8th International Conference on Formal Structures for Computation and Deduction

FSCD 2023, July 3–6, 2023, Rome, Italy

Edited by

Marco Gaboardi
Femke van Raamsdonk

LIPICS

LIPIcs – Vol. 260 – FSCD 2023 www.dagstuhl.de/lipics

Editors

Marco Gaboardi
Boston University, MA, USA
gaboardi@bu.edu

Femke van Raamsdonk
Vrije Universiteit Amsterdam, The Netherlands
f.van.raamsdonk@vu.nl

ACM Classification 2012
Theory of computation → Models of computation; Theory of computation → Formal languages and automata theory; Theory of computation → Logic; Theory of computation → Semantics and reasoning; Security and privacy → Formal methods and theory of security; Computing methodologies → Symbolic and algebraic manipulation; Software and its engineering → Formal software verification

ISBN 978-3-95977-277-8

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-277-8.

Publication date
June, 2023

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

Digital Object Identifier: 10.4230/LIPIcs.FSCD.2023.0

ISBN 978-3-95977-277-8 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Papers

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

This volume contains the proceedings of the 8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023), which was held July 3–6, 2023 in Rome, Italy. FSCD 2023 was co-located with the 29th International Conference on Automated Deduction, (CADE 2023), which was held July 1–4, 2023 in Rome, Italy.

The conference FSCD (`https://fscd-conference.org/`) covers all aspects of formal structures for computation and deduction, from theoretical foundations to applications. Building on two communities, RTA (Rewriting Techniques and Applications) and TLCA (Typed Lambda Calculi and Applications), FSCD embraces their core topics and broadens their scope to include closely related areas in logic and proof theory, new emerging models of computation, as well as semantics and verification in new and challenging areas.

The FSCD program featured four invited talks, given by Maribel Fernández (King's College London, UK), Mateja Jamnik (University of Cambridge, UK), Giulio Manzonetto (LIPN&CNRS, Université Sorbonne Paris Nord, France), and Akihisa Yamada (Cyber Physical Security Research Center, National Institute of Advanced Industrial Science and Technology - AIST, Japan). The invited talks by Maribel Fernández and Mateja Jamnik were joint with CADE 2023; their contributions are also in the proceedings of CADE 2023.

The Program Committee of FSCD 2023 consisted of 31 members from 18 countries. FSCD 2023 received 60 submission with contributing authors from 19 countries. Almost every submitted paper has been reviewed by at least three PC members with the help of in total 90 external reviewers. The reviewing process, which included a rebuttal phase, took place over a period of nine weeks. A total of 30 papers were accepted for publication and are included in these proceedings. The EasyChair conference management system has been a very useful tool in all phases of the work of the Program Committee.

The Program Committee awarded the FSCD 2023 Best Paper Award by Junior Researchers to Taichi Uemura from Stockholm University for his paper 'Homotopy type theory as internal languages of diagrams of ∞-logoses'.

In addition to the main conference, nine FSCD-associated workshop were held before or during the conference:

- DCM 2023: 13th International Workshop on Developments in Computational Models,
- HOR 2023: 11th International Workshop on Higher-Order Rewriting,
- IFIP WG 1.6: annual meeting of the IFIP Working Group 1.6 on Rewriting,
- LFMTP 2023: International Workshop on Logical Frameworks and Meta-Languages: Theory and Practice,
- LSFA 2023: 18th International Workshop on Logical and Semantic Frameworks, with Applications,
- TLLA 2023: Seventh International Workshop on Trends in Linear Logic and Applications,
- UNIF 2023: 37th International Workshop on Unification,
- WiL 2023: 7th International Workshop Women in Logic,
- WPTE 2023: 10th International Workshop on Rewriting Techniques for Program Transformations and Evaluation.

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This volume of FSCD 2023 is published in the LIPIcs series under a Creative Commons license: online access is free to all papers and authors retain rights over their contributions. We thank the Leibniz Center for Informatics at Schloss Dagstuhl, and in particular Michael Wagner and Michael Didas, for their prompt and helpful replies to our questions regarding the production of these proceedings, for their flexibility, and for their user-friendly submission system.

Many people have helped to make FSCD 2023 a successful meeting. On behalf of the Program Committee, we thank the authors of submitted papers for considering FSCD as a venue for their work. We are grateful to the Program Committee and to the external reviewers for their careful and constructive review and evaluation of the submitted papers. We thank all invited speakers for enriching the program with their talks. In addition, the associated workshops highly contributed to the lively scientific atmosphere of the meeting. We thank the organizers of the workshops, the Steering Committee Workshop Chair Cynthia Kop, and the Conference Workshop Chair Ivano Salvo for their efforts. Warm thanks to the Conference Chair of FSCD 2023, Daniele Gorla and his colleagues for the excellent organization of the conference. We thank the co-chairs of CADE 2023, Brigitte Pientka from McGill University en Cesare Tinelli of the University of Iowa, for the fruitful collaboration in the organization of our co-located events. We are very grateful to the Steering Committee of FSCD for their valuable and helpful guidance in setting up the meeting, and for ensuring that FSCD will have a bright and enduring future. We thank in particular the Steering Committee chair Herman Geuvers and the Steering Committee Publicity Chair Carsten Fuhs for their help and advice in smaller and larger matters during the preparation of the conference. Finally, we thank all participants of the conference for creating a lively and interesting event.

