rightarrow_{\beta_v \pi}^*$. This means that $\rightarrow_{\pi}^* \circ \Rightarrow_v$ is a development (and that $(_)^\pi \circ (_)^\beta$ is a complete development). Hence \rightarrow is confluent. ◀

6 Standardization

In this section we prove the mandatory [11] standardization theorem for ΛJ_v . Contrary to many proofs [11, 6], our proof does not handle directly standard reduction sequences - in this we follow [8]. On the other hand, we do not rely on vector notation either; instead we make explicit the contribution of call-by-value evaluation to the standard reduction relation. The definition of cbv evaluation is interesting on its own.

The definition of cbv evaluation is in terms of **cbv evaluation contexts**:

$$\mathcal{E} ::= [\cdot] \mid \mathcal{E}(N, x.P)$$

Then **call-by-value evaluation**, denoted \mapsto_v , is defined as $\mathcal{E}[M] \mapsto_v \mathcal{E}[M']$ where $(M, M') \in \beta_v \cup \pi$ (in other words, $M \rightarrow M'$ is a root $\beta_v \pi$ -reduction). The familiar, alternative and equivalent definition is to say that \mapsto_v is inductively defined by $\beta_v \cup \pi$ and the closure rule: $M \rightarrow M'$ implies $M(N, x.P) \rightarrow M'(N, x.P)$.

$$\begin{array}{c}
\overline{x \Rightarrow x} \text{ VAR} \quad \frac{M \Rightarrow M'}{\lambda x.M \Rightarrow \lambda x.M'} \text{ ABS} \quad \frac{M \Rightarrow M' \quad N \Rightarrow N' \quad P \Rightarrow P'}{M(N, x.P) \Rightarrow M'(N', x.P')} \text{ APL} \\
\\
\frac{M \mapsto_v^* \lambda x.M' \quad [[N \setminus x]M' \setminus y]P \Rightarrow Q}{M(N, y.P) \Rightarrow Q} \text{ BETA} \\
\\
\frac{M \mapsto_v^* M'(N', x.P') \quad M'(N', x.(P'(N, y.P))) \Rightarrow Q}{M(N, y.P) \Rightarrow Q} \text{ PI}
\end{array}$$

■ **Figure 3** Standard reduction.

So we employ the single closure we would employ if we were defining call-by-name evaluation - the whole difference is in the β -rule. Notice that, due to rule π , evaluation is not a deterministic (univocal) relation.

Standard reduction, denoted $M \Rightarrow M'$, is defined in Fig. 3.

A simple induction shows that $\Rightarrow \subseteq \rightarrow_{\beta_v \pi}^*$. Despite its simplicity, this remark is important because its proof builds a $\beta_v \pi$ -reduction sequence from M to M' , given that $M \Rightarrow M'$. We refer to reduction sequences thus built as **standard reduction sequences**. If $M \Rightarrow M'$ is proved by *VAR*, *ABS* or *APL*, the outer constructor of M is frozen forever. For instance, if $M = \lambda x.M_0$, then $M' = \lambda x.M'_0$ and the reduction sequence given by induction hypothesis will have all of its members prefixed by λx . On the other hand, if $M \Rightarrow M'$ is proved by *BETA*, we prefix the reduction sequence (not its members) given by induction hypothesis by a sequence of cbv evaluation steps, namely the ones leading from $M(N, y.P)$ to $(\lambda x.M')(N, y.P)$ followed by the step $(\lambda x.M')(N, y.P) \mapsto_v [[N \setminus x]M' \setminus y]P$ implicit in rule *BETA*. Similar remarks apply to rule *PI*. The general description of a standard reduction sequence is this: it contains an initial segment performing cbv evaluation, until one decides to freeze the outer constructor of the last term obtained, and the standard reduction sequence proceeds by reducing in parallel the immediate sub-terms.

The converse of $\Rightarrow \subseteq \rightarrow_{\beta_v \pi}^*$ is a kind of completeness of \Rightarrow .

► **Theorem 11** (Standardization). $\rightarrow_{\beta_v \pi}^* \subseteq \Rightarrow$.

Proof. We show successively the closure rules I to VII in Fig. 4. The theorem follows from rule VII (together with rule I). We also need substitutivity of evaluation: If $M \mapsto_v^* M'$ then $[V/x]M \mapsto_v^* [V/x]M'$. This is an easy consequence of: If $M \mapsto_v M'$ then $[V/x]M \mapsto_v [V/x]M'$. The latter is proved by induction on $M \mapsto_v M'$.

Rule I is proved by an easy induction on M . Rule II is proved by induction on $M \mapsto_v M'$. Rule III is an easy consequence of II. Rule IV is proved by induction on $M \Rightarrow M'$ (the case relative to *BETA* requires substitutivity of evaluation). Rule V is proved by induction on $N \Rightarrow N'$ and uses rule IV. Rule VII is an easy consequence of Rule VI. Rule VI is the rule with most delicate proof. It is proved by induction on $M \Rightarrow M'$. The case relative to *APP* splits into several cases determined by a reduction step of the form $M'(N', y.P') \rightarrow_{\beta_v \pi} Q$. Two of them are proved with a sub-induction and use rules III and V. ◀

7 Call-by-name and call-by-value

In this section, we show simulations between ΛJ and ΛJ_v . For emphasis, we denote ΛJ by ΛJ_n , and β by β_n . Both simulation employ the “protecting-by-a-lambda” technique;

$$\begin{array}{c}
\overline{M \Rightarrow M} \quad I \\
\\
\frac{M \mapsto_v M' \Rightarrow Q}{M \Rightarrow Q} \quad II \qquad \frac{M \mapsto_v^* M' \Rightarrow Q}{M \Rightarrow Q} \quad III \\
\\
\frac{V \Rightarrow V' \quad M \Rightarrow M'}{[V/x]M \Rightarrow [V'/x]M'} \quad IV \qquad \frac{N \Rightarrow N' \quad M \Rightarrow M'}{[N \setminus x]M \Rightarrow [N' \setminus x]M'} \quad V \\
\\
\frac{M \Rightarrow M' \rightarrow_{\beta_v \pi} Q}{M \Rightarrow Q} \quad VI \qquad \frac{M \Rightarrow M' \rightarrow_{\beta_v \pi}^* Q}{M \Rightarrow Q} \quad VII
\end{array}$$

■ **Figure 4** Additional closure rules.

and, in their typed version, the simulations employ the same type translation, namely the replacement of A by $\top \supset A$ at appropriate places. So, we need neither cps-translations, nor type translations based on the insertion of double negations, in order to translate between the cbn and cbv variants of ΛJ .

We will use I to denote the combinator $\lambda x.x$; and MI will abbreviate $M(I, x.x)$. Also, $\lambda d.M$ will stand for a vacuous abstraction (d is a “dummy” variable, *i.e.* $d \notin M$). We fix some type variable X and put $\top := X \supset X$.

7.1 Simulation of cbn by cbv

For $M \in \Lambda J_n$, M° is defined by recursion on M as follows:

$$\begin{array}{l}
x^\circ = xI \\
(\lambda x.M)^\circ = \lambda x \lambda d.M^\circ \\
(M(N, y.P))^\circ = M^\circ(\lambda d.N^\circ, y.P^\circ)
\end{array}$$

This compilation is a variation on the “protecting-by-a-lambda” technique [11], now extended to deal with generalized applications. When translating these, the argument is protected by a vacuous abstraction, to pretend to be a value. Accordingly, variables are applied to a dummy argument. The generality of generalized applications commutes with the translation, but some novelty is observed in the translation of abstractions, where an unexpected, extra, vacuous abstraction shows up.

This surprise has a counterpart in the typed version of this translation, at the level of types. The type A° is defined by recursion on A by:

$$\begin{array}{l}
X^\circ = X \\
(A \supset B)^\circ = \underline{A} \supset \underline{B}
\end{array}$$

where $\underline{A} = \top \supset A^\circ$. The surprise is that $(A \supset B)^\circ$ is not defined as $\underline{A} \supset B^\circ$, as one would expect, if this was just the usual “protecting-by-a-lambda”, or “thunk-introduction” compilation (see *e.g.* [7, 5]).

As usual, $\underline{\Gamma} = \{(x : \underline{A}) \mid (x : A) \in \Gamma\}$. It is straightforward to show:

► **Proposition 12.** *If $\Gamma \vdash M : A$ then $\underline{\Gamma} \vdash M^\circ : A^\circ$.*

► **Lemma 13.** $(\lambda d.M)I \rightarrow_{\beta_v} M$.

Proof. By a simple calculation, using the fact $[M \setminus x]x = M$. ◀

► **Lemma 14.** $[\lambda d.N^\circ/x]M^\circ \rightarrow_{\beta_v}^* ([N/x]M)^\circ$.

Proof. By induction on M . The case $M = x$ uses the previous lemma, and is where the β_v -steps are generated. ◀

► **Theorem 15** (Simulation of ΛJ_n by ΛJ_v).

1. If $M \rightarrow_{\beta_n} N$ in ΛJ_n then $M^\circ \rightarrow_{\beta_v}^+ N^\circ$ in ΛJ_v .
2. If $M \rightarrow_\pi N$ in ΛJ_n then $M^\circ \rightarrow_\pi N^\circ$ in ΛJ_v .

Proof. Both items by induction on $M \rightarrow N$. The inductive cases are routine. The base case of the first item (case β_n) uses Lemma 14. The base case of the second item (case π) is a straightforward calculation. ◀

7.2 Simulation of cbv by cbn

In the 1970's [11], the need for cps-translations was justified by the fact that the compilation technique “protecting-by-a-lambda” did not extend to give a simulation of cbv by cbn. Next we show that this is not the case when cbn and cbv are given with generalized applications.

For $V, M \in \Lambda J_v$, V^\bullet and \overline{M} are defined by simultaneous recursion on V and M as follows:

$$\begin{aligned} x^\bullet &= x \\ (\lambda x.M)^\bullet &= \lambda x.\overline{M} \\ \overline{V} &= \lambda d.V^\bullet \\ \overline{M(N, y.P)} &= \overline{M(I, m.\overline{N(I, n.m(n, z.z(I, y.\overline{P}))})} \end{aligned}$$

This time, it is values which are wrapped with vacuous abstractions, and dummy arguments I are used in the translation of application to manipulate the flow of the computation; and even if the translation of applications is reminiscent of cps, the typed version confirms the type structure of this translation is still based on a top level given by $\top \supset A$, and not by a double negation.

The type A^\bullet is defined by recursion on A by:

$$\begin{aligned} X^\bullet &= X \\ (A \supset B)^\bullet &= \overline{A} \supset \overline{B} \end{aligned}$$

where $\overline{A} = \top \supset A^\bullet$. Indeed $A^\bullet = A^\circ$ and $\overline{A} = \underline{A}$.

As usual, $\Gamma^\bullet = \{(x : A^\bullet) \mid (x : A) \in \Gamma\}$. It is straightforward to show:

► **Proposition 16.**

1. If $\Gamma \vdash M : A$ then $\Gamma^\bullet \vdash \overline{M} : \overline{A}$.
2. If $\Gamma \vdash V : A$ then $\Gamma^\bullet \vdash V^\bullet : A^\bullet$.

► **Lemma 17.**

1. $[V^\bullet/x]\overline{M} = \overline{[V/x]M}$.
2. $\lambda d.[V^\bullet/x]W^\bullet = \overline{[V/x]W}$.

Proof. By simultaneous induction on M and V . ◀

In order to motivate the next two lemmas, observe that the term $I(M, x.P)$ behaves like an explicit substitution, both in ΛJ_n and ΛJ_v . In the former case, the term reduces to $[M/x]P$; in the latter, it reduces to $[M \setminus x]P$. We would like to have terms that, in ΛJ_n , reduce to $[M \setminus x]P$ and $[[N \setminus x]M \setminus y]P$.

► **Lemma 18.** $\overline{M}(I, x.\overline{P}) \rightarrow_{\beta_n\pi}^+ \overline{[M\backslash x]P}$.

Proof. By induction on M . The case $M = V$ uses Lemma 17. ◀

► **Lemma 19.** $\overline{N}(I, n.(\lambda x.\overline{M})(n, z.z(I, y.\overline{P}))) \rightarrow_{\beta_n\pi}^+ \overline{[[N\backslash x]M\backslash y]P}$.

Proof. By induction on N . The case $N = V$ uses Lemmas 17 and 18. ◀

► **Theorem 20** (Simulation of ΛJ_v by ΛJ_n).

1. If $M \rightarrow_{\beta_v} N$ in ΛJ_v then $\overline{M} \rightarrow_{\beta_n\pi}^+ \overline{N}$ in ΛJ_n .
2. If $M \rightarrow_{\pi} N$ in ΛJ_v then $\overline{M} \rightarrow_{\pi} \overline{N}$ in ΛJ_n .

Proof. Both items by induction on $M \rightarrow N$. The inductive cases are routine. The base case of the first item (case β_v) uses Lemma 19. The base case of the second item (case π) is a straightforward calculation. ◀

8 Final remarks

We identified a call-by-value variant ΛJ_v of system ΛJ , sharing the set of terms with the original, cbn variant, and only differing in the definition of substitution. We established the main rewriting-theoretic properties (strong normalization and confluence), and proved the standardization theorem in a way that makes evident the contribution of call-by-value evaluation for standard reduction. Finally, we proved that the cbn and cbv variant simulate each other, not via cps-translations, but rather via the technique of “protecting-by-a-lambda”[11], or “think-introduction”[7], which is here shown for the first time to extend to a simulation of cbv by cbn.

In [5] one sees the simulation of the cbn, ordinary λ -calculus and of Plotkin’s cbv λ -calculus into a common modal language, via modal embeddings: cps-translations are dispensed with, because use can be made of the extra facilities of the modal target. But here, no extension of the logic is required, we never leave intuitionistic implicational logic. Instead, use is made of the structural extension provided by generalized applications.

Our goal was to define the cbv variant of ΛJ and the cbv variant of natural deduction with generalized elimination rules: we believe this was not attempted before, we made a proposal and studied it. On the other hand, the study of cbv λ -calculi is an active field of research, with new calculi being proposed for decades ([11, 9, 13, 14, 2, 1, 6]). Although this was not our primary goal, we believe we made a contribution to this line of work, since ΛJ_v seems to have a singular place among the panoply of systems in the literature, for various reasons: first, it is strongly anchored in proof theory; second: it is extremely simple; third, it exploits the original syntactic idea of fusing into a single constructor (generalized application) ordinary application with let-expressions.

Finally, there may be another reason for studying ΛJ_v further: contrary to, say, Plotkin’s cbv λ -calculus, β -redexes in ΛJ_v always reduce, never get stuck. Now this may turn the system suitable for “open” call-by-value [1] - but that remains future work.

References

- 1 B. Accattoli and G. Guerrieri. Open Call-by-Value. In *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, volume 10017 of *Lecture Notes in Computer Science*, pages 206–226, 2016.
- 2 R. Dyckhoff and S. Lengrand. Call-by-value lambda calculus and LJQ. *Journal of Logic and Computation*, 17:1109–1134, 2007.

- 3 J. Espírito Santo. Delayed substitutions. In Franz Baader, editor, *Proceedings of RTA'07*, volume 4533 of *Lecture Notes in Computer Science*, pages 169–183. Springer-Verlag, 2007.
- 4 J. Espírito Santo. A note on preservation of strong normalisation in the λ -calculus. *Theoretical Computer Science*, 412(11):1027–1032, 2011.
- 5 J. Espírito Santo, L. Pinto, and T. Uustalu. Modal Embeddings and Calling Paradigms. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany.*, volume 131 of *LIPICs*, pages 18:1–18:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019.
- 6 G. Guerrieri, L. Paolini, and S. Ronchi Della Rocca. Standardization and Conservativity of a Refined Call-by-Value lambda-Calculus. *Logical Methods in Computer Science*, 13(4), 2017.
- 7 J. Hatcliff and O. Danvy. Thunks and the λ -Calculus (Extended Version). Technical Report BRICS RS-97-7, DIKU, 1997.
- 8 F. Joachimski and R. Matthes. Standardization and Confluence for a Lambda Calculus with Generalized Applications. In *Proceedings of RTA 2000*, volume 1833 of *LNCS*, pages 141–155. Springer, 2000.
- 9 E. Moggi. Computational lambda-calculus and monads. Technical Report ECS-LFCS-88-86, University of Edinburgh, 1988.
- 10 S. Negri and J. von Plato. *Structural Proof Theory*. Cambridge University Press, 2001.
- 11 G. Plotkin. Call-by-name, call-by-value and the λ -calculus. *Theoretical Computer Science*, 1:125–159, 1975.
- 12 L. Regnier. Une équivalence sur les lambda-termes. *Theoretical Computer Science*, 126(2):281–292, 1994.
- 13 A. Sabry and M. Felleisen. Reasoning about programmes in continuation-passing-style. *LISP and Symbolic Computation*, 6(3/4):289–360, 1993.
- 14 A. Sabry and P. Wadler. A reflection on call-by-value. *ACM Trans. on Programming Languages and Systems*, 19(6):916–941, 1997.
- 15 M. Takahashi. Parallel reductions in λ -calculus. *Information & Computation*, 118:120–127, 1995.
- 16 J. von Plato. Natural deduction with general elimination rules. *Annals of Mathematical Logic*, 40(7):541–567, 2001.

Dynamic Complexity Meets Parameterised Algorithms

Jonas Schmidt

TU Dortmund University, Dortmund, Germany

Thomas Schwentick

TU Dortmund University, Dortmund, Germany

Nils Vortmeier

TU Dortmund University, Dortmund, Germany

Thomas Zeume

TU Dortmund University, Dortmund, Germany

Ioannis Kokkinis

TU Dortmund University, Dortmund, Germany

Abstract

Dynamic Complexity studies the maintainability of queries with logical formulas in a setting where the underlying structure or database changes over time. Most often, these formulas are from first-order logic, giving rise to the dynamic complexity class DynFO. This paper investigates extensions of DynFO in the spirit of parameterised algorithms. In this setting structures come with a parameter k and the extensions allow additional “space” of size $f(k)$ (in the form of an additional structure of this size) or additional time $f(k)$ (in the form of iterations of formulas) or both. The resulting classes are compared with their non-dynamic counterparts and other classes. The main part of the paper explores the applicability of methods for parameterised algorithms to this setting through case studies for various well-known parameterised problems.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Logic and databases; Theory of computation → Complexity theory and logic

Keywords and phrases Dynamic complexity, parameterised complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.36

Related Version A full version of this paper is available at <https://arxiv.org/abs/1910.06281>.

Funding The authors acknowledge the financial support by DFG grant SCHW 678/6-2.

Acknowledgements We are grateful to Till Tantau for some valuable discussions.

1 Introduction

Parameterised complexity studies aspects of problems that make them computationally hard. The main interest has been in the class FPT which subsumes all problems that can be solved in time $f(k)\text{poly}(|x|)$ for an input x with a *parameter* $k \in \mathbb{N}$ and a computable function f . In recent work, much smaller parameterised classes have been studied, derived from classical classes in a uniform way by replacing the requirement of a polynomial bound of e.g. the circuit size (time, space, . . . , respectively) by a bound of the form $f(k)\text{poly}(|x|)$. In this fashion classical circuit classes AC^i and NC^i naturally translate to parameterised classes para-AC^i and para-NC^i . The lowest of these classes, para-AC^0 corresponds to the class AC^0 of problems computable by uniform families of constant-depth, polynomial size circuits with \wedge -, \vee - and \neg -gates of unbounded fan-in [19, 3].



© Jonas Schmidt, Thomas Schwentick, Nils Vortmeier, Thomas Zeume, and Ioannis Kokkinis; licensed under Creative Commons License CC-BY

28th EACSL Annual Conference on Computer Science Logic (CSL 2020).

Editors: Maribel Fernández and Anca Muscholl; Article No. 36; pp. 36:1–36:17



Leibniz International Proceedings in Informatics

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

This paper adds the aspect of changing inputs and dynamic maintenance of results to the exploration of the landscape between para-AC^0 and FPT .

The study of low-level complexity classes under dynamic aspects was started in [30, 15] in the context of dynamically maintaining the result of database queries. Similarly, as for *dynamic algorithms*, in this setting a dynamic program can make use of auxiliary relations that can store knowledge about the current input data (database). After a small change of the database (most often: insertion or deletion of a tuple), the program needs to compute the query result for the modified database in very short parallel time. To capture the problems/queries, for which this is possible, Patnaik and Immerman introduced the class DynFO [30]. Here, “FO” stands for first-order logic, which is equivalent to AC^0 , in the presence of arithmetic [7, 24].

In this paper, we study dynamic programs that have additional resources in a “parameterised sense”. We explore two such resources, which can be described as *parameterised space* and *parameterised time*, respectively. For ease of exposition, we discuss these two resources in the context of AC^0 first.

One way to strengthen AC^0 circuit families is to allow circuits of *size* $f(k)\text{poly}(|x|)$. We denote the class thus obtained as para-S-AC^0 (even though it corresponds to the class para-AC^0). A second dimension is to let the *depth* of circuits depend on the parameter. As the depth of circuits corresponds to the (parallel) time the circuits need for a computation, we denote the class of problems captured by such circuits by para-T-AC^0 . Of course, both dimensions can also be combined, yielding the parameterised class para-ST-AC^0 .

Surprisingly, several parameterised versions of NP-complete problems can even be solved in para-S-AC^0 . Examples are the vertex cover problem and the hitting set problem parameterised by the size of the vertex cover and the hitting set, respectively [4]. However, classical circuit lower bounds unconditionally imply that this is not possible for all FPT -problems. For instance, in [3] it was observed that the existence of simple paths of length k (the parameter) cannot be tested in para-S-AC^0 . Likewise, the feedback vertex set problem with the size of the feedback vertex set as parameter cannot be solved in para-ST-AC^0 .

When translated from circuits to logical formulas, depth roughly translates into iteration of formulas [24, Theorem 5.22], whereas size translates into the size of an additional structure by which the database is extended before formulas are evaluated. Slightly more formally, para-T-AC^0 corresponds to the class para-T-FO consisting of problems that can be defined by iterating a formula $f(k)$ many times. The class para-S-AC^0 corresponds to the class para-S-FO where formulas are evaluated on structures \mathcal{D} extended by an *advice structure* whose size depends on the parameter only. In the class para-ST-FO both dimensions are combined. The parameterised *dynamic* classes that we study in this paper are obtained from DynFO just like the above classes are obtained from FO : para-S-DynFO , para-T-DynFO and para-ST-DynFO extend DynFO by an additional structure of parameterised size, $f(k)$ iterations of formulas, or both, respectively.

As our first main contribution, we introduce a uniform framework for small dynamic, parameterised complexity classes (Section 3) based on advice structures (corresponding to additional space) or iterations of formulas (corresponding to additional time) and investigate how the resulting classes relate to each other and to other non-dynamic (and even non-parameterised) complexity classes (Section 4).

As our second main contribution, we explore how methods for parameterised algorithms can be applied in this framework through case studies for various parameterised problems (Section 5). Due to space limitations, many proofs are omitted and can be found in the full version of this paper.

Related work. There is a rich literature on parameterised dynamic algorithms, e.g. [23, 16, 28, 8, 1]. Closer to our work is the investigation of (static) parameterised small (parallel) complexity classes that was initiated 20 years ago in [9]. Later, in [19], parameterised versions of space and circuit classes were defined and several known parameterised problems were shown to be complete for these classes. Also in [3] it was shown, by applying the colour-coding technique, that several parameterised problems belong in para-AC^0 . Furthermore Chen and Flum [10] presented some unconditional proofs showing that some parameterised problems do not belong in para-AC^0 .

The descriptive complexity of parameterised classes has also been investigated in the past. For example Flum and Grohe [20] and Bannach and Tantau [6] presented syntactic descriptions of parameterised complexity classes using logical formulas. Additionally Chen, Flum and Huang [11] showed that the k -slices of several problems can be defined using FO-formulas of quantifier rank independent of k and explored the connection between the quantifier rank of FO-sentences and the depth of AC^0 -circuits.

2 Preliminaries

By $[n]$ we denote the set $\{1, \dots, n\}$. We assume familiarity with first-order logic FO and refer to [27] for basics of finite model theory. A (*relational*) *schema* τ consists of a set of relation symbols with a corresponding arity. A *structure* \mathcal{D} over schema τ with domain D has, for every relation symbol $R \in \tau$, a relation over D with the same arity as R . Throughout this work domains are finite. A k -*ary query* Q on τ -structures is a mapping that assigns a subset of D^k to every τ -structure over domain D and commutes with isomorphisms. Each first-order formula $\varphi(\bar{x})$ over schema τ defines a query Q whose result on a τ -structure \mathcal{D} is $\{\bar{a} \mid \mathcal{D} \models \varphi(\bar{a})\}$. Queries of arity 0 are also called *Boolean queries* or *problems*.

We mainly consider first-order formulas that have access to arithmetic, that is to a linear order $<$ on the domain as well as suitable, compatible addition $+$ and multiplication \times . We require that the result of the formulas is invariant¹ under the choice of the linear order $<$. This logic is referred to as *order-invariant first-order logic with arithmetic* and denoted by $\text{FO}(+, \times)$. In linearly ordered domains, we often identify domain elements with natural numbers, the smallest element representing 1.

Dynamic Complexity. We work in the dynamic complexity framework as introduced by Patnaik and Immerman [30], and refer to [32] for details. In a nutshell, dynamic programs answer a query for an input structure that is subjected to a sequence of changes. To this end they maintain an auxiliary structure using logical formulas.

By Δ_τ we denote the set of *single-tuple change operations* for a schema τ , which consists of the insertion operations INS_R and the deletion operations DEL_R for each relation $R \in \tau$. For example, $\text{INS}_E(a, b)$ could add edge (a, b) to a graph. A *dynamic query* (Q, Δ) consists of a query Q over some input schema τ_{in} and a set $\Delta \subseteq \Delta_{\tau_{\text{in}}}$. Later on we will sometimes consider slightly more general change operations.

A *dynamic program* \mathcal{P} for a dynamic query (Q, Δ) continuously answers Q on an *input structure* \mathcal{I} over some *input schema* τ_{in} under changes of the input structure from Δ . The domain D of \mathcal{I} is fixed and in particular changes cannot introduce new elements.² The program \mathcal{P} maintains an *auxiliary structure* \mathcal{A} over some *auxiliary schema* τ_{aux} with the

¹ In our scenario it is not relevant that invariance is undecidable for first-order formulas.

² We note that this is not a severe restriction, see e.g. [12, Theorem 17].

same domain as \mathcal{I} . We call $(\mathcal{I}, \mathcal{A})$ a *state* of \mathcal{P} and consider it as one relational structure. The auxiliary structure includes one particular *query relation* ANS that is supposed to contain the answer of Q over \mathcal{I} . For each auxiliary relation $S \in \tau_{\text{aux}}$ and each change operation $\delta \in \Delta$, \mathcal{P} has an update rule that specifies how S is updated after a change. It is of the form **on change** $\delta(\bar{p})$ **update** $S(\bar{x})$ **as** $\phi_{\delta}^S(\bar{p}; \bar{x})$ where the *update formula* $\phi_{\delta}^S(\bar{p}; \bar{x})$ is a formula over $\tau_{\text{in}} \cup \tau_{\text{aux}}$. For example, if the tuple \bar{a} is inserted into an input relation R , each auxiliary relation S is replaced by the relation $\{\bar{b} \mid (\mathcal{I}, \mathcal{A}) \models \phi_{\text{INS}_R}^S(\bar{a}; \bar{b})\}$. By $\alpha(\mathcal{I})$ we denote the input structure that results from \mathcal{I} by applying a sequence α of changes, and by $\mathcal{P}_{\alpha}(\mathcal{I}, \mathcal{A})$ the state $(\alpha(\mathcal{I}), \mathcal{A}')$ of \mathcal{P} that results from $(\mathcal{I}, \mathcal{A})$ after processing α . The dynamic program \mathcal{P} *maintains* (Q, Δ) if the relation ANS in $\mathcal{P}_{\alpha}(\mathcal{I}_0, \mathcal{A}_0)$ equals the query result $Q(\alpha(\mathcal{I}_0))$, for each sequence α of changes over Δ , each initial input structure \mathcal{I}_0 with arbitrary (finite) domain and empty relations, and the auxiliary structure \mathcal{A}_0 with empty relations.

The class DynFO is the set of dynamic queries that can be maintained by a dynamic program with first-order update formulas. The class $\text{DynFO}(+, \times)$ is defined analogously via $\text{FO}(+, \times)$ update formulas. We note that in the case of $\text{DynFO}(+, \times)$, we consider the arithmetic relations to be part of the input structure \mathcal{I} , but they can not be modified. Technically, an additional schema τ_{arith} contains the arithmetic predicates and the update formulas are over $\tau_{\text{in}} \cup \tau_{\text{aux}} \cup \tau_{\text{arith}}$. Note that τ_{arith} cannot be used for defining a query.

Parameterised Complexity. A *parameterised query* is a pair (Q, κ) , where Q is a query over some schema τ and κ is a function, called the *parameterisation*, that assigns a parameter from \mathbb{N} to every τ -structure. The well-known parameterised complexity class FPT contains all Boolean parameterised queries (Q, κ) having an algorithm that decides for each τ -structure \mathcal{D} whether $\mathcal{D} \in Q$ in time $f(\kappa(\mathcal{D}))|\mathcal{D}|^c$, for some constant c and computable function $f: \mathbb{N} \rightarrow \mathbb{N}$ [17]. Like [5], we demand that κ is first-order definable, which is always the case if the parameter is explicitly given in the input.

► **Example 1.** p - VERTEXCOVER is a well-studied parameterised query. Formally it is the set Q of pairs (G, k) , where G is an undirected graph that has a vertex cover of size k , together with the parameterisation $\kappa: (G, k) \mapsto k$. In more accessible notation:

Problem: p - VERTEXCOVER

Input: An undirected graph $G = (V, E)$ and $k \in \mathbb{N}$, **Parameter:** k

Question: Is there a set $S \subseteq V$ such that $|S| = k$ and $u \in S$ or $v \in S$ for every $(u, v) \in E$?

The search-tree based algorithm for p - VERTEXCOVER is a classical parameterised algorithm. It is based on the simple observation that, for each edge (u, v) of a graph, each vertex cover needs to contain u or v (or both). On input (G, k) the algorithm recursively constructs the search tree as follows, starting from the root of an otherwise empty tree. If E is empty it accepts, otherwise it rejects if $k = 0$. If $k > 0$ it chooses some edge $(u, v) \in E$, labels the current node with (u, v) , and constructs two new tree nodes below the current node. It then continues recursively, from both children starting from the instance $(G - u, k - 1)$ in the first child, and from $(G - v, k - 1)$ in the second child. The algorithm accepts if any of its branches accepts. Since the inner nodes of the tree have two children and its depth is bounded by k , it can have at most $2^{k+1} - 1$ tree nodes. The overall running time can be bounded by $\mathcal{O}(2^k n^2)$. Thus p - $\text{VERTEXCOVER} \in \text{FPT}$.

3 A Framework for Parameterised, Dynamic Complexity

We first present a uniform point of view on parameterised first-order logic. As explained in the introduction, formulas can be parameterised with respect to (at least) two dimensions: additional time by iterating formulas with the number of iterations depending on the parameter; additional space by advice structures whose size depends on the parameter.

A *first-order program* \mathcal{F} over schema τ is a tuple (Ψ, φ) where Ψ is a set of $\text{FO}(+, \times)$ -formulas over schema $\tau \uplus \tau_\Psi$ and $\varphi \in \Psi$ is supposed to compute the final result of the program. Here, τ_Ψ is a schema that contains a fresh relation symbol R_ψ for each formula $\psi \in \Psi$ of the same arity as ψ . The semantics of \mathcal{F} on a τ -structure \mathcal{D} is based on inductively defined τ_Ψ -structures $\mathcal{D}_\Psi^{(\ell)}$. Initially, in $\mathcal{D}_\Psi^{(0)}$, all relations $R_\psi^{(0)}$ are empty. The ℓ -step result $\mathcal{D}_\Psi^{(\ell)}$ of \mathcal{F} , for $\ell > 0$, is defined via $R_\psi^\ell \stackrel{\text{def}}{=} \{\bar{a} \mid (\mathcal{D}, \mathcal{D}_\Psi^{(\ell-1)}) \models \psi(\bar{a})\}$. Finally, the *result* $\mathcal{F}(\mathcal{D})$ is $R_\varphi^{(\ell)}$ if $\mathcal{D}_\Psi^{(\ell-1)} = \mathcal{D}_\Psi^{(\ell)}$, for some ℓ . In this case, we say that the program reaches a fixed point after ℓ steps. Otherwise, $\mathcal{F}(\mathcal{D})$ is the empty set.

We now define how first-order programs can use advice. An τ_{adv} -*advice* π is a computable mapping from \mathbb{N} to τ_{adv} -structures for some fixed advice schema τ_{adv} . Suppose that \mathcal{F} is a first-order program over schema $\tau \uplus \tau_{\text{adv}}$. The result of \mathcal{F} for a τ -structure \mathcal{D} with advice π and parameter $k \in \mathbb{N}$ is simply the result of \mathcal{F} on the structure $\mathcal{D} \uplus \pi(k)$.

For two computable functions $f, g : \mathbb{N} \rightarrow \mathbb{R}$ and a parameterised query (Q, κ) over a schema τ , an (f, g) -*parameterised first-order program for* (Q, κ) is a tuple (\mathcal{F}, π) where \mathcal{F} is a first-order program over schema $\tau \uplus \tau_{\text{adv}}$ and π is an τ_{adv} -advice such that

- (a) the result of \mathcal{F} with advice π is $Q(\mathcal{D})$, for all τ -structures \mathcal{D} ;
- (b) $|\pi(\kappa(\mathcal{D}))| \leq f(\kappa(\mathcal{D}))$ for all τ -structures \mathcal{D} ; and
- (c) \mathcal{F} always reaches a fixed point and does so after at most $g(\kappa(\mathcal{D}))$ steps.

For computable functions f and g let $\text{para-ST-FO}(f, g)$ be the class of parameterised queries definable by an (f, g) -parameterised first-order program. We note that these programs use $\text{FO}(+, \times)$ formulas, and thus have access to arithmetic³ over the domain of $\mathcal{D} \uplus \pi(k)$. We do not make this explicit in our naming scheme. We use the following abbreviations:

- $\text{para-ST-FO} \stackrel{\text{def}}{=} \bigcup_{f, g} \text{para-ST-FO}(f, g)$,
- $\text{para-S-FO} \stackrel{\text{def}}{=} \bigcup_f \text{para-ST-FO}(f, 1)$,
- $\text{para-T-FO} \stackrel{\text{def}}{=} \bigcup_g \text{para-ST-FO}(0, g)$.

The class para-S-FO is in fact the same as para-AC^0 , and para-ST-FO corresponds to the class $\text{para-AC}^{0\uparrow}$ in [3]. To the best of our knowledge, para-T-FO has not been studied in the context of first-order logic before.

► **Example 2.** We sketch a first-order program $\mathcal{F} = (\Psi, \varphi)$ that witnesses $p\text{-VERTEXCOVER} \in \text{para-T-FO}$. Recall the search-tree based parameterised algorithm for $p\text{-VERTEXCOVER}$ from Example 1. Intuitively, the formulas $\psi \in \Psi$ are used to traverse the search tree in a depth-first manner. At any moment, the auxiliary relations contain information about the path from the root to the current node. In particular, the *candidate set* of the current node, i.e., the set of vertices selected along its path is available. Each application of these formulas simulates one elementary step of the search: either a new child is added to the current path, or, if the current node has maximal depth or if all possible children were already added, the current node is discarded and a backtrack step to its parent is performed. If the candidate set is a vertex cover, the search ends. Since each edge of the search tree needs to be traversed at most twice, 2^{k+2} iterative steps suffice. More detail is given in the full version.

³ In particular, “ $+|\mathcal{D}|$ ” induces a correspondence between \mathcal{D} and $\pi(k)$.

The following lemma basically states that every boolean parameterised query can be answered in para-S-FO on instances whose domain size is bounded by a function in the parameter.