Marco Gaboardi and Femke van Raamsdonk
Program Committee co-chairs of FSCD 2023

Committees

Program Committee

Martin Avanzini	INRIA, France
Patrick Bahr	ITU Copenhagen, Denmark
Pablo Barenbaum	University of Quilmes (CONICET) & ICC, Argentina
Filippo Bonchi	University of Pisa, Italy
Sabine Broda	University of Porto, Portugal
Valeria De Paiva	Topos Institute, USA
Andrej Dudenhefner	TU Dortmund, Germany
Santiago Escobar	Universitat Politècnica de València, Spain
Claudia Faggian	University of Paris (IRIF), France
Frank (Peng) Fu	Dalhousie University, Canada
Marco Gaboardi (Co-Chair)	Boston University, USA
Silvio Ghilardi	University of Milano, Italy
Clemens Grabmayer	Gran Sasso Science Institute, Italy
Nao Hirokawa	JAIST, Japan
Mirai Ikebuchi	National Institute of Informatics, Japan
Ambrus Kaposi	Eötvös Loránd University, Hungary
Ian Mackie	University of Sussex, UK
Radu Mardare	University of Strathclyde, UK
Aart Middeldorp	University of Innsbruck, Austria
Anders Mörtberg	University of Stockholm, Sweden
Daniele Nantes-Sobrinho	Imperial College London, UK & University of Brasília, Brazil
Vincent van Oostrom	Independent researcher, The Netherlands
Femke van Raamsdonk (Co-Chair)	Vrije Universiteit Amsterdam, The Netherlands
Grigore Rosu	University of Illinois Urbana-Champaign, USA
Luca Roversi	University of Torino, Italy
Aleksy Schubert	University of Warsaw, Poland
Jakob G. Simonsen	University of Copenhagen, Denmark
Alwen Tiu	Australian National University, Australia
Valeria Vignudelli	CNRS/ENS Lyon, France
Johannes Waldmann	HTWK Leipzig, Germany
Sarah Winkler	University of Bolzano, Italy

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Conference Chair

Daniele Gorla	Sapienza Università di Roma, Italy

Workshop Chair

Ivano Salvo	Sapienza Università di Roma, Italy

Steering Committee Workshop Chair

Cynthia Kop	Radboud University Nijmegen, The Netherlands

Steering Committee Publicity Chair

Carsten Fuhs	Birkbeck, University of London, UK

Steering Committee

Zena Ariola	University of Oregon, USA
Alejandro Díaz-Caro	Quilmes University & ICC/CONICET, Argentina
Amy Felty	University of Ottawa, Canada
Carsten Fuhs	Birkbeck, University of London, UK
Herman Geuvers (Chair)	Radboud University Nijmegen, The Netherlands
Silvia Ghilezan	University of Novi Sad, Serbia
Jürgen Giesl	RWTH Aachen University, Germany
Stefano Guerrini	Université de Paris 13, France
Delia Kesner	Université de Paris Diderot, France
Naoki Kobayashi	University of Tokyo, Japan
Cynthia Kop	Radboud University Nijmegen, The Netherlands
Luigi Liquori	Inria, France
Daniele Nantes	Imperial College London, UK & University of Brasília, Brazil

External Reviewers

Nicolas Amat
Pedro Ângelo
Mauricio Ayala-Rincón
Giorgio Bacci
Davide Barbarossa
Stefano Berardi
Benno van den Berg
Richard Blute
Rafaël Bocquet
Roberto Bruni
Mihai Calancea
Matteo Capucci
Evan Cavallo
David Cerna
Jules Chouquet
Pierre Clairambault
Liron Cohen
Bruce Collie
Anupam Das
Ivan Di Liberti
Martín Diéguez
Harley Eades III
Jörg Endrullis
Mário Florido
Jonas Frey
Francesco Gavazzo
Alfons Geser
Alexis Ghyselen
Alessandro Gianola
Marianna Girlando
Tamara von Glehn
Rajeev Gore
Daniel Gratzer
Adrien Guatto
Stefano Guerrini
Jules Hedges
Stefan Hetzl
Naohiko Hoshino
Andrzej Indrzejczak
Marie Kerjean
Delia Kesner
Cynthia Kop
Clemens Kupke
Dietrich Kuske
Felix Laarmann
James Laird
Salvador Lucas
Giulio Manzonetto
Ralph Matthes
Nelma Moreira
Haiko Muller
Guillaume Munch-Maccagnoni
Yuito Murase
Keisuke Nakano
Koji Nakazawa
Lê Thành Dũng Nguyễn
Paulo Oliva
Dominic Orchard
Eugenio Orlandelli
Hugo Paquet
Romain Péchoux
Luc Pellissier
Thomas Powell
Jakob von Raumer
Laurent Regnier
Michel Rigo
Alessio Santamaria
Ralph Sarkis
Alexis Saurin
Gabriel Scherer
Ezra Schoen
Jonas Schöpf
Ian Shillito
Michael Shulman
Alex Simpson
Nicholas Smallbone
Siva Somayyajula
Bas Spitters
Toby St Clere Smithe
Lutz Straßburger
René Thiemann
Taichi Uemura
Gabriele Vanoni
Pedro Vasconcelos
Daniel Ventura
Luca Viganò
Albert Visser
Jan Willem Klop
Akihisa Yamada
Noson Yanofsky

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Thorsten Altenkirch (24)
School of Computer Science,
University of Nottingham, UK

Pablo Barenbaum (11)
ICC, Universidad de Buenos Aires,
Argentina; Universidad Nacional
de Quilmes (CONICET), Buenos
Aires, Argentina

Emmanuel Beffara (28)
Univ. Grenoble Alpes, CNRS,
Grenoble INP, LIG, 38000
Grenoble, France

Nicolas Behr (17)
CNRS, Université Paris Cité, IRIF,
France

Lison Blondeau-Patissier (13)
Aix Marseille Univ, CNRS, I2M
and LIS, Marseille, France

Valentin Blot (32)
Inria, Laboratoire Méthodes
Formelles, Université Paris-Saclay,
France

Rafaël Bocquet (18)
Eötvös Loránd University,
Budapest, Hungary

Flavien Breuvart (21)
LIPN, USPN, Villetaneuse, France

Félix Castro (28)
IRIF, Université Paris Cité,
France; IMERL, Facultad de
Ingeniería, Universidad de la
República, Montevideo, Uruguay

Serenella Cerrito (23)
Université Paris Saclay, Univ
EVRY, France

Pierre Clairambault (13)
Aix Marseille Univ, CNRS, LIS,
Marseille, France

Francesco Dagnino (25)
DIBRIS, University of Genova,
Italy

Ugo Dal Lago (20)
University of Bologna, Italy;
INRIA Sophia Antipolis, France

Anupam Das (27)
University of Birmingham, UK

Rémi Di Guardia (26)
Univ Lyon, EnsL, UCBL, CNRS,
LIP, F-69342, LYON Cedex 07,
France

Saraid Dwyer Satterfield (30)
University of Mary Washington,
Fredericksburg, VA, USA

Thomas Ehrhard (8)
Université de Paris Cité, CNRS,
IRIF, F-75013, Paris, France

Serdar Erbatur (30)
University of Texas at Dallas, TX,
USA

José Espírito Santo (19)
Centre of Mathematics, University
of Minho, Portugal

Claudia Faggian (8)
Université de Paris Cité, CNRS,
IRIF, F-75013, Paris, France

Maribel Fernández (1)
Department of Informatics, King's
College London, UK

Samuel Frontull (22)
Universität Innsbruck, Austria

Didier Galmiche (31)
Université de Lorraine, CNRS,
LORIA, Nancy, France

Thomas Genet (7)
Université de Rennes, IRISA,
France

Dan R. Ghica (14)
University of Birmingham, UK

Sergey Goncharov (34)
Friedrich-Alexander-Universität
Erlangen-Nürnberg, Germany

Valentin Goranko (23)
Stockholm University, Sweden; and
University of the Witwatersrand,
Johannesburg, South Africa
(visiting professorship)

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Mauricio Guillermo (28)
IMERL, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay

Nao Hirokawa (12)
School of Information Science, JAIST, Ishikawa, Japan

Naohiko Hoshino (20)
Sojo University, Japan

Ievgen Ivanov (9)
Taras Shevchenko National University of Kyiv, Ukraine

Mateja Jamnik (2)
Department of Computer Science and Technology, University of Cambridge, UK

Thomas Jensen (7)
Inria, Université de Rennes, France

Ambrus Kaposi (18, 24)
Eötvös Loránd University, Budapest, Hungary

George Kaye (14)
University of Birmingham, UK

Marie Kerjean (21)
CNRS, LIPN, USPN, Villetaneuse, France

Cynthia Kop (15)
Institute for Computing and Information Sciences, Radboud University, Nijmegen, The Netherlands

Nikolai Kudasov (10)
Innopolis University, Tatarstan Republic, Russia

James Laird (33)
Department of Computer Science, University of Bath, UK

Olivier Laurent (26)
Univ Lyon, EnsL, UCBL, CNRS, LIP, F-69342, LYON Cedex 07, France

Théo Losekoot (7)
Université de Rennes, IRISA, France

Giulio Manzonetto (3)
Université Sorbonne Paris Nord, LIPN, UMR 7030, CNRS, F-93430 Villetaneuse, France