► **Lemma 3.** *Let $f: \mathbb{N} \rightarrow \mathbb{N}$ be a computable function and (Q, κ) a boolean parameterised query with decidable Q . There is a computable function g and a $(g, 1)$ -parameterised first-order program (φ, π) that answers Q correctly on instances \mathcal{D} of size at most $f(\kappa(\mathcal{D}))$.*

Proof idea. We explain the proof idea for input structures consisting of a graph G of size n and a parameter value k with $n \leq f(k)$. The advice π produces an advice structure with domain $[2^{f(k)^2}]$. It has a ternary relation E' that contains, for every $i \in [2^{f(k)^2}]$ all tuples (i, j_1, j_2) , for which the i -th graph over $[f(k)]$ in some canonical enumeration has an edge (j_1, j_2) . It further contains a unary relation F that contains all numbers i , for which the i -th graph is a yes-instance of Q . The formula φ simply determines with the help of E' and built-in arithmetic the number i of G (as a graph over $[n]$) and tests whether $F(i)$ holds. ◀

Parameterised Dynamic Complexity. We study parameterised queries in a dynamic context. Formally, a *dynamic parameterised query* (Q, κ, Δ) consists of a parameterised query (Q, κ) and a set Δ of change operations. We say that a parameterised query (Q, κ) has an *explicit parameter*, if Q consists of pairs $\mathcal{I} = (\mathcal{I}', k)$, where \mathcal{I}' is a structure, k is a suitably encoded number, and $\kappa(\mathcal{I}) = k$. All concrete parameterised queries we consider in this paper have an explicit parameter. For example, we often consider the dynamic variant $(p\text{-VERTEXCOVER}, \Delta_E \cup \pm 1)$ of the parameterised vertex cover query, where $\Delta_E \stackrel{\text{def}}{=} \{\text{INS}_E, \text{DEL}_E\}$ and $\pm 1 \stackrel{\text{def}}{=} \{+1, -1\}$ denotes the set of change operations that increment or decrement the given number k by one, as long as k stays in the admissible range. So, given some graph G with n vertices, $+1(G, k) \stackrel{\text{def}}{=} (G, k + 1)$ if $k < n$, and $-1(G, k) \stackrel{\text{def}}{=} (G, k - 1)$ if $k > 1$, and otherwise the changes have no effect.

For most queries⁴ in this paper only parameter values in $\{1, \dots, n\}$ are meaningful and we only allow such values. They can be represented by elements of the domain.

Similarly as parameterised first-order programs generalise first-order formulas, parameterised dynamic programs extend conventional dynamic programs in two directions: (1) they may use an advice structure whose size depends on the parameter, and (2) they may use first-order programs of parameterised iteration depth.

A *dynamic program with iteration and advice* is a tuple (\mathcal{P}, π) where \mathcal{P} is a dynamic program where auxiliary relations are updated with first-order programs and π is an τ_{adv} -advice for an advice schema τ_{adv} . For a dynamic parameterised query (Q, κ, Δ) , the program \mathcal{P} has update rules of the form **on change** $\delta(\bar{p})$ **update** $S(\bar{x})$ **as** (Ψ_S, φ_S) for every $\delta \in \Delta$, where (Ψ_S, φ_S) is a first-order program over schema $\tau_{\text{in}} \cup \tau_{\text{aux}} \cup \tau_{\text{adv}}$ such that φ_S has the same arity as S . States of the program \mathcal{P} are of the form $(D \uplus D_{\text{adv}}, \mathcal{I}, \mathcal{A}, \mathcal{A}_{\text{adv}})$ where \mathcal{I} is the input structure, \mathcal{A} the auxiliary structure, and \mathcal{A}_{adv} is an advice structure over a schema τ_{adv} . Tuples of the auxiliary structure \mathcal{A} may range over the domain $D \uplus D_{\text{adv}}$.

For two computable functions $f, g: \mathbb{N} \rightarrow \mathbb{R}$, an (f, g) -parameterised dynamic program is a dynamic program (\mathcal{P}, π) with iteration and advice such that $|\pi(k)| \leq f(k)$ for all $k \in \mathbb{N}$ and all first-order programs of \mathcal{P} always reach a fixed point after at most $g(\kappa(\mathcal{I}))$ steps. The initial state of such a program depends on an initial input structure \mathcal{I}_0 and a number $k \in \mathbb{N}$. It is given as $(D \cup D_{\text{adv}}, \mathcal{I}_0, \mathcal{A}_0, \mathcal{A}_{\text{adv}}^k)$ where $\mathcal{A}_{\text{adv}}^k \stackrel{\text{def}}{=} \pi(k)$, D and D_{adv} are the domains of \mathcal{I}_0 and $\pi(k)$, respectively, and \mathcal{A}_0 is an empty τ_{aux} -structure.

⁴ The only exception is p -KNAPSACK in Section 5.4.

A dynamic parameterised query (Q, κ, Δ) is maintained by (\mathcal{P}, π) if a distinguished relation ANS in $\mathcal{P}_\alpha(D \cup D_{\text{adv}}, \mathcal{I}_0, \mathcal{A}_0, \mathcal{A}_{\text{adv}}^k)$ equals $Q(\alpha(\mathcal{I}_0))$, for all empty⁵ input structures \mathcal{I}_0 , all $k \in \mathbb{N}$, and all sequences α of changes over Δ such that $\kappa(\alpha'(\mathcal{I}_0)) \leq k$ for all prefixes α' of α . So, the dynamic program (\mathcal{P}, π) only needs to maintain (Q, κ, Δ) as long as the parameter value is bounded by the initially given number k ; nevertheless the program needs to work for arbitrary values of k . We denote this number k in the following as k_{max} .

For computable functions $f, g : \mathbb{N} \rightarrow \mathbb{R}$ we define $\text{para-ST-DynFO}(f, g)$ as the class of dynamic parameterised queries that can be maintained by an (f, g) -parameterised dynamic program. We define:

- $\text{para-ST-DynFO} \stackrel{\text{def}}{=} \bigcup_{f, g} \text{para-ST-DynFO}(f, g)$,
- $\text{para-S-DynFO} \stackrel{\text{def}}{=} \bigcup_f \text{para-ST-DynFO}(f, 1)$,
- $\text{para-T-DynFO} \stackrel{\text{def}}{=} \bigcup_g \text{para-ST-DynFO}(0, g)$,

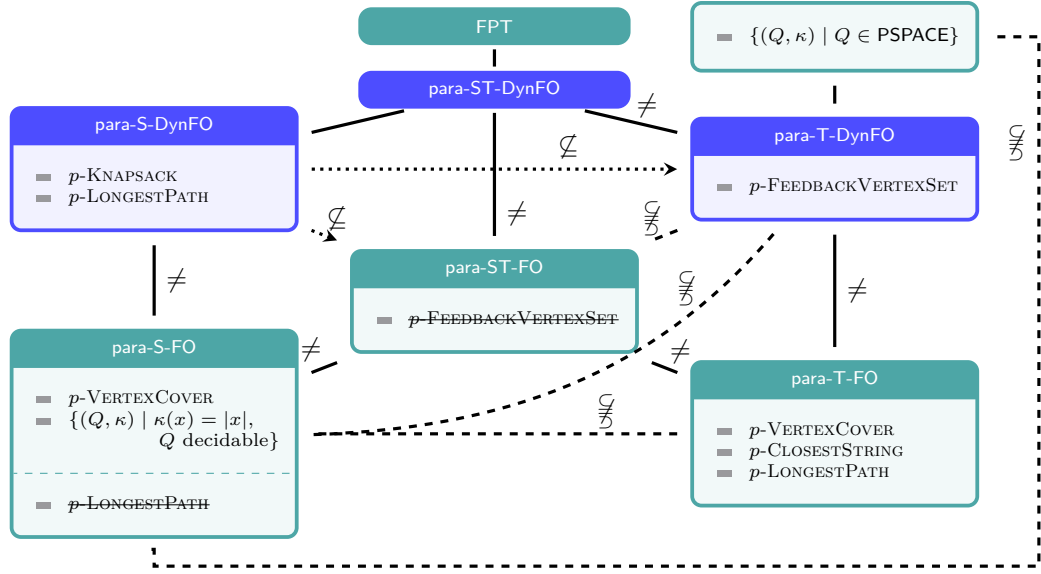
Since the purpose of this article is to explore the basic principles of parameterised, dynamic complexity, we keep the setting simple, in particular with respect to the following two aspects. First, dynamic programs get a bound k_{max} for the parameter values at initialisation time and the program then only needs to deal with changes that obey this parameter bound. This ensures that the advice structure does not change throughout the dynamic process. Second, we assume the presence of arithmetic throughout. In non-parameterised dynamic complexity, it is known that under mild assumptions on the query, arithmetic relations can be constructed by a dynamic program on the fly [12]. Similar techniques can be applied for the parameterised setting, yet we ignore this aspect here and assume that $\mathcal{I}_0 \uplus \pi(k)$ comes with relations $<$, $+$, and \times over $D \uplus D_{\text{adv}}$.

For some first intuition we provide a parameterised dynamic program that shows that $(p\text{-VERTEXCOVER}, \{\text{INS}_E\} \cup \pm 1)$ is in para-S-DynFO via the search-tree based approach. This result is not surprising, as it is known that $p\text{-VERTEXCOVER} \in \text{para-S-FO}$ [11, 4]. However, the dynamic program for maintaining search trees is conceptually very simple.

► **Example 4.** We recall the search-tree based parameterised algorithm for $p\text{-VERTEXCOVER}$ from Example 1. The first-order program of Example 2 witnesses $p\text{-VERTEXCOVER} \in \text{para-T-FO}$ (and thus also in para-T-DynFO) by constructing a search tree from scratch. In contrast, a dynamic program witnessing $(p\text{-VERTEXCOVER}, \{\text{INS}_E\} \cup \pm 1) \in \text{para-S-DynFO}$ can *maintain* a search tree. To this end, for a given bound k_{max} , its advice structure $\mathcal{A}_{\text{adv}}^{k_{\text{max}}}$ stores a full binary “background” tree T of depth k_{max} . Its auxiliary structure represents the actual search tree T' by maintaining an upward closed set of nodes and the candidate sets of each of those nodes. As in the search tree algorithm from Example 1, in every inner node x of T' a branching on the endpoints of some edge e of G is being simulated and in each of x 's two children one vertex of e is added to the candidate set. A node x of T is a leaf of T' , if the assigned candidate set of x is an actual vertex cover of G or if x is in level k_{max} of T . The program then only needs to check whether there is a leaf representing a valid vertex cover at a level below the current value of k . Maintenance under changes from ± 1 is therefore easy.

Maintaining T' under insertion of an edge (u, v) is easy as well: for each leaf of T' that is *not* at level k_{max} , and whose candidate set does not cover (u, v) , the program adds u to the left child and v to the right child (assuming $u < v$). Leaves at level k_{max} are not modified, but it might happen that a former vertex cover attached to such a leaf becomes invalid by not covering (u, v) . Maintaining T' under edge *deletions* is slightly more subtle and will be considered in the proof of Proposition 10.

⁵ For queries with explicit parameter, we require only that in $\mathcal{I}_0 = (\mathcal{I}'_0, k)$, \mathcal{I}'_0 is empty, but k can be non-zero.



■ **Figure 1** Inclusion diagram of the main classes. Solid lines indicate inclusions. Dashed lines marked with $\not\subseteq$ indicate that the two classes are incomparable. A directed, dotted edge marked with \subsetneq from \mathcal{C} to \mathcal{C}' indicates $\mathcal{C} \setminus \mathcal{C}' \neq \emptyset$. If \mathcal{C} is a dynamic class and \mathcal{C}' a static class, $\mathcal{C} \subseteq \mathcal{C}'$ means that for each $(Q, \kappa, \Delta) \in \mathcal{C}$ with exhaustive Δ it holds that $(Q, \kappa) \in \mathcal{C}'$, and $\mathcal{C}' \subseteq \mathcal{C}$ means that for each $(Q, \kappa) \in \mathcal{C}'$ it holds that $(Q, \kappa, \Delta) \in \mathcal{C}$, for arbitrary Δ .

4 Relationships between Parameterised Classes

In this section we examine how parameterised dynamic and static complexity classes relate to each other. These relationships are summarised in Figure 1.

As a sanity check, we show first that every parameterised query (Q, κ) with $(Q, \kappa, \Delta) \in \text{para-ST-DynFO}$ is in FPT. For queries in para-T-DynFO the respective algorithm only needs polynomial space. Both statements require that Δ is *exhaustive*, i.e., that it contains the single-tuple insertion operation INS_R for every input relations R . This ensures that every possible input structure for Q can be obtained by a change sequence.⁶

► **Proposition 5.**

- (a) For every $(Q, \kappa, \Delta) \in \text{para-ST-DynFO}$ with exhaustive Δ it holds that $(Q, \kappa) \in \text{FPT}$.
- (b) For every $(Q, \kappa, \Delta) \in \text{para-T-DynFO}$ with exhaustive Δ , the parameterised query (Q, κ) can be solved by an FPT-algorithm that uses at most polynomial space with respect to the input size. In particular, $Q \in \text{PSPACE}$.

Statement (b) does not hold for parameterised classes with advice, as we formalise with the next proposition, which is an immediate consequence of Lemma 3.

► **Proposition 6.** Every parameterised query (Q, κ) with decidable Q and $\kappa(x) = |x|$ is in para-S-FO.

► **Proposition 7.** For any $(Q, \kappa) \in \text{para-S-FO}$ and any $\Delta \subseteq \Delta_{\tau_{\text{in}}}$ (or $\Delta \subseteq \Delta_{\tau_{\text{in}}} \cup \pm 1$) it holds that $(Q, \kappa, \Delta) \in \text{para-S-DynFO}$.

⁶ Clearly, a more general definition would be possible here, but we avoid that in the interest of simplicity.

Proof sketch. Let $(Q, \kappa) \in \text{para-S-FO}$ by some $(f, 1)$ -parameterised FO program \mathcal{F} . In principle, a parameterised dynamic program can simulate \mathcal{F} from scratch after each change. However, since the parameter of \mathcal{I} might change, it might need different advice structures from \mathcal{F} . However, there is an easy solution for this. For the given k_{\max} , the dynamic program gets as its advice *all* advice structures $\pi(1), \dots, \pi(k_{\max})$ of \mathcal{F} . ◀

The same argument can be applied for para-ST-FO and para-ST-DynFO .

In addition to the above inclusions and those that are immediate from the definitions, we observe the following separations between parameterised classes (also see Figure 1). Some proofs are deferred to the next section.

► **Proposition 8.**

- (a) *There is a $(Q, \kappa) \in \text{para-S-FO}$ such that $(Q, \kappa, \Delta) \notin \text{para-T-DynFO}$, for any exhaustive Δ .*
- (b) *There is a $(Q, \kappa) \in \text{para-T-FO}$ such that $(Q, \kappa) \notin \text{para-S-FO}$.*
- (c) *There is a $(Q, \kappa, \Delta) \in \text{para-T-DynFO}$ with exhaustive Δ such that $(Q, \kappa) \notin \text{para-ST-FO}$.*
- (d) *There is a $(Q, \kappa, \Delta) \in \text{para-S-DynFO}$ with exhaustive Δ such that $(Q, \kappa) \notin \text{para-ST-FO}$.*

Proof sketch. Part (a) is a consequence of Proposition 5 and Proposition 6, and witnessed by any parameterised problem (Q, κ) with decidable $Q \notin \text{PSPACE}$ and $\kappa(x) = |x|$. Part (b) is witnessed by the problem p -LONGESTPATH which is not in para-S-FO [3], but in para-T-FO as we will see in Proposition 9. For (c) we observe that p -FEEDBACKVERTEXSET is not in para-ST-FO , as otherwise the restriction to inputs with parameter $k = 0$ would yield a first-order formula that expresses acyclicity of undirected graphs. In Proposition 12 we will show that $(p\text{-FEEDBACKVERTEXSET}, \Delta_E \cup \pm 1)$ is in para-T-DynFO . The separation for (d) can be shown with the help of connectivity of undirected graphs. To this end, we consider the parameterisation by the maximal node degree. It is well-known that even for fixed $k = 2$ this property is not expressible in $\text{FO}(+, \times)$, see [21], and thus it is not in para-ST-FO . On the other hand, towards (d), the unparameterised version is in DynFO and thus the parameterised version is in para-S-DynFO .⁷ ◀

5 Methods for Parameterised Complexity

The goal of this section is to explore the transferability of known methods from the realm of parameterised algorithms to dynamic parameterised complexity. We are thus not always interested in “best algorithms” but rather want to exemplify how sequential algorithmic methods for static problems translate into the dynamic (highly parallel) setting.

We start by describing colour-coding, since it turns out as particularly useful in the dynamic context and we use it in many other subsections. Then we consider three classical methods for parameterised algorithms, bounded search trees, kernelisation and dynamic programming. Afterwards we give an example for the iterated compression method, which uses an adaption of a technique from dynamic complexity.

5.1 Colour-Coding

In this subsection, we establish the usefulness of the colour-coding technique, as presented in [2], in our setting by a concrete example, p -LONGESTPATH.

⁷ Of course, this argument could have been used for (c) as well, but there we prefer a more “natural” parameterisation.

36:10 Dynamic Complexity Meets Parameterised Algorithms

Problem: p -LONGESTPATH

Input: An undirected graph $G = (V, E)$, $s, t \in V$ and $\ell \in \mathbb{N}$, **Parameter:** ℓ

Question: Is there a (simple) path from s to t of length ℓ ?

This problem can be solved with the help of *universal colouring families*. Such a family is a small set of functions that map nodes to colours such that if a path of length ℓ exists, one of these functions colours the nodes of the path with a fixed sequence of $\ell + 1$ colours. A parallel algorithm for p -LONGESTPATH therefore only needs to test in parallel, for each function of a universal colouring family, whether it produces such a coloured path from s to t .

More precisely, a (n, k, c) -universal colouring family Λ has, for every subset $S \subseteq [n]$ of size k and for every mapping $\mu : S \rightarrow [c]$, at least one function $\lambda \in \Lambda$ with $\lambda(s) = \mu(s)$, for every $s \in S$. In [3, Theorem 3.2] a family $\Lambda_{n,k,c}$ of such functions is defined. The definition can be found in the full version. In the presence of arithmetic, these functions are easily first-order definable and can be enumerated in a first-order fashion.

► **Proposition 9.**

(a) p -LONGESTPATH \in para-S-DynFO.

(b) p -LONGESTPATH \in para-T-FO.

Proof sketch. In both parts of the proof, we use the colour-coding approach as sketched above. For a graph G , a colouring function λ , and a set C of colours, a C -coloured path under λ is a path whose nodes are mapped to C in a one-one fashion by λ .

For solving the p -LONGESTPATH problem with parameter ℓ , we consider the (n, k, k) -universal colouring family $\Lambda \stackrel{\text{def}}{=} \Lambda_{n,k,k}$ with $k \stackrel{\text{def}}{=} \ell + 1$. Then a graph has a simple path of length ℓ from s to t if and only if there is a $[k]$ -coloured path from s to t under some $\lambda \in \Lambda$.

We first show p -LONGESTPATH \in para-S-DynFO. The dynamic program uses a dynamic programming approach (in the classical sense of this term). It stores, for each $\lambda \in \Lambda$ and each pair (u, v) of nodes, the set \mathcal{C} of color sets C , for which there is a C -coloured path from u to v under λ .

That p -LONGESTPATH \in para-T-FO can be shown with the help of the same universal colouring family Λ as above, which consists of $f(k)\text{poly}(n)$ colourings. The idea for the program is to test, in $f(k)$ iterations and in each iteration for $\text{poly}(n)$ colourings in parallel, whether there is a $[k]$ -coloured path from s to t under the current colouring. A suitably coloured path can be found in k iterations. ◀

5.2 Bounded-depth search trees

Bounded-depth search trees are a classical technique in parameterised complexity. Already in Example 4 we outlined that search trees are a viable tool also in the dynamic context by showing how a search tree for p -VERTEXCOVER can be maintained under edge insertions. Here we provide more examples. First we extend Example 4 towards edge deletions. Afterwards we consider two further problems, for which the known search-tree based algorithms can be adapted to place them in para-T-FO or para-T-DynFO, respectively: p -CLOSESTSTRING and p -FEEDBACKVERTEXSET. Although we conjecture that these problems are also in para-S-DynFO, we were not able to prove it.

► **Proposition 10.** $(p\text{-VERTEXCOVER}, \Delta_E \cup \pm 1) \in$ para-S-DynFO *by a search-tree-based dynamic program.*

Proof sketch. Let T and T' be defined as in Example 4. It remains to explain how edge deletions can be handled. If an edge (u, v) is deleted, and a node x of T' used (u, v) for its branching step, the induced subtree of x can be replaced by the induced subtree of its left

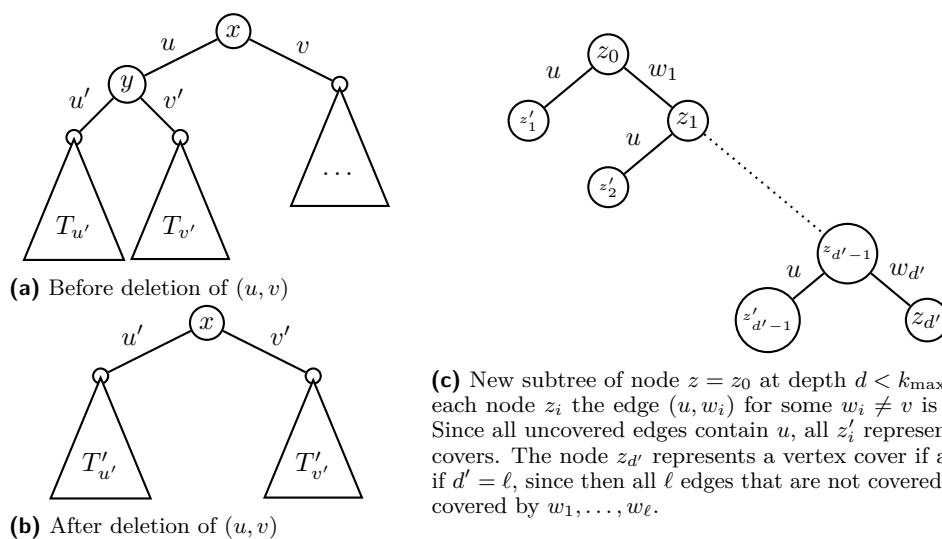


Figure 2 Modification of the search tree for p -VERTEXCOVER after deletion of an edge (u, v) . The new sub-trees $T'_{u'}$, $T'_{v'}$ of x are obtained from $T_{u'}$, $T_{v'}$ respectively, by adding two new children to leaves that do not represent a vertex cover.

child y , see Figure 2.⁸ More precisely, the children u' and v' of y become the new children of x , and in all candidate sets below u' and v' the vertex u is removed.

The subtree of x might now (1) have leaves of depth $k_{\max} - 1$ that do not represent an actual vertex cover, since the modification reduces the depth of all nodes in the subtree of x , and (2) have leaves at a smaller depth $d < k_{\max} - 1$ which do not represent a vertex cover, since u is removed from the candidate sets and thus edges adjacent to u may not be covered any more. These defects can be corrected successively.

First, for each of the leaves from (1), two new children are added, with the help of the lexicographically smallest uncovered edge (u'', v'') .

Regarding a leave z with property (2), observe that its candidate set can miss only edges of the form (u, w) , where $w \neq v$. It is easy to see that the subtree rooted at z can be chosen in the following shape. Let $W = \{w_1, \dots, w_\ell\}$ be the set of vertices with an uncovered edge $(u, w_i), i \in [\ell]$. The new subtree having depth $d' = \min\{\ell, k_{\max} - d\}$ consists of a path with nodes $z_0, \dots, z_{d'}$ such that $z_0 = z$ and for each $i \geq 0$, the left child of z_i is a leaf obtained by adding u to the candidate set and for the right child z_{i+1} , w_{i+1} is added to the candidate set.

This new subtree can be defined in a first-order fashion with the help of colour coding. Let U be the candidate set of z . Then W consists of all neighbours of u that are not in U , so W is easily FO-definable. To define the subtree, d' vertices have to be chosen from W . To this end, we consider colourings of W that map W to $[\ell]$. With the help of an (n, k_{\max}, k_{\max}) -universal colouring family, one can quantify over such colourings and by picking (a canonical) one, the new subtree can be defined by choosing each w_i as the node coloured with i , for every $i \in [d']$. All these updates can be expressed by first-order formulas. ◀

For the closest string problem, we fix an alphabet Σ , and let $d_H(s_1, s_2)$ denote the Hamming distance of s_1 and s_2 , i.e. the number of positions where s_1 and s_2 differ.

⁸ Of course, the right child would work equally well.

36:12 Dynamic Complexity Meets Parameterised Algorithms

Problem: p -CLOSESTSTRING

Input: Strings $s_1, \dots, s_n \in \Sigma^n$ for some $n \in \mathbb{N}$, and $d \in \mathbb{N}$, **Parameter:** d

Question: Is there a string $s \in \Sigma^n$ such that $d_H(s, s_i) \leq d$?

An input to p -CLOSESTSTRING with strings of length n is represented by a structure with domain $[n]$. It has the natural linear order on $[n]$ and, for every $\sigma \in \Sigma$ a relation $R_\sigma(i, j)$ with the meaning $s_i[j] = \sigma$, i.e. string s_i has symbol σ at position j .

A search tree (see [29, Section 8.5]) of depth at most d and degree at most $d + 1$ gradually adapts a candidate string s , which is initially set to s_1 . If an input string s_i is “far apart” from s , the tree branches on the first $d + 1$ differences and changes s towards s_i .

► **Proposition 11.** p -CLOSESTSTRING \in para-T-FO.

The construction is quite straightforward and can be found in the full version.

Next, we explore the parameterised problem p -FEEDBACKVERTEXSET. Given a graph $G = (V, E)$, a feedback vertex set (FVS) for G is a set $S \subseteq V$ such that for every cycle C in G , $S \cap C \neq \emptyset$ holds, i.e. $G - S$ is a forest.

Problem: p -FEEDBACKVERTEXSET

Input: An undirected graph G , **Parameter:** k

Question: Does G have a feedback vertex set of size k ?

► **Proposition 12.** $(p$ -FEEDBACKVERTEXSET, $\Delta_E \cup \pm 1$) \in para-T-DynFO.

Proof idea. We show that p -FEEDBACKVERTEXSET can be maintained in para-T-DynFO using a depth-bounded search tree, similarly as for p -VERTEXCOVER. The result uses a well-known approach relying on the fact that if a graph of minimum degree 3 has a FVS of size k then the length of its minimal cycle is bounded by $2k$ (e.g. [18]). A branching step consists of two phases: removing vertices of degree 1 or 2, and finding a small cycle. Then, each branch selects one of these cycle vertices for the FVS candidate. At the leaves of the search tree it has to be checked if the graph obtained by deleting the chosen vertices of the current branch is acyclic. A cycle exists, if there exists an edge (u, v) and u is reachable from v in $G - (u, v)$, thus this can be decided with the transitive closure of the edge relation. The latter can be maintained in DynFO under edge insertions and deletions [12] and, as we show in the full version of this paper, also under vertex deletions (simulated by removing all edges of a vertex). ◀

5.3 Kernelisation

Bannach and Tantau [5, Theorem 2.3] show that the famous meta-theorem “a problem is fixed parameter tractable if and only if a kernel for it can be computed in polynomial time” can be adapted to connect the AC-hierarchy with its parameterised counterpart. In this section we (partially) translate this relationship to the parameterised, dynamic setting.

A *kernelisation* of a Boolean parameterised query (Q, κ) over schema τ is a self-reduction K from τ -structures to τ -structures such that (1) $\mathcal{I} \in Q$ if and only if $K(\mathcal{I}) \in Q$, and (2) $|K(\mathcal{I})| \leq h(\kappa(\mathcal{I}))$, for all τ -structures \mathcal{I} and some fixed computable function $h : \mathbb{N} \rightarrow \mathbb{N}$. The images of a kernelisation K are called *kernels*. We say that a kernel of (Q, κ) can be maintained in some class \mathcal{C} under some set Δ of change operations, if the kernels with respect to some kernelisation K can be maintained in \mathcal{C} under changes from Δ .

► **Theorem 13.** *Let (Q, κ, Δ) be a Boolean parameterised dynamic query of τ -structures.*

- (a) *If a kernel for (Q, κ) can be maintained under Δ in $\text{DynFO}(+, \times)$ then (Q, κ, Δ) is in para-S-DynFO. In addition, if (Q, κ) has an explicit parameter and $\Delta = \Delta_\tau \cup \pm 1$ then also the converse holds.*
- (b) *If $Q \in \text{PSPACE}$ and a kernel for (Q, κ) can be maintained under Δ in $\text{DynFO}(+, \times)$ then (Q, κ, Δ) is in para-T-DynFO.*

Proof sketch. Towards proving (a), suppose that a kernel of (Q, κ) with respect to a kernelisation K can be maintained under Δ by a $\text{DynFO}(+, \times)$ -program \mathcal{P} . A para-S-DynFO-program \mathcal{P}' for (Q, κ, Δ) maintains a kernel for the current input structure by simulating \mathcal{P} . The kernel $K(\mathcal{I})$ of an input structure \mathcal{I} is represented by at most $h(\kappa(\mathcal{I}))$ elements, where h is the function from the second condition of the definition of the kernelisation K . Therefore \mathcal{P}' can check whether $K(\mathcal{I}) \in Q$ by Lemma 3 and Proposition 7.

For proving the converse of (a) under the stated assumptions, suppose that (Q, κ) has an explicit parameter and that $\Delta = \Delta_\tau \cup \pm 1$. We construct, from a para-S-DynFO-program \mathcal{P} with advice π that maintains (Q, κ, Δ) , a $\text{DynFO}(+, \times)$ -program \mathcal{P}' that maintains a kernel for (Q, κ) . The idea is to use a standard trick from parameterised complexity, a case distinction between small and large parameters. If the parameter is small enough in comparison to the domain size, \mathcal{P}' can compute the advice structure of \mathcal{P} at initialisation time and can simulate \mathcal{P} from then on. If the parameter is large, \mathcal{P}' uses the “small” input instance as a trivial kernel.

Towards proving (b), suppose that a kernel of (Q, κ) with respect to a kernelisation K can be maintained under Δ by a $\text{DynFO}(+, \times)$ -program \mathcal{P} , and that $Q \in \text{PSPACE}$. Recall that unlimited (or equivalently exponential) iteration of FO-formulas captures PSPACE over ordered structures (see, e.g., [24, Theorem 10.13]). A para-T-DynFO-program can maintain the current kernel $K(\mathcal{I})$ by simulating \mathcal{P} . After updating the kernel after a change, it computes the result of Q for $K(\mathcal{I})$ by iterating the first-order formulas of the PSPACE algorithm with a parameterised first-order program. Since at most $2^{|K(\mathcal{I})|^{O(1)}}$ iterations are necessary, it follows that the first-order program only needs a parameterised number of iterations. ◀

The assumptions for the proof of the second part of (a) are chosen because they are easy to state and satisfied by many natural parameterised dynamic queries. They can be relaxed though and, as an example, the result also holds for the standard change operations and the non-explicit parameter “maximal node degree” for graphs.

We now give an example of an algorithm whose underlying kernelisation can be simulated in $\text{DynFO}(+, \times)$. For a set of points in \mathbb{N}^d , for some $d \geq 2$, a *cover* is a set of lines such that each of the points is on at least one line. For a fixed dimension $d \geq 2$, the problem p - d -POINTLINECOVER (“PointLineCover”) is defined as follows:

Problem: p - d -POINTLINECOVER

Input: Distinct points $\bar{p}_1, \dots, \bar{p}_n \in \mathbb{N}^d$, **Parameter:** k

Question: Is there a cover of the points of size k ?

Each point \bar{p}_i with $i \in [n]$ is given by d coordinates p_i^1, \dots, p_i^d of n bits each. To encode these numbers, we identify the domain of size n with the set $[n]$ and use d binary relations X^1, \dots, X^d . We let $(i, j) \in X^\ell$ if the j -th bit of p_i^ℓ is 1.

A classical kernel (see e.g. [25] or [26]) for p - d -POINTLINECOVER can be obtained by realising that if a line contains at least $k+1$ points then it has to be used in a cover. Otherwise the points on this line can only be covered by using at least $k+1$ distinct lines. A kernel

for an instance can now be constructed by iteratively applying the following rule as long as possible: remove all points that belong to a simple line that contains at least $k + 1$ points and reduce k by 1. If, in the end, more than k^2 points remain, there is no line cover with k lines.

In [5] it was observed that the above reduction can be performed in parallel, since removing all points of a line removes at most one point from any other line. This immediately yields that p - d -POINTLINECOVER is in para-TC^0 , since lines with at least $k + 1$ points can be identified in TC^0 . The problem, however, is not in $\text{para-AC}^0 = \text{para-S-FO}$ [5] due to the bottleneck that collinearity of n -bit points cannot be tested in AC^0 .

We show that with an oracle for testing whether three points are collinear, a kernel of p - d -POINTLINECOVER can be actually expressed in $\text{FO}(+, \times)$. Since collinearity of three points can be maintained in $\text{DynFO}(+, \times)$ under bit changes of points, a kernel can be maintained in $\text{DynFO}(+, \times)$. Here the allowed changes are to modify single bits of the points $\bar{p}_1, \dots, \bar{p}_n$, to enable or disable a point, and to change the number k . To allow that points can be enabled or disabled, we add an additional unary relation P to structures that contains i if \bar{p}_i is part of the current instance, that is, if it is *enabled*.

► **Lemma 14.** *Collinearity of three d -dimensional points with n -bit coordinates can be maintained in $\text{DynFO}(+, \times)$ under changes of single bits, for each fixed $d \in \mathbb{N}$.*

► **Theorem 15.** *Let $\Delta \stackrel{\text{def}}{=} \Delta_{\{X_1, \dots, X_d, P\}} \cup \{\pm 1\}$.*

(a) $(p$ - d -POINTLINECOVER, Δ) \in para-S-DynFO

(b) $(p$ - d -POINTLINECOVER, Δ) \in para-T-DynFO

Proof idea. By the previous lemma, a dynamic program can maintain a relation C that contains a triple (i_1, i_2, i_3) if the points $\bar{p}_{i_1}, \bar{p}_{i_2}, \bar{p}_{i_3}$ are collinear, using Lemma 14. The statement now follows from Theorem 13 and the observation that a kernel can be defined in $\text{FO}(+, \times)$ from C .

If $k \geq \log n$, the input structure \mathcal{I} itself is a kernel of size at most $f(k)$. Otherwise, the counting abilities of $\text{FO}(+, \times)$ (see for example [14]) can be used to define a kernel. Since $k < \log n$, the set L of lines with at least $k + 1$ enabled points can be defined in $\text{FO}(+, \times)$, as well as the number $|L|$ of such lines. Additionally, the set P of enabled points that are not on any line from L is definable, and it can be determined in $\text{FO}(+, \times)$ whether there are more than k^2 of these points. Then the current kernel is defined as follows. If $|L| > k$, or $|L| \leq k$ and $|P| > k^2$, then it outputs a constant no-instance. Otherwise the kernel is the set P with the parameter $k - |L|$. ◀

5.4 Dynamic programming

Dynamic programming is a fundamental technique in algorithm design and as such it has been applied in the field of parameterised algorithms many times (e.g., [29, Section 9]). A classical parameterised algorithm with dynamic programming shows p -KNAPSACK \in FPT.

Problem: p -KNAPSACK

Input: A set of n items with profits p_1, \dots, p_n and weights w_1, \dots, w_n , a capacity bound B and a profit threshold T , **Parameter:** B

Question: Is there a subset $S \subseteq [n]$ such that $\sum_{i \in S} p_i \geq T$ and $\sum_{i \in S} w_i \leq B$?

All numbers are from \mathbb{N} and given as n -bit numbers. We choose a similar input encoding as for p - d -POINTLINECOVER in Subsection 5.3: we identify the domain of size n with the set $[n]$, encode the profits p_i using a binary relation P such that $(i, j) \in P$ if the j -th bit of

p_i is 1, and analogously encode the weights w_i and the numbers B, T by a binary relation W and unary relations B, T , respectively.⁹

► **Proposition 16.** $(p\text{-KNAPSACK}, \Delta_{\text{KS}}) \in \text{para-S-DynFO}$.

Here, Δ_{KS} denotes the set of changes that can arbitrarily replace the profit and the weight of one item, and set a number B or T to any value.

Proof sketch. The program combines the usual static algorithm with an idea that was used to capture regular languages in DynFO [22]. Intuitively, it maintains a three-dimensional table A such that $A(i, j, b)$ gives the maximum profit one can achieve by picking items with overall weight exactly b from $\{i, \dots, j\}$. This table is encoded by a relation A_{BIT} of arity four in a straightforward manner. ◀

5.5 Iterative compression

The iterative compression method (introduced in [31], see also [29, Section 11.3]) is used to obtain fixed parameter tractable algorithms for minimisation problems which are parameterised by the solution size. It can roughly be described as follows: First, a trivial solution is computed for a very small fraction of the input instance. Afterwards, the fraction is continuously increased and each time a straightforwardly updated (but maybe too big) solution is constructed and improved (“compressed”) afterwards (if necessary), until the input instance is completed and a valid solution is constructed. We illustrate the transfer of this technique to the dynamic setting with $p\text{-VERTEXCOVER}$. First we describe intuitively, how the static algorithm described in [29, Subsection 11.3.2] can be adapted to the dynamic setting.