Andrew M. Marshall (30)
University of Mary Washington, Fredericksburg, VA, USA

Lukas Melgaard (27)
University of Birmingham, UK

Paul-André Melliès (17)
CNRS, Université Paris Cité, INRIA, France

Filipa Mendes (19)
Centre of Mathematics, University of Minho, Portugal

Aart Middeldorp (12)
Department of Computer Science, Universität Innsbruck, Austria

Samuel Mimram (16)
LIX, École polytechnique, Palaiseau, France

Étienne Miquey (28)
Aix-Marseille Université, CNRS, I2M, Marseille, France

Simon Mirwasser (21)
LIPN, USPN, Villetaneuse, France

Georg Moser (22)
Universität Innsbruck, Austria

Daniel Méry (31)
Université de Lorraine, CNRS, LORIA, Nancy, France

Michele Pagani (8)
Université de Paris Cité, IRIF, F-75013, Paris, France

Sophie Paillocher (23)
Université Paris Saclay Univ EVRY, France

Fabio Pasquali (25)
DIMA, University of Genova, Italy

Daniele Pautasso (29)
Dipartimento di Informatica, University of Torino, Italy

Paolo Pistone (20)
University of Bologna, Italy

Christophe Ringeissen (30)
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Simona Ronchi Della Rocca (29)
Dipartimento di Informatica, University of Torino, Italy

Christian Sattler (18)
Chalmers University of Technology, Gothenburg, Sweden

Cristian Sottile (11)
ICC, Universidad de Buenos Aires (CONICET), Argentina; Universidad Nacional de Quilmes, Buenos Aires, Argentina

Taichi Uemura (5)
Stockholm University, Sweden

Deivid Vale (15)
Institute for Computing and Information Sciences, Radboud University, Nijmegen, The Netherlands

Niels van der Weide (6)
Institute for Computing and Information Sciences, Radboud University, Nijmegen, The Netherlands

Vincent van Oostrom (22)
Barendrecht, The Netherlands

Lionel Vaux Auclair (13)
Aix Marseille Univ, CNRS, I2M, Marseille, France

Tamás Végh (24)
Eötvös Loránd University, Budapest, Hungary

Akihisa Yamada (4)
National Institute of Advanced Industrial Science and Technology, Tokyo, Japan

Noam Zeilberger (17)
École Polytechnique, LIX, Palaiseau, France

Artjoms Šinkarovs (24)
Heriot-Watt University, Edinburgh, UK

Nominal Techniques for Software Specification and Verification

Maribel Fernández
Department of Informatics, King's College London, UK

Abstract

In this talk we discuss the nominal approach to the specification of languages with binders and some applications to programming languages and verification.

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

Keywords and phrases Binding operator, Nominal Logic, Nominal Rewriting, Unification, Equational Theories, Type Systems

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.1

Category Invited Talk

Funding This work is partially funded by the Royal Society (International Exchanges, grant number IES\R2\212106).

Acknowledgements The work described in this abstract is the result of collaborations with several researchers, cited below. Special thanks to my PhD students and co-authors.

1 Overview

The nominal approach to the specification of languages with binding operators, introduced by Gabbay and Pitts [27, 21, 20], has its roots in nominal set theory [26]. Its user-friendly syntax and first-order presentation (indeed, nominal logic [25] is defined as a theory in first-order logic) makes formal reasoning about binding operators similar to conventional on-paper reasoning.

Nominal logic uses the well-understood concept of *permutation groups acting on sets* to provide a rigorous, first-order treatment of common informal practice to do with fresh and bound names. Nominal matching and nominal unification [34, 35] (which work modulo α-equivalence) are decidable and efficient algorithms exist [7, 8, 22, 9], which are the basis for efficient implementations of nominal rewriting [19, 17, 18].

A number of systems (such as Nominal Isabelle [33]) highlighted the benefits of the nominal approach, which gave rise to elegant formalisations of Gödel's theorems [24] and the π-calculus [5] and to advances in programming language semantics [23]. However, there are still some obstacles to the inclusion of nominal features in programming languages and verification environments.

In this talk, I will present our current work towards incorporating nominal techniques into two widely-used rule-based first-order verification environments: the K specification framework [29] and the Maude programming language [11, 12].

An important component of rule-based programming and verification environments is the algorithm used to check equivalence of terms and to solve equations (unification). In practice, unification problems arise in the context of equational axioms (e.g., to take into account associative and commutative (AC) operators [32, 31, 13, 14, 6]). The first part of the talk will discuss notions of α-equivalence modulo associativity and commutativity axioms [1],

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 1; pp. 1:1–1:4
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

extensions of nominal matching and unification to deal with AC operators [2], and the use of nominal narrowing [3] to deal with equational theories presented by convergent nominal rewriting rules.

Another important component of these environments is the type system. In the second part of the talk, I will discuss type systems for nominal languages (including polymorphic systems [15] and intersection systems [4]). Dependent type theories, the dominant approach to formalising programming languages, have been extended with nominal features [10, 28, 30]. A lambda-less nominal dependent type system is available [16] and we are currently working on a type checker for this system.