Let $G = (V, E)$ and $G' = (V, E')$ be two input graphs, where G' results from G by inserting one edge $e = (u, v)$. Let us assume that C_0 is an optimal vertex cover for G of size k . The set $C = C_0 \cup \{u\}$ of size $k + 1$ is trivially a vertex cover for G' , but the optimal one C' might have size k . The crucial observation is that if $C' = Z \cup Z'$ has size k , for a subset Z of C and a set Z' disjoint from C , then Z' must consist of all neighbours of vertices in $C - Z$ that are not in Z . By a combination of colour coding with an adaptation of a technique from [13] for the parameterised setting, a dynamic program with advice (for the universal colouring family) can basically try out all subsets of C for Z .

► **Proposition 17.** $(p\text{-VERTEXCOVER}, \Delta_{E \cup \pm 1}) \in \text{para-S-FO}$ by a compression-based dynamic program.

6 Conclusion

In this work we started to investigate dynamic complexity from a parameterised algorithms point of view. Besides the definition of the framework, we explored how well-known techniques from parameterised algorithms translate to our setting. Kernelisation and colour-coding worked quite well for both settings. Search-tree based techniques translated well to the setting with parameterised time and were more challenging for parameterised space. On the other hand, dynamic programming (with superpolynomial parameter values) seems better suited for parameterised space. The compression-based program for $p\text{-VERTEXCOVER}$ translates,

⁹ We note that this restricts the possible weights and profits to numbers bounded by $2^n - 1$. Larger values can be achieved by a larger domain, where additionally represented items have profit and weight 0.

in principle, also to para-T-DynFO but the handling of instances with large minimal vertex cover basically requires an additional implementation of some other method and therefore makes this approach a bit pointless. We also considered *greedy localisation* and algorithms for structures with bounded tree-width, but did not find any meaningful applications in the dynamic setting, as discussed in the full version of this paper.

Particular open questions are whether p -CLOSESTSTRING or p -FEEDBACKVERTEXSET can be maintained with parameterised space and whether para-ST-DynFO is more expressive than para-S-DynFO.

References

- 1 Josh Alman, Matthias Mnich, and Virginia Vassilevska Williams. Dynamic Parameterized Problems and Algorithms. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, volume 80 of *LIPICs*, pages 41:1–41:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPICs.ICALP.2017.41.
- 2 Noga Alon, Raphael Yuster, and Uri Zwick. Color-Coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.
- 3 Max Bannach, Christoph Stockhusen, and Till Tantau. Fast Parallel Fixed-Parameter Algorithms via Color Coding. In *10th International Symposium on Parameterized and Exact Computation, IPEC 2015*, pages 224–235. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPICs.IPEC.2015.224.
- 4 Max Bannach and Till Tantau. Computing Hitting Set Kernels by AC^0 -Circuits. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018*, volume 96 of *LIPICs*, pages 9:1–9:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.STACS.2018.9.
- 5 Max Bannach and Till Tantau. Computing Kernels in Parallel: Lower and Upper Bounds. In Christophe Paul and Michal Pilipczuk, editors, *13th International Symposium on Parameterized and Exact Computation, IPEC 2018*, pages 13:1–13:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPICs.IPEC.2018.13.
- 6 Max Bannach and Till Tantau. On the Descriptive Complexity of Color Coding. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019*, volume 126 of *LIPICs*, pages 11:1–11:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPICs.STACS.2019.11.
- 7 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On Uniformity within NC^1 . *J. Comput. Syst. Sci.*, 41(3):274–306, 1990. doi:10.1016/0022-0000(90)90022-D.
- 8 Hans-Joachim Böckenhauer, Elisabet Burjons, Martin Raszyk, and Peter Rossmanith. Re-optimization of Parameterized Problems, 2018. arXiv:1809.10578.
- 9 Marco Cesati and Miriam Di Ianni. Parameterized Parallel Complexity. In David J. Pritchard and Jeff Reeve, editors, *Euro-Par '98 Parallel Processing, 4th International Euro-Par Conference, Proceedings*, volume 1470 of *Lecture Notes in Computer Science*, pages 892–896. Springer, 1998. doi:10.1007/BFb0057945.
- 10 Yijia Chen and Jörg Flum. Some Lower Bounds in Parameterized AC^0 . In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science, MFCS 2016*, volume 58 of *LIPICs*, pages 27:1–27:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPICs.MFCS.2016.27.
- 11 Yijia Chen, Jörg Flum, and Xuanguai Huang. Slicewise Definability in First-Order Logic with Bounded Quantifier Rank. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic, CSL 2017*, volume 82 of *LIPICs*, pages 19:1–19:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPICs.CSL.2017.19.
- 12 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.

- 13 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.
- 14 Larry Denenberg, Yuri Gurevich, and Saharon Shelah. Definability by Constant-Depth Polynomial-Size Circuits. *Information and Control*, 70(2/3):216–240, 1986. doi:10.1016/S0019-9958(86)80006-7.
- 15 Guozhu Dong, Jianwen Su, and Rodney W. Topor. Nonrecursive Incremental Evaluation of Datalog Queries. *Ann. Math. Artif. Intell.*, 14(2-4):187–223, 1995. doi:10.1007/BF01530820.
- 16 Rodney G. Downey, Judith Egan, Michael R Fellows, Frances A Rosamond, and Peter Shaw. Dynamic Dominating set and Turbo-Charging Greedy Heuristics. *Tsinghua Science and Technology*, 19(4):329–337, 2014. doi:10.1109/TST.2014.6867515.
- 17 Rodney G. Downey and Michael R. Fellows. Fixed-Parameter Tractability and Completeness I: Basic Results. *SIAM J. Comput.*, 24(4):873–921, 1995. doi:10.1137/S0097539792228228.
- 18 Rodney G. Downey and Michael R. Fellows. Parameterized Computational Feasibility. In *Feasible mathematics II*, pages 219–244. Springer, 1995. doi:10.1007/978-1-4612-2566-9_7.
- 19 Michael Elberfeld, Christoph Stockhusen, and Till Tantau. On the Space and Circuit Complexity of Parameterized Problems: Classes and Completeness. *Algorithmica*, 71(3):661–701, 2015. doi:10.1007/s00453-014-9944-y.
- 20 Jörg Flum and Martin Grohe. Describing Parameterized Complexity Classes. *Inf. Comput.*, 187(2):291–319, 2003. doi:10.1016/S0890-5401(03)00161-5.
- 21 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, Circuits, and the Polynomial-Time Hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984. doi:10.1007/BF01744431.
- 22 Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *ACM Trans. Comput. Log.*, 13(3):19, 2012. doi:10.1145/2287718.2287719.
- 23 Sepp Hartung and Rolf Niedermeier. Incremental List Coloring of Graphs, Parameterized by Conservation. *Theor. Comput. Sci.*, 494:86–98, 2013. doi:10.1016/j.tcs.2012.12.049.
- 24 Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.
- 25 Stefan Kratsch, Geevarghese Philip, and Saurabh Ray. Point Line Cover: The Easy Kernel is Essentially Tight. *ACM Trans. Algorithms*, 12(3):40:1–40:16, 2016. doi:10.1145/2832912.
- 26 Stefan Langerman and Pat Morin. Covering Things with Things. *Discrete & Computational Geometry*, 33(4):717–729, 2005. doi:10.1007/s00454-004-1108-4.
- 27 Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004. doi:10.1007/978-3-662-07003-1.
- 28 Bernard Mans and Luke Mathieson. Incremental Problems in the Parameterized Complexity Setting. *Theory Comput. Syst.*, 60(1):3–19, 2017. doi:10.1007/s00224-016-9729-6.
- 29 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Number 31 in Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, 2006. doi:10.1093/acprof:oso/9780198566076.001.0001.
- 30 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.
- 31 Bruce A. Reed, Kaleigh Smith, and Adrian Vetta. Finding Odd Cycle Transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004. doi:10.1016/j.orl.2003.10.009.
- 32 Thomas Schwentick and Thomas Zeume. Dynamic Complexity: Recent Updates. *SIGLOG News*, 3(2):30–52, 2016. doi:10.1145/2948896.2948899.

Dynamic Complexity of Parity Exists Queries

Nils Vortmeier

TU Dortmund University, Dortmund, Germany
nils.vortmeier@tu-dortmund.de

Thomas Zeume

TU Dortmund University, Dortmund, Germany
thomas.zeume@tu-dortmund.de

Abstract

Given a graph whose nodes may be coloured red, the parity of the number of red nodes can easily be maintained with first-order update rules in the dynamic complexity framework DynFO of Patnaik and Immerman. Can this be generalised to other or even all queries that are definable in first-order logic extended by parity quantifiers? We consider the query that asks whether the number of nodes that have an edge to a red node is odd. Already this simple query of quantifier structure parity-exists is a major roadblock for dynamically capturing extensions of first-order logic.

We show that this query cannot be maintained with quantifier-free first-order update rules, and that variants induce a hierarchy for such update rules with respect to the arity of the maintained auxiliary relations. Towards maintaining the query with full first-order update rules, it is shown that degree-restricted variants can be maintained.

2012 ACM Subject Classification Theory of computation → Logic and databases; Theory of computation → Complexity theory and logic

Keywords and phrases Dynamic complexity, parity quantifier, arity hierarchy

Digital Object Identifier 10.4230/LIPIcs.CSL.2020.37

Related Version A full version of this paper is available at <http://arxiv.org/abs/1910.06004>.

Funding The authors acknowledge the financial support by DFG grant SCHW 678/6-2.

Acknowledgements We are grateful to Samir Datta, Raghav Kulkarni, Anish Mukherjee and Thomas Schwentick for illuminating discussions.

1 Introduction

The query PARITY – given a unary relation U , does U contain an odd number of elements? – cannot be *expressed* in first-order logic, even with arbitrary numerical built-in relations [2, 9]. However, it can easily be *maintained* in a dynamic scenario where single elements can be inserted into and removed from U , and helpful information for answering the query is stored and updated by first-order definable update rules upon changes. Whenever a new element is inserted into or an existing element is removed from U , then a stored bit P is flipped¹. In the dynamic complexity framework by Patnaik and Immerman [13] this can be expressed by the following first-order update rules:

on insert a into U **update** P as $(\neg U(a) \wedge \neg P) \vee (U(a) \wedge P)$

on delete a from U **update** P as $(U(a) \wedge \neg P) \vee (\neg U(a) \wedge P)$

¹ This bit is preserved if a change re-inserts an element that already is in U , or tries to delete an element that is not in U .



This simple program proves that PARITY is in the dynamic complexity class DynFO which contains all queries that can be maintained via first-order formulas that use (and update) some additional stored auxiliary relations.

Motivated by applications in database theory and complexity theory, the class DynFO has been studied extensively in the last three decades. In database theory it is well-known that first-order logic corresponds to the relational core of SQL (see, e.g., [1]). Thus, if a query can be maintained with first-order update rules then, in particular, it can be updated using SQL queries. From a complexity theoretic point of view, first-order logic with built-in arithmetic corresponds to the circuit complexity class uniform AC^0 [3]. Hence queries in DynFO can be evaluated in a highly parallel fashion in dynamic scenarios.

The focus of research on DynFO has been its expressive power. The parity query is a first witness that DynFO is more expressive than FO (the class of queries expressible by first-order formulas in the standard, non-dynamic setting), but it is not the only witness. Further examples include the reachability query for general directed graphs [4], another textbook query that is not in FO but complete for the complexity class NL, which can be characterised (on ordered structures) by the extension of first-order logic with a transitive closure operator. On (classes of) graphs of bounded treewidth, DynFO includes all queries that can be defined in monadic second-order logic [5], which extends first-order logic by quantification over sets. In particular, on strings DynFO also contains all MSO-definable Boolean queries, that is, all regular languages. Actually for strings the update rules do not need any quantifiers [10] proving that regular languages are even in the dynamic complexity class DynProp which is defined via quantifier-free first-order update rules.

These examples show that dynamically first-order logic can, in some cases, sidestep quantifiers and operators which it cannot express statically: parity and set quantifiers, as well as transitive closure operators. Immediately the question arises whether first-order update rules can dynamically maintain *all* queries that are statically expressible in extensions of first-order logic by one of these quantifiers or operators. Note that this does not follow easily, for instance, from the result that the NL-complete reachability query is in DynFO, because the notions of reductions that are available in the dynamic setting are too weak [13].

The extension FO+Parity of first-order logic by parity quantifiers is the natural starting point for a more thorough investigation of how DynFO relates to extensions of FO, as it is arguably the simplest natural extension that extends the expressive power. Unfortunately, however, a result of the form $FO+Parity \subseteq DynFO$ is not in sight². While PARITY is in DynFO, already for slightly more complex queries expressible in FO+Parity it seems not to be easy to show that they are in DynFO. In this paper we are particularly interested in the following generalisation of the parity query:

PARITYEXISTS: Given a graph whose nodes may be coloured red. Is the number of nodes connected to a red node odd? Edges can be inserted and deleted; nodes can be coloured or uncoloured.

As it is still unknown whether PARITYEXISTS is in DynFO, this query is a roadblock for showing that DynFO captures (large subclasses of) FO+Parity. For this reason we study the dynamic complexity of PARITYEXISTS. We focus on the following two directions: (1) its relation to the well-understood quantifier-free fragment DynProp of DynFO, and (2) the dynamic complexity of degree-restricted variants.

² Formally one has to be a little more precise. For technical reasons, one cannot express the query “The size of the domain is even.” in DynFO. Therefore we are interested in results of this form for domain-independent queries, that is, queries whose result does not change when isolated elements are added to the domain.

The update rules given above witness that PARITY is in DynProp . We show that this is not the case any more for PARITYEXISTS .

► **Theorem 1.** $\text{PARITYEXISTS} \notin \text{DynProp}$.

A fine-grained analysis of the quantifier-free complexity is the main contribution of this paper, which also implies Theorem 1. Let $\text{PARITYEXISTS}_{\text{deg} \leq k}$ be the variant of the PARITYEXISTS query that asks whether the number of nodes that have both an edge to a red node and degree at most k is odd, for some fixed number $k \in \mathbb{N}$.

► **Theorem 2.** $\text{PARITYEXISTS}_{\text{deg} \leq k}$ can be maintained in DynProp with auxiliary relations of arity k , but not with auxiliary relations of arity $k - 1$, for any $k \geq 3$.

This result actually has an impact beyond the lower bound given by Theorem 1. It clarifies the structure of DynProp , as it shows that auxiliary relations with higher arities increase the expressive power of quantifier-free update formulas.

Already Dong and Su showed that DynFO has an arity hierarchy [6], i.e., that for each $k \in \mathbb{N}$ there is a query q_k that can be maintained using first-order update rules and k -ary auxiliary relations, but not using $(k - 1)$ -ary auxiliary relations. The query q_k from [6] is a k -ary query q_k that is evaluated over a $(6k + 1)$ -ary relation T and returns all k -ary tuples \bar{a} such that the number of $(5k + 1)$ -ary tuples \bar{b} with $(\bar{a}, \bar{b}) \in T$ is divisible by 4. Dong and Su ask whether the arity of the relation T can be reduced to $3k$, k , or even to 2. Their question for reducing it below $3k$ was motivated by a known reduction of the arity to $3k + 1$ [7].

An arity hierarchy for DynProp follows because the query q_k from [6] can be maintained with quantifier-free update rules, though again only for input relations whose arity depends on k . Some progress towards an arity hierarchy for Boolean graph queries was made in [17], where the arities up to $k = 3$ were separated for such queries. If only insertions are allowed, then DynProp is known to have an arity hierarchy for Boolean graph queries [16].

An arity hierarchy for quantifier-free update rules and Boolean graph properties is now an immediate consequence of Theorem 2, in connection with the results for $k \leq 3$ from [17].

► **Corollary 3.** DynProp has a strict arity hierarchy for Boolean graph queries.

Such an arity hierarchy does *not* exist for DynProp when we consider not graphs as inputs but strings. Gelade et al. show that the class of Boolean queries on strings that are in DynProp are exactly the regular languages, and that every such language can be maintained with binary auxiliary relations [10]. So, relations of higher arity are never necessary in this case.

With respect to DynFO , we cannot answer the question whether $\text{PARITYEXISTS} \in \text{DynFO}$, but we can generalise the result of Theorem 2 to restrictions beyond fixed numbers k , at least if the update formulas have access to additional built-in relations. Let $\text{PARITYEXISTS}_{\text{deg} \leq \log n}$ be the query that asks for the parity of the number of nodes that are connected to a red node and have degree at most $\log n$, where n is the number of nodes of the graph. The binary BIT predicate essentially gives the bit encoding of natural numbers.

► **Theorem 4.** $\text{PARITYEXISTS}_{\text{deg} \leq \log n}$ can be maintained in DynFO with binary auxiliary relations in the presence of a linear order and BIT .

In particular, the queries $\text{PARITYEXISTS}_{\text{deg} \leq k}$, for $k \in \mathbb{N}$, do not induce an arity hierarchy for DynFO . For fixed k , essentially already unary auxiliary relations suffice.

► **Theorem 5.** $\text{PARITYEXISTS}_{\text{deg} \leq k}$ can be maintained in DynFO with unary auxiliary relations in the presence of a linear order, for every $k \in \mathbb{N}$.

In both results, Theorem 4 and 5, the assumption on the presence of a built-in linear order and the BIT predicate can be lifted when the degree bound of $\text{PARITYEXISTS}_{\text{deg} \leq \log n}$ refers to the active domain instead of the whole domain; see Section 4 for a discussion.

Finally, we complement our results by a discussion of how queries expressible in FO extended by arbitrary modulo quantifiers can be maintained in an extension of DynFO. This observation is based on discussions with Samir Datta, Raghav Kulkarni, and Anish Mukherjee.

Outline. After recalling the dynamic descriptive complexity scenario in Section 2, we prove Theorem 2 in Section 3, followed by Theorem 4 and Theorem 5 in Section 4. We conclude in Section 5.

2 Preliminaries: A short introduction to dynamic complexity

We shortly recapitulate the dynamic complexity framework as introduced by Patnaik and Immerman [13], and refer to [15] for details.

In this framework, a (relational, finite) structure \mathcal{I} over some schema σ_{in} can be changed by inserting a tuple into or removing a tuple from a relation of \mathcal{I} . A *change* $\alpha = \delta(\bar{a})$ consists of an (abstract) *change operation* δ , which is either INS_R or DEL_R for a relation symbol $R \in \sigma_{\text{in}}$, and a tuple \bar{a} over the domain of \mathcal{I} . The change $\text{INS}_R(\bar{a})$ inserts \bar{a} into the relation R of \mathcal{I} , and $\text{DEL}_R(\bar{a})$ deletes \bar{a} from that relation. We denote by $\alpha(\mathcal{I})$ the structure that results from applying a change α to the structure \mathcal{I} .