The talk is structured as follows: we will start with the definition of nominal logic (including the notions of fresh atoms and alpha-equivalence) followed by a brief introduction to nominal matching and unification. We will then define nominal rewriting, a generalisation of first-order rewriting that provides in-built support for alpha-equivalence following the nominal approach. Finally, we will discuss notions of nominal unification and rewriting modulo AC operators and briefly overview typed versions of nominal languages.

References

1 Mauricio Ayala-Rincón, Washington de Carvalho Segundo, Maribel Fernández, Daniele Nantes-Sobrinho, and Ana Oliveira. A formalisation of nominal α-equivalence with A, C, and AC symbols. *Theor. Comput. Sci.*, 781:3–23, 2019. doi:10.1016/j.tcs.2019.02.020.

2 Mauricio Ayala-Rincón, Washington de Carvalho Segundo, Maribel Fernández, Gabriel Ferreira Silva, and Daniele Nantes-Sobrinho. Formalising nominal C-unification generalised with protected variables. *Math. Struct. Comput. Sci.*, 31(3):286–311, 2021. doi:10.1017/S0960129521000050.

3 Mauricio Ayala-Rincón, Maribel Fernández, and Daniele Nantes-Sobrinho. Nominal Narrowing. In *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016*, page 11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

4 Maurício Ayala-Rincón, Maribel Fernández, Ana Cristina Rocha-Oliveira, and Daniel Lima Ventura. Nominal essential intersection types. *Theoretical Computer Science*, 737:62–80, 2018. doi:10.1016/j.tcs.2018.05.008.

5 Jesper Bengtson and Joachim Parrow. Formalising the pi-calculus using nominal logic. *LMCS*, 5(2), 2009. URL: http://arxiv.org/abs/0809.3960.

6 Alexandre Boudet, Evelyne Contejean, and Hervé Devie. A New AC Unification Algorithm with an Algorithm for Solving Systems of Diophantine Equations. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), Philadelphia, Pennsylvania, USA, June 4-7, 1990*, pages 289–299. IEEE Computer Society, 1990.

7 Christophe Calvès and Maribel Fernández. A polynomial nominal unification algorithm. *Theor. Comp. Sci.*, 403:285–306, August 2008.

8 Christophe Calvès and Maribel Fernández. Matching and alpha-equivalence check for nominal terms. *Journal of Comp. Syst. Sci.*, 76(5):283–301, 2010.

9 Christophe Calvès and Maribel Fernández. The first-order nominal link. In *Logic-Based Program Synthesis and Transformation - 20th International Symposium, LOPSTR 2010, Hagenberg, Austria, July 23-25, 2010, Revised Selected Papers*, volume 6564 of *LNCS*, pages 234–248. Springer, 2011.

10 James Cheney. A dependent nominal type theory. *Logical Methods in Computer Science*, 8(1), 2012.

11 Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott, editors. *All About Maude - A High-Performance Logical Framework*, volume 4350 of *LNCS*. Springer, 2007. doi:10.1007/978-3-540-71999-1.

12 Francisco Durán, Steven Eker, Santiago Escobar, Narciso Martí-Oliet, José Meseguer, Rubén Rubio, and Carolyn L. Talcott. Programming and symbolic computation in Maude. *J. Log. Algebr. Meth. Program.*, 110, 2020.

13 François Fages. Associative-Commutative Unification. In Robert E. Shostak, editor, *7th International Conference on Automated Deduction, Napa, California, USA, May 14-16, 1984, Proceedings*, volume 170 of *LNCS*, pages 194–208. Springer, 1984.

14 François Fages. Associative-Commutative Unification. *J. of Sym. Computation*, 3(3):257–275, 1987.

15 Elliot Fairweather and Maribel Fernández. Typed nominal rewriting. *ACM Transactions on Computational Logic*, 19(1):6:1–6:46, 2018. doi:10.1145/3161558.

16 Elliot Fairweather, Maribel Fernández, Nora Szasz, and Alvaro Tasistro. Dependent types for nominal terms with atom substitutions. In *Typed Lambda Calculus and Applications (Proceedings of TLCA)*, pages 180–195, 2015. doi:10.4230/LIPIcs.TLCA.2015.180.

17 Maribel Fernández and Murdoch J. Gabbay. Nominal rewriting. *Inf. Comput.*, 205(6):917–965, 2007.

18 Maribel Fernández and Murdoch J. Gabbay. Closed nominal rewriting and efficiently computable nominal algebra equality. In *LFMTP*, 2010.

19 Maribel Fernández, Murdoch J. Gabbay, and Ian Mackie. Nominal rewriting systems. In *PPDP*, pages 108–119. ACM Press, August 2004.

20 Murdoch J. Gabbay. Foundations of nominal techniques: logic and semantics of variables in abstract syntax. *Bulletin of Symbolic Logic*, 2011.

21 Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13(3–5):341–363, July 2001.

22 Jordi Levy and Mateu Villaret. An efficient nominal unification algorithm. In *Proceedings of the 21st International Conference on Rewriting Techniques and Applications (RTA 2010)*, volume 6 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 209–226. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010.