A *dynamic program* \mathcal{P} stores an input structure \mathcal{I} as well as an auxiliary structure \mathcal{A} over some auxiliary schema σ_{aux} . For each change operation δ and each auxiliary relation $S \in \sigma_{\text{aux}}$, the dynamic program has a first-order update rule that specifies how S is updated after a change. Each such rule is of the form **on change** $\delta(\bar{p})$ **update** $S(\bar{x})$ **as** $\varphi_\delta^S(\bar{p}, \bar{x})$ where the *update formula* φ_δ^S is over the combined schema $\sigma_{\text{in}} \cup \sigma_{\text{aux}}$ of \mathcal{I} and \mathcal{A} . Now, for instance, if a tuple \bar{a} is inserted into an input relation R , the auxiliary relation S is updated to $\{\bar{b} \mid (\mathcal{I}, \mathcal{A}) \models \varphi_{\text{INS}_R}^S(\bar{a}, \bar{b})\}$. In the standard scenario, all relations in both \mathcal{I} and \mathcal{A} are empty initially.

A *k-ary query* q on σ -structures, for some schema σ , maps each σ -structure with some domain D to a subset of D^k , and commutes with isomorphism. A query q is *maintained* by \mathcal{P} if \mathcal{A} has one distinguished relation ANS which, after each sequence of changes, contains the result of q for the current input structure \mathcal{I} .

The class DynFO contains all queries that can be maintained by first-order update rules. The class DynProp likewise contains the queries that can be maintained by quantifier-free update rules. We say that a query q is in *k-ary DynFO* (DynProp), for some number $k \in \mathbb{N}$, if it is in DynFO (DynProp) via a dynamic program that uses at most k -ary auxiliary relations.

Sometimes we allow the update formulas to access built-in relations, as for example a predefined linear order \leq and the BIT predicate. We then assume that the input provides a linear order \leq , which allows to identify the domain with a prefix of the natural numbers, and a binary relation BIT that contains a tuple (i, j) if the j -th bit in the binary representation of i is 1. Both relations cannot be changed.

For expressibility results we will use the standard scenario from [13] that uses initial input and auxiliary structures with empty relations. Our inexpressibility results are stated for the more powerful scenario where the auxiliary structure is initialised arbitrarily. See also [17] for a discussion of these different scenarios.

Already quantifier-free programs are surprisingly expressive, as they can maintain, for instance, all regular languages [10] and the transitive closure of deterministic graphs [11]. As we have seen in the introduction, also the query PARITY can be maintained by quantifier-free update rules. The following example illustrates a standard technique for maintaining queries with quantifier-free update rules which will also be exploited later.

► **Example 6.** For fixed $k \in \mathbb{N}$ let IN-DEG- k be the unary query that, given a graph G , returns the set of nodes with in-degree k . This query is easily definable in FO for each k . We show here that IN-DEG- k can be maintained by a DynProp-program \mathcal{P} .

The dynamic program we construct uses k -lists, a slight extension of the list technique introduced in [10]. The list technique was used in [17] to maintain emptiness of a unary relation U under insertions and deletions of single elements with quantifier-free formulas. To this end a binary relation LIST which encodes a linked list of the elements in U in the order of their insertion is maintained. Additionally, two unary relations mark the first and the last element of the list. The key insight is that a quantifier-free formula can figure out whether the relation U becomes empty when an element a is deleted by checking whether a is both the first *and* the last element of the list.

To maintain IN-DEG- k the quantifier-free dynamic program \mathcal{P} stores, for every node $v \in V$, a list of all nodes u with $(u, v) \in E$, using a ternary relation LIST₁. More precisely, if u_1, \dots, u_m are the in-neighbours of v then LIST₁ contains the tuples $(v, u_{i_j}, u_{i_{j+1}})$ where j_1, \dots, j_m is some permutation of $\{1, \dots, m\}$. Additionally, the program uses ternary relations LIST₂, \dots , LIST _{k} such that LIST _{i} describes paths of length i in the linked list LIST₁. For example, if $(v, u_1, u_2), (v, u_2, u_3)$ and (v, u_3, u_4) are tuples in LIST₁, then $(v, u_1, u_4) \in$ LIST₃. The list LIST₁ comes with $2k$ binary relations FIRST₁, \dots , FIRST _{k} , LAST₁, \dots , LAST _{k} that mark, for each $v \in V$, the first and the last k elements of the list of in-neighbours of v , as well as with $k + 2$ unary relations IS₀, \dots , IS _{k} , IS_{> k} that count the number of in-neighbours for each $v \in V$ up to k . We call nodes u with $(v, u) \in$ FIRST _{i} or $(v, u) \in$ LAST _{i} the i -first or the i -last element for v , respectively.

Using these relations, the query can be answered easily: the result is the set of nodes v with $v \in$ IS _{k} . We show how to maintain the auxiliary relations under insertions and deletions of single edges, and assume for ease of presentation of the update formulas that if a change INS _{E} (u, v) occurs then $(u, v) \notin E$ before the change, and a change DEL _{E} (u, v) only happens if $(u, v) \in E$ before the change.

Insertions of edges. When an edge (u, v) is inserted, then the node u needs to be inserted into the list of v . This node u also becomes the last element of the list (encoded by a tuple $(v, u) \in$ LAST₁), and the i -last node u' for v becomes the $(i + 1)$ -last one, for $i < k$. If only i elements are in the list for v before the change, u becomes the $(i + 1)$ -first element for v . The update formulas are as follows:

$$\begin{aligned}
\varphi_{\text{INS}_E}^{\text{LIST}_i}(u, v; x, y, z) &\stackrel{\text{def}}{=} \text{LIST}_i(x, y, z) \vee (v = x \wedge \text{LAST}_i(x, y) \wedge u = z) && \text{for } i \in \{1, \dots, k\} \\
\varphi_{\text{INS}_E}^{\text{LAST}_1}(u, v; x, y) &\stackrel{\text{def}}{=} (v \neq x \wedge \text{LAST}_1(x, y)) \vee (v = x \wedge u = y) \\
\varphi_{\text{INS}_E}^{\text{LAST}_i}(u, v; x, y) &\stackrel{\text{def}}{=} (v \neq x \wedge \text{LAST}_i(x, y)) \vee (v = x \wedge \text{LAST}_{i-1}(y)) && \text{for } i \in \{2, \dots, k\} \\
\varphi_{\text{INS}_E}^{\text{FIRST}_i}(u, v; x, y) &\stackrel{\text{def}}{=} \text{FIRST}_i(x, y) \vee (v = x \wedge u = y \wedge \text{IS}_{i-1}(x)) && \text{for } i \in \{1, \dots, k\} \\
\varphi_{\text{INS}_E}^{\text{IS}_0}(u, v; x) &\stackrel{\text{def}}{=} (v \neq x \wedge \text{IS}_0(x)) \\
\varphi_{\text{INS}_E}^{\text{IS}_i}(u, v; x) &\stackrel{\text{def}}{=} (v \neq x \wedge \text{IS}_i(x)) \vee (v = x \wedge \text{IS}_{i-1}(x)) && \text{for } i \in \{1, \dots, k\} \\
\varphi_{\text{INS}_E}^{\text{IS}_{>k}}(u, v; x) &\stackrel{\text{def}}{=} \text{IS}_{>k}(x) \vee (v = x \wedge \text{IS}_k(x))
\end{aligned}$$

Deletions of edges. When an edge (u, v) is deleted, the hardest task for quantifier-free update formulas is to determine whether, if the in-degree of v was *at least* $k + 1$ before the change, the in-degree of v is now *exactly* k . We use that if an element u is the j -first and at the same time the j' -last element for v , then the list for v contains exactly $j + j' - 1$ elements. If u is removed from the list, $j + j' - 2$ elements remain. So, using the relations FIRST_j and $\text{LAST}_{j'}$, the exact number m of elements after the change can be determined, if $m \leq 2k - 2$. The relations FIRST_i (and, symmetrically the relations LAST_i) can be maintained using the relations LIST_j : if the i' -first element u is removed from the list for v , u' becomes the i -first element for $i' \leq i \leq k$ if $(v, u, u') \in \text{LIST}_{i-i'+1}$. The update formulas exploit these insights:

$$\begin{aligned}
\varphi_{\text{DEL}_E}^{\text{LIST}_i}(u, v; x, y, z) &\stackrel{\text{def}}{=} (v \neq x) \wedge \text{LIST}_i(x, y, z) \\
&\quad \vee (v = x \wedge u \neq y \wedge \bigwedge_{i' \leq i} \neg \text{LIST}_{i'}(x, y, u) \wedge \text{LIST}_i(x, y, z)) \\
&\quad \vee (v = x \wedge \bigvee_{\substack{j, j' \\ j+j'=i+1}} \text{LIST}_j(x, y, u) \wedge \text{LIST}_{j'}(x, u, z)) && \text{for } i \in \{1, \dots, k\} \\
\varphi_{\text{DEL}_E}^{\text{LAST}_i}(u, v; x, y) &\stackrel{\text{def}}{=} (v \neq x \wedge \text{LAST}_i(x, y)) \\
&\quad \vee (v = x \wedge \bigwedge_{i' \leq i} \neg \text{LAST}_{i'}(u) \wedge \text{LAST}_i(y)) \\
&\quad \vee (v = x \wedge \bigvee_{i' \leq i} (\text{LAST}_{i'}(u) \wedge \text{LIST}_{i-i'+1}(x, y, u))) && \text{for } i \in \{1, \dots, k\} \\
\varphi_{\text{DEL}_E}^{\text{FIRST}_i}(u, v; x, y) &\stackrel{\text{def}}{=} (v \neq x \wedge \text{FIRST}_i(x, y)) \\
&\quad \vee (v = x \wedge \bigwedge_{i' \leq i} \neg \text{FIRST}_{i'}(u) \wedge \text{FIRST}_i(y)) \\
&\quad \vee (v = x \wedge \bigvee_{i' \leq i} (\text{FIRST}_{i'}(u) \wedge \text{LIST}_{i-i'+1}(x, u, y))) && \text{for } i \in \{1, \dots, k\} \\
\varphi_{\text{DEL}_E}^{\text{IS}_i}(u, v; x) &\stackrel{\text{def}}{=} (v \neq x \wedge \text{IS}_i(x)) \\
&\quad \vee (v = x \wedge \bigvee_{\substack{j, j' \\ j+j'-2=i}} \text{FIRST}_j(x, u) \wedge \text{LAST}_{j'}(x, u)) && \text{for } i \in \{0, \dots, k\} \\
\varphi_{\text{DEL}_E}^{\text{IS}_{>k}}(u, v; x) &\stackrel{\text{def}}{=} (v \neq x \wedge \text{IS}_{>k}(x)) \\
&\quad \vee (v = x \wedge \text{IS}_{>k}(x) \wedge \bigwedge_{\substack{j, j' \\ j+j'-2=k}} (\neg \text{FIRST}_j(x, u) \vee \neg \text{LAST}_{j'}(x, u)))
\end{aligned}$$

3 ParityExists and quantifier-free updates

In this section we start our examination of the PARITYEXISTS query in the context of quantifier-free update rules. Let us first formalize the query. It is evaluated over *coloured graphs*, that is, directed graphs (V, E) with an additional unary relation R that encodes a set of (red-)coloured nodes.³ A node w of such a graph is said to be *covered* if there is a coloured node $v \in R$ with $(v, w) \in E$. The query PARITYEXISTS asks, given a coloured graph, whether the number of covered nodes is odd.

³ We note that the additional relation R is for convenience of exposition. All our results are also valid for pure graphs: instead of using the relation R one could consider a node v coloured if it has a self-loop $(v, v) \in E$.

As stated in the introduction, PARITYEXISTS cannot be maintained with quantifier-free update rules. A closer examination reveals a close connection between a variant of this query and the arity structure of DynProp . Let k be a natural number. The variant $\text{PARITYEXISTS}_{\text{deg} \leq k}$ of PARITYEXISTS asks whether the number of covered nodes that additionally have in-degree at most k is odd. Note that $\text{PARITYEXISTS}_{\text{deg} \leq k}$ is a query on general coloured graphs, not only on graphs with bounded degree.

► **Theorem 2.** $\text{PARITYEXISTS}_{\text{deg} \leq k}$ can be maintained in DynProp with auxiliary relations of arity k , but not with auxiliary relations of arity $k - 1$, for any $k \geq 3$.

We repeat two immediate consequences which have already been stated in the introduction.

► **Theorem 1.** $\text{PARITYEXISTS} \notin \text{DynProp}$.

► **Corollary 3.** DynProp has a strict arity hierarchy for Boolean graph queries.

Proof. For every $k \geq 1$ we give a Boolean graph query that can be maintained using k -ary auxiliary relations, but not with $(k - 1)$ -ary relations.

For $k \geq 3$, we choose the query $\text{PARITYEXISTS}_{\text{deg} \leq k}$ which satisfies the conditions by Theorem 2.

For $k = 2$, already [17, Proposition 4.10] shows that the query S-T-TWOPATH which asks whether there exists a path of length 2 between two distinguished vertices s and t separates unary DynProp from binary DynProp .

For $k = 1$, we consider the Boolean graph query PARITYDEGREEDIV3 that asks whether the number of nodes whose degree is divisible by 3 is odd. This query can easily be maintained in DynProp using only unary auxiliary relations. In a nutshell, a dynamic program can maintain for each node v the degree of v modulo 3. So, it maintains three unary relations M_0, M_1, M_2 with the intention that $v \in M_i$ if the degree of v is congruent to i modulo 3. These relations can easily be updated under edge insertions and deletions. Similar as for PARITY , a bit P that gives the parity of $|M_0|$ can easily be maintained.

On the other hand, PARITYDEGREEDIV3 cannot be maintained in DynProp using nullary auxiliary relations. Suppose, towards a contradiction, that it can be maintained by some dynamic program \mathcal{P} that only uses nullary auxiliary relations, and consider an input instance that contains five node $V = \{u_1, u_2, v_1, v_2, v_3\}$ as well as edges $E = \{(u_1, v_1), (u_1, v_2), (u_2, v_1)\}$. No matter the auxiliary database, \mathcal{P} needs to give the same answer after the changes $\alpha_1 \stackrel{\text{def}}{=} \text{INS}_E(u_1, v_3)$ and $\alpha_2 \stackrel{\text{def}}{=} \text{INS}_E(u_2, v_3)$, as it cannot distinguish these tuples using quantifier-free first-order formulas. But α_1 leads to a yes-instance for PARITYDEGREEDIV3 , and α_2 does not. So, \mathcal{P} does not maintain PARITYDEGREEDIV3 . ◀

The rest of this section is devoted to the proof of Theorem 2. First, in Subsection 3.1, we show that $\text{PARITYEXISTS}_{\text{deg} \leq k}$ can be maintained with k -ary auxiliary relations, for $k \geq 3$. Here we employ the list technique introduced in Example 6. Afterwards, in Subsection 3.2, we prove that auxiliary relations of arity $k - 1$ do not suffice. This proof relies on a known tool for proving lower bounds for DynProp that exploits upper and lower bounds for Ramsey numbers [16].

3.1 Maintaining $\text{ParityExists}_{\text{deg} \leq k}$

We start by proving that $\text{PARITYEXISTS}_{\text{deg} \leq k}$ can be maintained in DynProp using k -ary auxiliary relations. In Subsection 3.2 we show that this arity is optimal.

► **Proposition 7.** For every $k \geq 3$, $\text{PARITYEXISTS}_{\text{deg} \leq k}$ is in k -ary DynProp .

In the following proof, we write $[n]$ for the set $\{1, \dots, n\}$ of natural numbers.

Proof. Let $k \geq 3$ be some fixed natural number. We show how a DynProp-program \mathcal{P} can maintain $\text{PARITYEXISTS}_{\text{deg} \leq k}$ using at most k -ary auxiliary relations.

The idea is as follows. Whenever a formerly uncoloured node v gets coloured, a certain number $c(v)$ of nodes become covered: v has edges to all these nodes, but no other coloured node has. Because the number $c(v)$ can be arbitrary, the program \mathcal{P} necessarily has to store for each uncoloured node v the parity of $c(v)$ to update the query result. But this is not sufficient. Suppose that another node v' is coloured by a change and that, as a result, a number $c(v')$ of nodes become covered, because they have an edge from v' and so far no incoming edge from another coloured neighbour. Some of these nodes, say, $c(v, v')$ many, also have an incoming edge from v . Of course these nodes do not *become* covered any more when afterwards v is coloured, because they *are* already covered. So, whenever a node v' gets coloured, the program \mathcal{P} needs to update the (parity of the) number $c(v)$, based on the (parity of the) number $c(v, v')$. In turn, the (parity of the) latter number needs to be updated whenever another node v'' is coloured, using the (parity of the) analogously defined number $c(v, v', v'')$, and so on.

It seems that this reasoning does not lead to a construction idea for a dynamic program, as information for more and more nodes needs to be stored, but observe that only those covered nodes are relevant for the query that have in-degree at most k . So, a number $c(v_1, \dots, v_k)$ does not need to be updated when some other node v_{k+1} gets coloured, because no relevant node has edges from all nodes v_1, \dots, v_{k+1} .

We now present the construction in more detail. A node w is called *active* if its in-degree $\text{in-deg}(w)$ is at most k . Let $A = \{a_1, \dots, a_\ell\}$ be a set of coloured nodes and let $B = \{b_1, \dots, b_m\}$ be a set of uncoloured nodes, with $\ell + m \leq k$. By $\mathcal{N}_G^{\bullet\circ}(A, B)$ we denote the set of active nodes w of the coloured graph G whose coloured (in-)neighbours are exactly the nodes in A and that have (possibly amongst others) the nodes in B as uncoloured (in-)neighbours. So, $w \in \mathcal{N}_G^{\bullet\circ}(A, B)$ if (1) $\text{in-deg}(w) \leq k$, (2) $(v, w) \in E$ for all $v \in A \cup B$, and (3) there is no edge $(v', w) \in E$ from a coloured node $v' \in R$ with $v' \notin A$. We omit the subscript G and just write $\mathcal{N}^{\bullet\circ}(A, B)$ if the graph G is clear from the context. The dynamic program \mathcal{P} maintains the parity of $|\mathcal{N}_G^{\bullet\circ}(A, B)|$ for all such sets A, B .

Whenever a change $\alpha = \text{INS}_R(v)$ colours a node v of G , the update is as follows. We distinguish the three cases (1) $v \in A$, (2) $v \in B$ and (3) $v \notin A \cup B$. In case (1), the set $\mathcal{N}_{\alpha(G)}^{\bullet\circ}(A, B)$ equals the set $\mathcal{N}_G^{\bullet\circ}(A \setminus \{v\}, B \cup \{v\})$, and the existing auxiliary information can be copied. In case (2), actually $\mathcal{N}_{\alpha(G)}^{\bullet\circ}(A, B) = \emptyset$, as B contains a coloured node. The parity of the cardinality 0 of \emptyset is even. For case (3) we distinguish two further cases. If $|A \cup B| = k$, no active node w can have incoming edges from every node in $A \cup B \cup \{v\}$ as w has in-degree at most k , so $\mathcal{N}_{\alpha(G)}^{\bullet\circ}(A, B) = \mathcal{N}_G^{\bullet\circ}(A, B)$ and the existing auxiliary information is taken over. If $|A \cup B| < k$, then $\mathcal{N}_{\alpha(G)}^{\bullet\circ}(A, B) = \mathcal{N}_G^{\bullet\circ}(A, B) \setminus \mathcal{N}_G^{\bullet\circ}(A, B \cup \{v\})$ and \mathcal{P} can combine the existing auxiliary information.

When a change $\alpha = \text{DEL}_R(v)$ uncolours a node v of G , the necessary updates are symmetrical. The case $v \in A$ is similar to case (2) above: $\mathcal{N}_{\alpha(G)}^{\bullet\circ}(A, B) = \emptyset$, because A contains an uncoloured node. The case $v \in B$ is handled similarly as case (1) above, as we have $\mathcal{N}_{\alpha(G)}^{\bullet\circ}(A, B) = \mathcal{N}_G^{\bullet\circ}(A \cup \{v\}, B \setminus \{v\})$. The third case $v \notin A \cup B$ is treated analogously as case (3) above, but in the sub-case $|A \cup B| < k$ we have that $\mathcal{N}_{\alpha(G)}^{\bullet\circ}(A, B) = \mathcal{N}_G^{\bullet\circ}(A, B) \cup \mathcal{N}_G^{\bullet\circ}(A \cup \{v\}, B)$.

Edge insertions and deletions are conceptionally easy to handle, as they change the sets $\mathcal{N}^{\bullet\circ}(A, B)$ by at most one element. Given all nodes of A and B and the endpoints of the changed edge as parameters, quantifier-free formulas can easily determine whether this is the case for specific sets A, B .

We now present \mathcal{P} formally. For every $\ell \leq k+1$ the program maintains unary relations N_ℓ and N_ℓ^\bullet with the indented meaning that for a node w it holds $w \in N_\ell$ if $\text{in-deg}(w) = \ell$ and $w \in N_\ell^\bullet$ if w has exactly ℓ coloured in-neighbours. These relations can be maintained as presented in Example 6, requiring some additional, ternary auxiliary relations. We also use a relation $\text{ACTIVE} \stackrel{\text{def}}{=} N_1 \cup \dots \cup N_k$ that contains all active nodes with at least one edge.

For every $\ell, m \geq 0$ with $1 \leq \ell + m \leq k$ the programs maintains $(\ell + m)$ -ary auxiliary relations $P_{\ell,m}$ with the intended meaning that a tuple $(a_1, \dots, a_\ell, b_1, \dots, b_m)$ is contained in $P_{\ell,m}$ if and only if

- the nodes $a_1, \dots, a_\ell, b_1, \dots, b_m$ are pairwise distinct,
- $a_i \in R$ and $b_j \notin R$ for $i \in [\ell], j \in [m]$, and
- the set $\mathcal{N}^{\bullet\circ}(A, B)$ has an odd number of elements, where $A = \{a_1, \dots, a_\ell\}$ and $B = \{b_1, \dots, b_m\}$.

The following formula $\theta_{\ell,m}$ checks the first two conditions:

$$\theta_{\ell,m}(x_1, \dots, x_\ell, y_1, \dots, y_m) \stackrel{\text{def}}{=} \bigwedge_{i \neq j \in [\ell]} x_i \neq x_j \wedge \bigwedge_{i \neq j \in [m]} y_i \neq y_j \wedge \bigwedge_{i \in [\ell]} R(x_i) \wedge \bigwedge_{i \in [m]} \neg R(y_i)$$

Of course, \mathcal{P} also maintains the Boolean query relation ANS .

We now describe the update formulas of \mathcal{P} for the relations $P_{\ell,m}$ and ANS , assuming that each change actually alters the input graph, so, for example, no changes $\text{INS}_E(v, w)$ occur such that the edge (v, w) already exists.

Let $\varphi \oplus \psi \stackrel{\text{def}}{=} (\varphi \wedge \neg \psi) \vee (\neg \varphi \wedge \psi)$ denote the Boolean exclusive-or connector.

Colouring a node v . A change $\text{INS}_R(v)$ increases the total number of active, covered nodes by the number of active nodes that have so far no coloured in-neighbour, but an edge from v . That is, this number is increased by $|\mathcal{N}^{\bullet\circ}(\emptyset, \{v\})|$. The update formula for ANS is therefore

$$\varphi_{\text{INS}_R}^{\text{ANS}}(v) \stackrel{\text{def}}{=} \text{ANS} \oplus P_{0,1}(v).$$

We only spell out the more interesting update formulas for the relations $P_{\ell,m}$, for different values of ℓ, m . These formulas list the conditions for tuples $\bar{a} = a_1, \dots, a_\ell$ and $\bar{b} = b_1, \dots, b_m$ that $\mathcal{N}^{\bullet\circ}(\{\bar{a}\}, \{\bar{b}\})$ is of odd size after a change. The other update formulas are simple variants.

$$\begin{aligned} \varphi_{\text{INS}_R}^{P_{\ell,m}}(v; x_1, \dots, x_\ell, y_1, \dots, y_m) &\stackrel{\text{def}}{=} \\ &\bigvee_{i \in [\ell]} (v = x_i \wedge P_{\ell-1, m+1}(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_\ell, \bar{y}, v)) \\ &\vee \left(\bigwedge_{i \in [\ell]} v \neq x_i \wedge \bigwedge_{i \in [m]} v \neq y_i \wedge (P_{\ell,m}(\bar{x}, \bar{y}) \oplus P_{\ell, m+1}(\bar{x}, \bar{y}, v)) \right) \quad \text{for } \ell \geq 1, \ell + m < k \\ \varphi_{\text{INS}_R}^{P_{\ell,m}}(v; x_1, \dots, x_\ell, y_1, \dots, y_m) &\stackrel{\text{def}}{=} \\ &\bigvee_{i \in [\ell]} (v = x_i \wedge P_{\ell-1, m+1}(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_\ell, \bar{y}, v)) \\ &\vee \left(\bigwedge_{i \in [\ell]} v \neq x_i \wedge \bigwedge_{i \in [m]} v \neq y_i \wedge P_{\ell,m}(\bar{x}, \bar{y}) \right) \quad \text{for } \ell \geq 1, \ell + m = k \end{aligned}$$

Uncolouring a node v . The update formulas for a change $\text{DEL}_R(v)$ are analogous to the update formulas for a change $\text{INS}_R(v)$ as seen above; they are provided in the full version.

Inserting an edge (v, w) . When an edge (v, w) is inserted, the number of active, covered nodes can change at most by one. At first, a covered node w might become inactive. This happens when w had in-degree k before the insertion. Or, an active node w becomes covered. This happens if v is coloured and w had no coloured in-neighbour and in-degree at most $k - 1$ before the change. The update formula for ANS is accordingly

$$\varphi_{\text{INS}_E}^{\text{ANS}}(v, w) \stackrel{\text{def}}{=} \text{ANS} \oplus \left((N_k(w) \wedge \bigvee_{i \in [k]} N_i^\bullet(w)) \vee (R(v) \wedge N_0^\bullet(w) \wedge \bigvee_{i \in [k]} N_{i-1}(w)) \right).$$

The necessary updated for relations $P_{\ell, m}$ are conceptionally very similar. We list the conditions that characterize whether the membership of w in $\mathcal{N}^{\bullet\circ}(A, B)$ changes, for a set $A = \{x_1, \dots, x_\ell\}$ of coloured nodes and a set $B = \{y_1, \dots, y_m\}$ of uncoloured nodes.

- Before the change, $w \in \mathcal{N}^{\bullet\circ}(A, B)$ holds, but not afterwards. This is either because w becomes inactive or because the new edge (v, w) connects w with another coloured node v . This case is expressed by the formula

$$\psi_1 \stackrel{\text{def}}{=} \bigwedge_{i \in [\ell]} E(x_i, w) \wedge N_\ell^\bullet(w) \wedge \bigwedge_{i \in [m]} E(y_i, w) \wedge (N_k(w) \vee R(v)).$$

- Before the change, $w \in \mathcal{N}^{\bullet\circ}(A, B)$ does not hold, but it does afterwards. Then w needs to be active and to have an incoming edge from all but one node from $A \cup B$, and v is that one node. Additionally, w has no other coloured in-neighbours. The following formulas ψ_2, ψ_3 express these conditions for the cases $v \in A$ and $v \in B$, respectively.

$$\begin{aligned} \psi_2 &\stackrel{\text{def}}{=} \bigvee_{i \in [\ell]} (v = x_i \wedge \bigwedge_{j \in [\ell] \setminus \{i\}} E(x_j, w) \wedge \bigwedge_{j \in [m]} E(y_j, w) \wedge N_{\ell-1}^\bullet(w) \wedge \bigvee_{j \in [k]} N_{j-1}(w)) \\ \psi_3 &\stackrel{\text{def}}{=} \bigvee_{i \in [m]} (v = y_i \wedge \bigwedge_{j \in [\ell] \setminus \{i\}} E(y_j, w) \wedge \bigwedge_{j \in [\ell]} E(x_j, w) \wedge N_\ell^\bullet(w) \wedge \bigvee_{j \in [k]} N_{j-1}(w)) \end{aligned}$$

The update formula for $P_{\ell, m}$ is then

$$\varphi_{\text{INS}_E}^{P_{\ell, m}}(v, w; x_1, \dots, x_\ell, y_1, \dots, y_m) \stackrel{\text{def}}{=} \theta_{\ell, m}(\bar{x}, \bar{y}) \wedge (P_{\ell, m}(\bar{x}, \bar{y}) \oplus (\psi_1 \vee \psi_2 \vee \psi_3)).$$

Deleting an edge (v, w) . The ideas to construct the update formulas for changes $\text{DEL}_E(v, w)$ are symmetrical to the constructions for changes $\text{INS}_E(v, w)$. When an edge (v, w) is deleted, the node w becomes active if its in-degree before the change was $k + 1$. It is (still) covered, and then is a new active and covered node, if it has coloured in-neighbours other than v . This is the case if w has at least two coloured in-neighbours before the change, or if it has at least one coloured in-neighbour and v is not coloured.

On the other hand, if v was the only coloured in-neighbour of an active node w , this node is not covered any more.

Details are provided in the full version. ◀

Our proof does not go through for $k < 3$, as we use ternary auxiliary relations to maintain whether a node has degree at most k , see Example 6.

3.2 Inexpressibility results for $\text{ParityExists}_{\text{deg} \leq k}$

In this subsection we prove that k -ary auxiliary relations are not sufficient to maintain $\text{PARITYEXISTS}_{\text{deg} \leq k+1}$, for every $k \in \mathbb{N}$. The proof technique we use, and formalise as Lemma 8, is a reformulation of the proof technique of [16], which combines techniques from

[10] and [17] with insights regarding upper and lower bounds for Ramsey numbers. We actually use a special case of the formalisation from [14, Lemma 7.4], which is sufficient for our application.

The technique consists of a sufficient condition under which a Boolean query q cannot be maintained in DynProp with at most k -ary auxiliary relations. The condition basically requires that for each collection \mathcal{B} of subsets of size $k+1$ of a set $\{1, \dots, n\}$, for an arbitrary n , there is a structure \mathcal{I} and a sequence $\alpha(x_1), \dots, \alpha(x_{k+1})$ of changes such that (1) the elements $1, \dots, n$ cannot be distinguished by quantifier-free formulas, and (2) the structure that results from \mathcal{I} by applying the changes $\alpha(i_1), \dots, \alpha(i_{k+1})$ in that order is a positive instance for q exactly if $\{i_1, \dots, i_{k+1}\} \in \mathcal{B}$.

In the following, we denote the set $\{1, \dots, n\}$ by $[n]$ and write $(\mathcal{I}, \bar{a}) \equiv_0 (\mathcal{I}, \bar{b})$ if \bar{a} and \bar{b} have the same length and agree on their quantifier-free type in \mathcal{I} , that is, $\mathcal{I} \models \psi(\bar{a})$ if and only if $\mathcal{I} \models \psi(\bar{b})$ for all quantifier-free formulas ψ . We denote the set of all subsets of size k of a set A by $\binom{A}{k}$.

► **Lemma 8** ([14]). *Let q be a Boolean query of σ -structures. Then q is not in k -ary DynProp, even with arbitrary initialisation, if for each $n \in \mathbb{N}$ and all subsets $\mathcal{B} \subseteq \binom{[n]}{k+1}$ there exist*

- a σ -structure \mathcal{I} and a set $P = \{p_1, \dots, p_n\}$ of distinct elements such that
 - P is a subset of the domain of \mathcal{I} ,
 - $(\mathcal{I}, p_{i_1}, \dots, p_{i_{k+1}}) \equiv_0 (\mathcal{I}, p_{j_1}, \dots, p_{j_{k+1}})$ for all strictly increasing sequences i_1, \dots, i_{k+1} and j_1, \dots, j_{k+1} over $[n]$, and
- a sequence $\alpha(x_1), \dots, \alpha(x_{k+1})$ of changes

such that for all strictly increasing sequences i_1, \dots, i_{k+1} over $[n]$:

$$(\alpha(p_{i_1}) \circ \dots \circ \alpha(p_{i_{k+1}}))(\mathcal{I}) \in q \iff \{i_1, \dots, i_{k+1}\} \in \mathcal{B}.$$

With the help of Lemma 8 we can show the desired inexpressibility result.

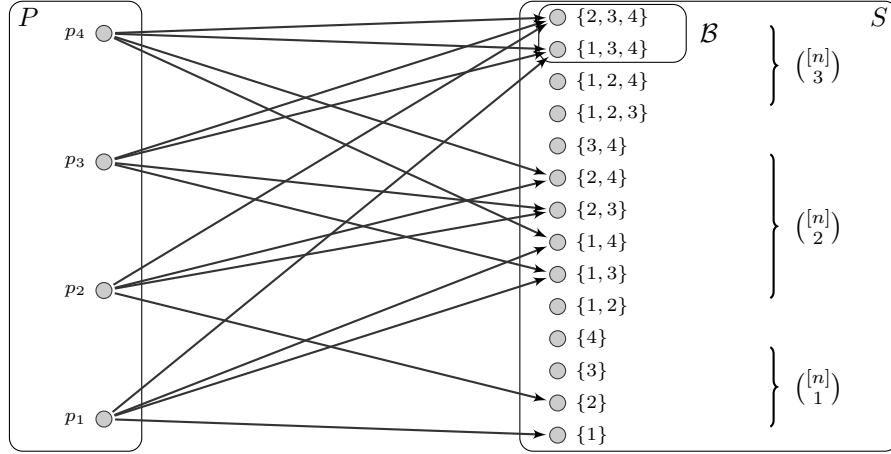
► **Proposition 9.** *For every $k \geq 0$, $\text{PARITYEXISTS}_{\text{deg} \leq k+1}$ is not in k -ary DynProp, even with arbitrary initialisation.*

In the following, for a graph $G = (V, E)$ and some set $X \subseteq V$ of nodes we write $\mathcal{N}^{\rightarrow}(X)$ for the set $\{v \mid \exists u \in X: E(u, v)\}$ of out-neighbours of nodes in X . For singleton sets $X = \{x\}$ we just write $\mathcal{N}^{\rightarrow}(x)$ instead of $\mathcal{N}^{\rightarrow}(\{x\})$.

Proof. Let $k \in \mathbb{N}$ be fixed. We apply Lemma 8 to show that $\text{PARITYEXISTS}_{\text{deg} \leq k+1}$ is not in k -ary DynProp.

The basic proof idea is simple. Given a collection $\mathcal{B} \subseteq \binom{[n]}{k+1}$, we construct a graph $G = (V, E)$ with distinguished nodes $P = \{p_1, \dots, p_n\} \subseteq V$ such that (1) each node has in-degree at most $k+1$ and (2) for each $B \in \binom{[n]}{k+1}$ the set $\mathcal{N}^{\rightarrow}(\{p_i \mid i \in B\})$ is of odd size if and only if $B \in \mathcal{B}$. Then applying a change sequence α which colours all nodes p_i with $i \in B$ to G results in a positive instance of $\text{PARITYEXISTS}_{\text{deg} \leq k+1}$ if and only if $B \in \mathcal{B}$. An invocation of Lemma 8 yields the intended lower bound.

It remains to construct the graph G . Let S be the set of all non-empty subsets of $[n]$ of size at most $k+1$. We choose the node set V of G as the union of P and S . Only nodes in P will be coloured, and only nodes from S will be covered. A first attempt to realise the idea mentioned above might be to consider an edge set $E_{k+1} \stackrel{\text{def}}{=} \{(p_i, B) \mid B \in \mathcal{B}, i \in B\}$; then, having fixed some set $B \in \mathcal{B}$, the node B becomes covered whenever the nodes p_i with $i \in B$ are coloured. However, also some nodes $B' \neq B$ will be covered, namely if $B' \cap B \neq \emptyset$, and the number of these nodes influences the query result. We ensure that



■ **Figure 1** Example for the construction in the proof of Proposition 9, with $k = 2$ and $n = 4$.

the set of nodes $B' \neq B$ that are covered by $\{p_i \mid i \in B\}$ is of even size, so that the parity of $|\mathcal{N}^{\rightarrow}(\{p_i \mid i \in B\})|$ is determined by whether $B \in \mathcal{B}$ holds. This will be achieved by introducing edges to nodes $\binom{[n]}{i} \in S$ for $i \leq k$ such that for every subset P' of P of size at most k the number of nodes from S that have an incoming edge from *all* nodes from P' is even. By an inclusion-exclusion argument we conclude that for any set $\hat{P} \in \binom{P}{k+1}$ the number of nodes from S that have an incoming edge from *some* node of \hat{P} , but not from all of them, is even. It follows that whenever $k+1$ nodes $p_{i_1}, \dots, p_{i_{k+1}}$ are marked, the number of covered nodes is odd precisely if there is one node in S that has an edge from *all* nodes $p_{i_1}, \dots, p_{i_{k+1}}$, which is the case exactly if $\{i_1, \dots, i_{k+1}\} \in \mathcal{B}$.