23 Andrzej S. Murawski and Nikos Tzevelekos. Nominal game semantics. *FTPL*, 2:4:191–269, 2016. doi:10.1561/2500000017.

24 Lawrence C. Paulson. A mechanised proof of Gödel's incompleteness theorems using Nominal Isabelle. *J. Autom. Reasoning*, 55(1):1–37, 2015. doi:10.1007/s10817-015-9322-8.

25 Andrew M. Pitts. Nominal logic: A first order theory of names and binding. In *TACS*, volume 2215 of *LNCS*, pages 219–242. Springer, 2001.

26 Andrew M Pitts. *Nominal sets: Names and symmetry in computer science.* Cambridge UP, 2013.

27 Andrew M. Pitts and Murdoch J. Gabbay. A metalanguage for programming with bound names modulo renaming. In *Proceedings of the 5th international conference on the mathematics of program construction (MPC 2000)*, volume 1837 of *LNCS*, pages 230–255. Springer, December 2000. URL: http://www.gabbay.org.uk/papers.html#metpbn.

28 Andrew M. Pitts, Justus Matthiesen, and Jasper Derikx. A dependent type theory with abstractable names. *Electr. Notes Theor. Comput. Sci.*, 312:19–50, 2015. doi:10.1016/j.entcs.2015.04.003.

29 Grigore Rosu and Traian-Florin Serbanuta. An overview of the K semantic framework. *J. Log. Algebr. Program.*, 79(6):397–434, 2010. doi:10.1016/j.jlap.2010.03.012.

30 Ulrich Schöpp and Ian Stark. A Dependent Type Theory with Names and Binding. In *CSL*, pages 235–249, 2004.

31 Mark Stickel. A Unification Algorithm for Associative-Commutative Functions. *J. of the ACM*, 28(3):423–434, 1981.

32 Mark E. Stickel. A Complete Unification Algorithm for Associative-Commutative Functions. In *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, Tbilisi, Georgia, USSR, September 3-8, 1975*, pages 71–76, 1975.

33 Christian Urban. Nominal techniques in Isabelle/HOL. *J. Autom. Reason.*, 40(4):327–356, May 2008.

34 Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal unification. In *CSL*, volume 2803 of *LNCS*, pages 513–527. Springer, December 2003. URL: `http://www.gabbay.org.uk/papers.html#nomu`, `doi:10.1016/j.tcs.2004.06.016`.

35 Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal unification. *Theor. Comp. Sci.*, 323(1–3):473–497, 2004. `doi:10.1016/j.tcs.2004.06.016`.

How Can We Make Trustworthy AI?

Mateja Jamnik ✉ 🏠

Department of Computer Science and Technology, University of Cambridge, UK

Abstract

Not too long ago most headlines talked about our fear of AI. Today, AI is ubiquitous, and the conversation has moved on from whether we should use AI to how we can trust the AI systems that we use in our daily lives. In this talk I look at some key technical ingredients that help us build confidence and trust in using intelligent technology. I argue that intuitiveness, interaction, explainability and inclusion of human domain knowledge are essential in building this trust. I present some of the techniques and methods we are building for making AI systems that think and interact with humans in more intuitive and personalised ways, enabling humans to better understand the solutions produced by machines, and enabling machines to incorporate human domain knowledge in their reasoning and learning processes.

2012 ACM Subject Classification Computing methodologies → Artificial intelligence; Computing methodologies → Machine learning

Keywords and phrases AI, human-centric computing, knowledge representation, reasoning, machine learning

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.2

Category Invited Talk

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 2; pp. 2:1–2:1
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A Lambda Calculus Satellite

Giulio Manzonetto
Université Sorbonne Paris Nord, LIPN, UMR 7030, CNRS, F-93430 Villetaneuse, France

Abstract

We shortly summarize the contents of the book "A Lambda Calculus Satellite", presented at the 8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).

2012 ACM Subject Classification Theory of computation → Lambda calculus

Keywords and phrases λ-calculus, rewriting, denotational models, equational theories

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.3

Category Invited Talk

Acknowledgements I want to thank Henk Barendregt for accepting to embark with me in the ambitious project of writing a "sequel" of his yellow book *The Lambda Calculus*, for all the pleasant gatherings and interesting discussions about λ-calculus, consciousness and life. Both Henk and I are grateful to Stefano Guerrini and Vincent Padovani for their contributions to the Satellite.