We now make this precise. Let n be arbitrary and let $P = \{p_1, \dots, p_n\}$. For a set $X \subseteq [n]$ we write P_X for the set $\{p_i \mid i \in X\}$.

The structure \mathcal{I} we construct consists of a coloured graph $G = (V, E)$ with nodes $V \stackrel{\text{def}}{=} P \cup S$, where $S \stackrel{\text{def}}{=} \binom{[n]}{1} \cup \dots \cup \binom{[n]}{k+1}$, and initially empty set $R \stackrel{\text{def}}{=} \emptyset$ of coloured nodes. The edge set $E = E_1 \cup \dots \cup E_{k+1}$ is constructed iteratively in $k+1$ steps. We first define the set E_{k+1} and define the set E_j based on the set $E_{>j} \stackrel{\text{def}}{=} \bigcup_{j'=j+1}^{k+1} E_{j'}$.

The set E_{k+1} consists of all edges (p_i, B) such that $B \in \mathcal{B}$ and $i \in B$. For the construction of the set E_j with $j \in \{1, \dots, k\}$ we assume that all sets $E_{j'}$ with $j' > j$ have already been constructed. Let $X \in \binom{[n]}{j}$ be a set and let m be the number of nodes $Y \in S$ for which there are already edges $(p_i, Y) \in E_{>j}$ for all nodes p_i in P_X . If m is odd, then there is so far an odd number of nodes from S that have an incoming edge from all $p_i \in P_X$. As we want this number to be even, we let E_j contain edges (p_i, X) for all $i \in X$. If m is even, no edges are added to E_j . See Figure 1 for an example of this construction. Note that for each $X \in \binom{[n]}{i}$, for $i \in \{1, \dots, k+1\}$, the degree of X in G is at most i , and therefore also at most $k+1$.

We now show that for a set $B \in \binom{[n]}{k+1}$ the cardinality of $\mathcal{N}^{\rightarrow}(P_B)$ is indeed odd if and only if $B \in \mathcal{B}$. This follows by an inclusion-exclusion argument. For a set $X \subseteq [n]$ the set $\mathcal{N}^{\rightarrow}(P_X)$ contains all nodes with an incoming edge from a node in P_X . It is therefore equal to the union $\bigcup_{i \in X} \mathcal{N}^{\rightarrow}(p_i)$. When we sum up the cardinalities of these sets $\mathcal{N}^{\rightarrow}(p_i)$, any node in $\mathcal{N}^{\rightarrow}(P_X)$ with edges to both p_i and p_j , for numbers $i, j \in X$, is accounted for twice. Continuing this argument, the cardinality of $\mathcal{N}^{\rightarrow}(X)$ can be computed as follows.

$$|\mathcal{N}^{\rightarrow}(P_X)| = \sum_{i \in X} |\mathcal{N}^{\rightarrow}(p_i)| - \sum_{\substack{i, j \in X \\ i < j}} |\mathcal{N}^{\rightarrow}(p_i) \cap \mathcal{N}^{\rightarrow}(p_j)| + \dots + (-1)^{|X|-1} \left| \bigcap_{i \in X} \mathcal{N}^{\rightarrow}(p_i) \right|$$

By construction of G , the set $\bigcap_{i \in Y} \mathcal{N}^{\rightarrow}(p_i)$ is of even size, for all sets $Y \subseteq [n]$ of size at most k . Consequently, for each $X \in \binom{[n]}{k+1}$ the parity of $|\mathcal{N}^{\rightarrow}(P_X)|$ is determined by the parity of $\left| \bigcap_{i \in X} \mathcal{N}^{\rightarrow}(p_i) \right|$, the last term in the above equation. Only the node X can possibly have incoming edges from all nodes p_i in P_X , and these edges exist if and only if $X \in \mathcal{B}$.

Let $\alpha(x_1), \dots, \alpha(x_{k+1})$ be the change sequence $\text{INS}_R(x_1), \dots, \text{INS}_R(x_{k+1})$ that colours the nodes x_1, \dots, x_{k+1} . Let $B \in \binom{[n]}{k+1}$ be of the form $\{i_1, \dots, i_{k+1}\}$ with $i_1 < \dots < i_{k+1}$. The change sequence $\alpha_B \stackrel{\text{def}}{=} \alpha(p_{i_1}) \cdots \alpha(p_{i_{k+1}})$ results in a graph where the set of coloured nodes is exactly P_B . As all nodes in $\mathcal{N}^{\rightarrow}(P_B)$ have degree at most $k+1$ and the set $\mathcal{N}^{\rightarrow}(P_B)$ is of odd size exactly if $B \in \mathcal{B}$, we have that $\alpha_B(\mathcal{I})$ is a positive instance of $\text{PARITYEXISTS}_{\text{deg} \leq k+1}$ if and only if $B \in \mathcal{B}$. \blacktriangleleft

4 ParityExists and first-order updates

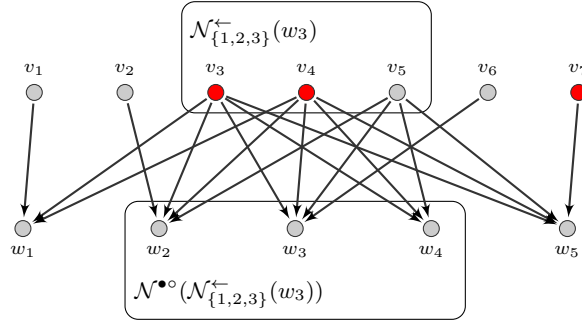
As discussed in the introduction, the PARITY query can be easily maintained with first-order update rules. So far we have seen that its generalisation PARITYEXISTS can only be maintained with quantifier-free update rules if the in-degree of covered nodes is bounded by a constant. Now we show that with first-order update rules, this query can be maintained if the in-degree is bounded by $\log n$, where n is the number of nodes in the graph. We emphasise that only the in-degree of covered nodes is bounded, while a coloured node v can cover arbitrarily many nodes. If also the out-degree of coloured node is restricted, maintenance in DynFO becomes trivial.

We start by providing a dynamic program with first-order update rules that maintains $\text{PARITYEXISTS}_{\text{deg} \leq k}$, for a constant k , and only uses unary relations apart from a linear order. Thus, in contrast to quantifier-free update rules, this query cannot be used to obtain an arity hierarchy for graph queries for first-order update rules. Afterwards we will exploit the technique used here to maintain $\text{PARITYEXISTS}_{\text{deg} \leq \log n}$ with binary auxiliary relations.

► **Theorem 5.** *$\text{PARITYEXISTS}_{\text{deg} \leq k}$ can be maintained in DynFO with unary auxiliary relations in the presence of a linear order, for every $k \in \mathbb{N}$.*

An intuitive reason why quantifier-free dynamic programs need auxiliary relations of growing arity to maintain $\text{PARITYEXISTS}_{\text{deg} \leq k}$ is that for checking whether some change, for instance the colouring of a node v , is “relevant” for some node w , it needs to have access to all of w ’s “important” neighbours. Without quantification, the only way to do this is to explicitly list them as elements of the tuple for which the update formula decides whether to include it in the auxiliary relation.

With quantification and a linear order, sets of neighbours can be defined more easily, if the total number of neighbours is bounded by a constant. Let us fix a node w with at most k (in-)neighbours, for some constant k . Thanks to the linear order, the neighbours can be distinguished as first, second, \dots , k -th neighbour of w , and any subset of these nodes is uniquely determined and can be defined in FO by the node w and a set $I \subseteq \{1, \dots, k\}$ that *indexes* the neighbours. With this idea, the proof of Proposition 7 can be adjusted appropriately for Theorem 5.



■ **Figure 2** An illustration of the notation used in the proof of Theorem 5. The set $\mathcal{N}_G^{\bullet\circ}(\mathcal{N}_{\{1,2,3\}}^{\leftarrow}(w_3))$ does not include w_1 , as there is no edge (v_5, w_1) , and it does not include w_5 , as there is an edge (v_7, w_5) for a coloured node $v_7 \notin \mathcal{N}_{\{1,2,3\}}^{\leftarrow}(w_3)$.

Proof sketch (of Theorem 5). Let $k \in \mathbb{N}$ be some constant. Again, we call a node *active* if its in-degree is at most k . We sketch a dynamic program that uses a linear order on the nodes and otherwise at most unary auxiliary relations.

Let I be a non-empty subset of $\{1, \dots, k\}$, and let w be an active node with at least $\max(I)$ in-neighbours. The set $\mathcal{N}_I^{\leftarrow}(w)$ of *I -indexed in-neighbours* of w includes a node v if and only if (v, w) is an edge in the input graph and v is the i -th in-neighbour of w with respect to the linear order, for some $i \in I$. The following notation is similar as in the proof of Proposition 7. For a graph G and an arbitrary set C of (coloured and uncoloured) nodes, we denote the set of active nodes that have an incoming edge from every node in C and no coloured in-neighbour that is not in C by $\mathcal{N}_G^{\bullet\circ}(C)$. An example for these notions is depicted in Figure 2.

For every $I \subseteq \{1, \dots, k\}$ with $I \neq \emptyset$ we introduce an auxiliary relation P_I with the following intended meaning. An active node w with at least $\max(I)$ neighbours is in P_I if and only if (1) w has no coloured in-neighbours that are not contained in $\mathcal{N}_I^{\leftarrow}(w)$, and (2) the set $\mathcal{N}_G^{\bullet\circ}(\mathcal{N}_I^{\leftarrow}(w))$ has odd size. Note that (1) implies that $w \in \mathcal{N}_G^{\bullet\circ}(\mathcal{N}_I^{\leftarrow}(w))$.

An auxiliary relation P_I basically replaces the relations $P_{\ell, m}$ with $\ell + m = |I|$ from the proof of Proposition 7, and the updates are mostly analogous.

We explain how the query relation ANS and the relations P_I are updated when a modification to the input graph occurs. When a node v is coloured, the query relation is only changed if v becomes the only coloured neighbour of an odd number of active nodes. This is the case if and only if there actually is an active and previously uncovered node w that v has an edge to and if $w \in P_I$ for the set $I \stackrel{\text{def}}{=} \{i\}$, where i is the number such that v is the i -th in-neighbour of w with respect to the linear order.

The update of a relation P_I after the colouring of a node v is as follows. Let G be the graph before the change is applied, and G' the changed graph. Let w be any active node. If v is an I -indexed in-neighbour of w , no change regarding $w \in P_I$ is necessary. Otherwise, some nodes in $\mathcal{N}_G^{\bullet\circ}(\mathcal{N}_I^{\leftarrow}(w))$ might now have a coloured neighbour v that is not contained in $\mathcal{N}_I^{\leftarrow}(w)$, and therefore are not contained in $\mathcal{N}_{G'}^{\bullet\circ}(\mathcal{N}_I^{\leftarrow}(w))$. Let w' be such a node, that is, a node with an edge from v and every node in $\mathcal{N}_I^{\leftarrow}(w)$, and let I' be such that $\mathcal{N}_{I'}^{\leftarrow}(w') = \mathcal{N}_I^{\leftarrow}(w) \cup \{v\}$. The parity of the number of nodes in $\mathcal{N}_G^{\bullet\circ}(\mathcal{N}_I^{\leftarrow}(w)) \setminus \mathcal{N}_{G'}^{\bullet\circ}(\mathcal{N}_I^{\leftarrow}(w))$ is odd if and only if $w' \in P_{I'}$. This can be used to update P_I .

We do not present the updates for the remaining changes as they can be easily constructed along the same lines. ◀

It is easy to maintain a linear order on the non-isolated nodes of an input graph [8], which is all that is needed for the proof of Theorem 5. So, $\text{PARITYEXISTS}_{\text{deg} \leq k}$ can also be maintained in DynFO without a predefined linear order, at the expense of binary auxiliary relations.

Unfortunately we cannot generalise the technique from Theorem 2 for $\text{PARITYEXISTS}_{\text{deg} \leq k}$ to PARITYEXISTS , but only to $\text{PARITYEXISTS}_{\text{deg} \leq \log n}$, which asks for the parity of the number of covered nodes with in-degree at most $\log n$. Here, n is the number of nodes of the graph.

► **Theorem 4.** *$\text{PARITYEXISTS}_{\text{deg} \leq \log n}$ can be maintained in DynFO with binary auxiliary relations in the presence of a linear order and BIT.*

Proof sketch. With the help of the linear order we identify the node set V of size n of the input graph with the numbers $\{0, \dots, n-1\}$, and use BIT to access the bit encoding of these numbers. Any node $v \in V$ then naturally encodes a set $I(v) \subseteq \{1, \dots, \log n\}$: $i \in \{1, \dots, \log n\}$ is contained in $I(v)$ if and only if the i -th bit in the bit encoding of v is 1.

The proof of Theorem 5 constructs a dynamic program that maintains unary relations P_I with $I \subseteq \{1, \dots, k\}$, and $w \in P_I$ holds if $w \in \mathcal{N}_G^{\bullet\circ}(\mathcal{N}_I^{\leftarrow}(w))$ and if $|\mathcal{N}_G^{\bullet\circ}(\mathcal{N}_I^{\leftarrow}(w))|$ is odd. We replace these relations by a single binary relation P , with the intended meaning that $(v, w) \in P$ if $w \in \mathcal{N}_G^{\bullet\circ}(\mathcal{N}_{I(v)}^{\leftarrow}(w))$ and if $|\mathcal{N}_G^{\bullet\circ}(\mathcal{N}_{I(v)}^{\leftarrow}(w))|$ is odd.

A dynamic program that maintains $\text{PARITYEXISTS}_{\text{deg} \leq \log n}$ can then be constructed along the same lines as in the proof of Theorem 5. ◀

In addition to a linear order, [8] also shows how corresponding relations addition and multiplication can be maintained for the active domain of a structure. As BIT is first-order definable in the presence of addition and multiplication, and vice versa (see e.g. [12, Theorem 1.17]), both a linear order and BIT on the active domain can be maintained, still using only binary auxiliary relations. So, the variant of $\text{PARITYEXISTS}_{\text{deg} \leq \log n}$ that considers n to be the number of non-isolated nodes, instead of the number of all nodes, can be maintained in binary DynFO without assuming built-in relations.

5 Conclusion

We studied the dynamic complexity of the query PARITYEXISTS as well as its bounded degree variants. While it remains open whether PARITYEXISTS is in DynFO, we showed that $\text{PARITYEXISTS}_{\text{deg} \leq \log n}$ is in DynFO and that $\text{PARITYEXISTS}_{\text{deg} \leq k}$ is in DynProp, for fixed $k \in \mathbb{N}$. The latter result is the basis for an arity hierarchy for DynProp for Boolean graph queries. Several open questions remain.

► **Open question.** *Can PARITYEXISTS be maintained with first-order updates rules? If so, are all (domain-independent) queries from $\text{FO} + \text{Parity}$ also in DynFO?*

► **Open question.** *Is there an arity hierarchy for DynFO for Boolean graph queries?*

Orthogonally to the perspectives taken in this work, one can ask how many auxiliary bits are necessary to maintain the query PARITYEXISTS or, more generally, all queries expressible in first-order logic extended by modulo quantifiers. It is convenient to switch the view point from first-order updates to updates computed by AC^0 circuits for discussing the amount of auxiliary bits. The class DynFO corresponds to (uniform) DynAC^0 , and allows for polynomially many auxiliary bits. It is not hard to see that if one allows quasi-polynomially many auxiliary bits and update circuits of quasi-polynomial size, then all queries expressible in first-order logic extended by modulo quantifiers can be maintained. This was observed in discussions with Samir Datta, Raghav Kulkarni and Anish Mukherjee. A proof sketch is provided in the full version of this paper.

References

- 1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.
- 2 Miklós Ajtai. Σ_1^1 -Formulae on finite structures. *Annals of Pure and Applied Logic*, 24(1):1–48, 1983. doi:10.1016/0168-0072(83)90038-6.
- 3 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On Uniformity within NC^1 . *J. Comput. Syst. Sci.*, 41(3):274–306, 1990. doi:10.1016/0022-0000(90)90022-D.
- 4 Samir Datta, Raghav Kulkarni, Anish Mukherjee, Thomas Schwentick, and Thomas Zeume. Reachability Is in DynFO. *J. ACM*, 65(5):33:1–33:24, August 2018. doi:10.1145/3212685.
- 5 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*, Volume 15, Issue 2, May 2019. doi:10.23638/LMCS-15(2:12)2019.
- 6 Guozhu Dong and Jianwen Su. Arity Bounds in First-Order Incremental Evaluation and Definition of Polynomial Time Database Queries. *J. Comput. Syst. Sci.*, 57(3):289–308, 1998. doi:10.1006/jcss.1998.1565.
- 7 Guozhu Dong and Louxin Zhang. Separating Auxiliary Arity Hierarchy of First-Order Incremental Evaluation Systems Using $(3k+1)$ -ary Input Relations. *Int. J. Found. Comput. Sci.*, 11(4):573–578, 2000. doi:10.1142/S0129054100000302.
- 8 Kousha Etessami. Dynamic Tree Isomorphism via First-Order Updates. In Alberto O. Mendelzon and Jan Paredaens, editors, *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 235–243. ACM Press, 1998. doi:10.1145/275487.275514.
- 9 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, Circuits, and the Polynomial-Time Hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984. doi:10.1007/BF01744431.
- 10 Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *ACM Trans. Comput. Log.*, 13(3):19, 2012. doi:10.1145/2287718.2287719.
- 11 William Hesse. *Dynamic Computational Complexity*. PhD thesis, University of Massachusetts Amherst, 2003.
- 12 Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999. doi:10.1007/978-1-4612-0539-5.
- 13 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.
- 14 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.
- 15 Thomas Schwentick and Thomas Zeume. Dynamic complexity: recent updates. *SIGLOG News*, 3(2):30–52, 2016. doi:10.1145/2948896.2948899.
- 16 Thomas Zeume. The dynamic descriptive complexity of k -clique. *Inf. Comput.*, 256:9–22, 2017. doi:10.1016/j.ic.2017.04.005.
- 17 Thomas Zeume and Thomas Schwentick. On the quantifier-free dynamic complexity of Reachability. *Inf. Comput.*, 240:108–129, 2015. doi:10.1016/j.ic.2014.09.011.