Introduction

The λ-calculus was introduced by Alonzo Church around 1930 as the kernel of a more general investigation on the foundations of mathematics and logic [18, 19], and played a prominent role in theoretical computer science for more than fifty years.[1] Under the influence of the pioneering work of Corrado Böhm, who established his famous separation theorem [11], and Dana Scott, who constructed the first model of λ-calculus [56], the research on λ-calculus has flourished in the seventies.[2] In that period a wealth of results were established by applying techniques arising from several areas of computer science and mathematics, namely recursion theory, algebra, topology and category theory. These approaches revealed that λ-calculus can be studied from different, although interconnected, perspectives:

- *The λ-calculus as a rewriting system.* The set Λ of *λ-terms* can be endowed with *notions of reductions*, thus it deserves the status of a higher-order term rewriting system:

  $$(\lambda x.M)N \to_\beta M[x{:=}N], \qquad\qquad \lambda x.Mx \to_\eta M,\ \text{ if } x \notin \mathrm{FV}(M).$$

  This approach has its roots in the Church-Rosser Theorem [20] establishing the confluence of $\beta(\eta)$-reduction, and therefore the unicity of normal forms, and in the Standardization Theorem [23] from which it follows that the leftmost-outermost reduction strategy is normalizing. Subsequently, researchers studied the reduction sequences originating from a λ-term by performing a fine-grained analysis of the creation of redexes and by tracking the residuals of sets of redexes. This analysis culminated in a proof of the Finite Developments Theorem, in its various formulations [25, 34, 32].
- *The analysis of the cost of normalization.* The idea of considering the λ-calculus an actual programming language was taken very seriously by Böhm and Gross, who designed in the sixties the CUCH machines as an implementational model [13]. Therefore, the

[1] We consider here the research on untyped λ-calculus – we could argue that variations of λ-calculus are still omnipresent in the literature concerning logical systems as well as idealized programming languages.
[2] For a more detailed survey of the early history of λ-calculus, we refer the reader to [17].

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 3; pp. 3:1–3:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

problem of determining the computational complexity of normalization via β-reduction arose naturally. This led Jean-Jacques Lévy to develop in his *Thèse de doctorat d'État* an interesting notion of *optimal reduction* [45].

- *The λ-calculus as a model of computation.* A well-established result shows that any partial recursive numerical function can be represented by a λ-term operating on Church numerals. However, observing the behavior of λ-terms exclusively on numerals represents a very narrow point of view. In general, a λ-term F can be seen as a total function

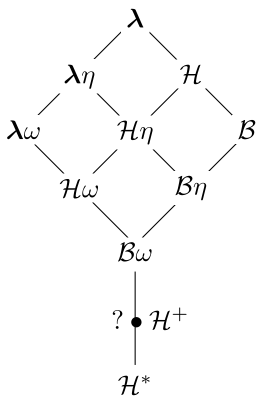
  $$F : \Lambda \to \Lambda$$

  and one can study its $\mathsf{Range}(F) = \{FN \mid N \in \Lambda\}$ or the set of its fixed points $\mathsf{Fix}(F) = \{N \mid FN =_\beta N\}$ and wonder if they are finite or infinite modulo β-conversion [6, 37]. Moreover, λ-terms can be classified into solvable and unsolvable, depending on their capability of interaction with the environment: a *solvable* term can be transformed into the identity I when plugged in a suitable context $C[\,]$, while *unsolvables* are unable to interact with any context. Equivalently, solvable terms must provide at least a stable portion of their output (their head nf) while unsolvables correspond to looping programs. This classification led Barendregt to define the *Böhm tree* $\mathrm{BT}(M)$ of a λ-term M, a possibly infinite tree constructed by coinductively collecting all stable pieces of information coming out of its computations, and eventually representing the complete evaluation of M [4].
- *The lattice of λ-theories.* The equational theories of λ-calculus are called *λ-theories* and become the main subject of study when one is more interested in program equivalence than in the process of reduction. Some λ-theories are particularly interesting for computer scientists because they capture operational properties of λ-terms, e.g., *extensional theories* equating all extensionally equivalent λ-terms, or *sensible theories* collapsing all unsolvables, or theories equating all λ-terms having the same evaluation trees. Morris introduced a class of *observational theories* specifying when two λ-terms are observationally equivalent [49]. This means that two λ-terms M and N are considered equivalent whenever one can plug either M or N into any context $C[\,]$ without noticing any difference in the global behavior. Observational equivalences thus depend on the kind of behavior one is interested in observing – a parametricity that can be represented by a set $\mathcal{O} \subseteq \Lambda$ of *observables*:

  $$\mathcal{T}_{\mathcal{O}} = \{M = N \mid \forall C[\,]\,.\,[\; C[M] \in \mathcal{O} \iff C[N] \in \mathcal{O}\;]\}.$$

  Therefore, even if the set of λ-theories constitutes a complete lattice of cardinality $2^{\aleph_0}$, researchers have mostly focussed on the following λ-theories that form a kite-shaped diagram where $\mathcal{T}_1$ is depicted above $\mathcal{T}_2$ whenever $\mathcal{T}_1 \subsetneq \mathcal{T}_2$ holds:

  $$\begin{array}{lcl}
  \boldsymbol{\lambda} & = & \{M = N \mid M =_\beta N\}. \\
  \boldsymbol{\lambda}\eta & = & \{M = N \mid M =_{\beta\eta} N\}. \\
  \mathcal{H} & = & \{M = N \mid M, N \text{ are unsolvable}\}. \\
  \mathcal{H}^+ & = & \mathcal{T}_{\mathrm{NF}},\ \text{where } \mathrm{NF} = \{M \in \Lambda \mid M \text{ has a } \beta\text{-nf}\}. \\
  \mathcal{H}^* & = & \mathcal{T}_{\mathrm{SOL}},\ \text{where } \mathrm{SOL} = \{M \in \Lambda \mid M \text{ is solvable}\}. \\
  \mathcal{B} & = & \{M = N \mid \mathrm{BT}(M) = \mathrm{BT}(N)\}. \\
  \mathcal{T}\eta & = & \text{the closure of a } \lambda\text{-theory } \mathcal{T} \text{ under the } \eta\text{-rule}. \\
  \mathcal{T}\omega & = & \text{the closure of a } \lambda\text{-theory } \mathcal{T} \text{ under the } \omega\text{-rule, which is} \\
  & & \dfrac{FZ = GZ \text{ for all } Z \in \Lambda^o}{F = G}\ \omega\text{-rule}
  \end{array}$$



The ω-*rule* defined above states if two λ-terms F, G are equal whenever they are applied to the same closed argument Z (i.e. $Z \in \Lambda^o$), then they are equal. This form of extensionality is inspired from set-theoretical definition of "function equality" and encompasses the η-rule. Proving that the inclusion $\boldsymbol{\lambda}\eta \subsetneq \boldsymbol{\lambda}\omega$ is strict requires some clever construction [53].

- *The model theoretic approach.* The model theory of λ-calculus has been developed along three axes. The first one is algebraic and proposes as models (a subclass of) the variety of *combinatory algebras*, based on the combinators $\mathbf{K}, \mathbf{S}$ from Combinatory Logic [23]. The second one, based on syntactic λ-models, is closely related to the algebraic definition, but more set-theoretical [31]. The third one is category-theoretic and focuses on the notion of reflexive object living in Cartesian closed categories [2]. The relationship among these notions has been explored by Koymans [42]: in general, categorical models correspond to λ-*algebras* (describing, e.g., closed term models), that are moreover λ-*models* when the underlying category is "well-pointed".

  Concerning the construction of individual models, the most famous ones are Scott's $\mathcal{D}_\infty$ whose theory is the observational theory $\mathcal{H}^*$, as proved by Hyland [33] and Wadsworth [67] (independently), and Plotkin-Scott's $\mathcal{P}_\omega$ that induces the theory $\mathcal{B}$ of Böhm trees [52, 57].

At the end of the seventies, Barendregt decided to collect all these results (and others) in the monograph "The Lambda Calculus. Its syntax and semantics" [5], presenting the state of the art of research in λ-calculus at that time. Subsequently translated in Russian and Chinese, this book is nowadays omnipresent in academic libraries of computer science all around the world. Several open problems concerning semantical and syntactic aspects of λ-calculus were proposed in the book, often in the form of conjectures:

1. Do invertible λ-terms correspond to bijective λ-terms, modulo $=_{\beta\eta}$? [5, Exercise 21.4.9].
2. Does the Perpendicular Lines Property hold, modulo β-equality? [5, Chapter 14].
3. The λ-theory $\mathcal{H}$ satisfies the range property. [5, Conjecture 20.2.8].
4. The position of $\mathcal{H}^+$ in the kite of λ-theories is $\mathcal{B}\omega \subsetneq \mathcal{H}^+ \subsetneq \mathcal{H}^*$. Conjecture by P. Sallé reported in the proof of [5, Theorem 17.4.16].
5. The λ-theory $\mathcal{H}\omega$ is Π^1_1-complete. [5, Conjecture 17.4.15].

Most of these problems have been solved in the subsequent 35 years, but the results are scattered throughout the literature and difficult to piece together. Some of these solutions occupied an entire PhD thesis, e.g. Folkerts [29] (1995), or part of a thesis, e.g. Polonsky [54] (2011), with the complexity (but not the method) of a proof using priority for results on the degrees of undecidability. In 2017, Hyland suggested to Barendregt and the present author that the time had come to write another book, intended as a "satellite" of [5], in which all these solutions could be presented in a more clear and uniform way, improving and simplifying the proofs (when possible). We accepted his challenge, and profited from the occasion to include the solution of problems that weren't explicitly stated in [5], and other related research that was conducted in the last decades.

The resulting monograph, entitled "A Lambda Calculus Satellite", has been published in 2022 by College Publications [9] and is structured in six parts:

1. Preliminaries
2. Reduction
3. Conversion
4. Theories
5. Models
6. Open Problems

In the reminder of the paper, we briefly describe the contents of these parts – each composed by several chapters – and discuss some pedagogical choices that we made.

1 Preliminaries

In order to make the satellite book [9] as self-contained as possible[3], we start with a preliminary part surveying all the notions and results about λ-calculus[4] that are needed in the rest of the book. This part is divided into three chapters: 1) about syntactic properties; 2) Böhm trees and their variations; 3) models and theories of λ-calculus.

Besides presenting the syntax and operational semantics of λ-calculus in a nutshell, Chapter 1 contains more developed sections discussing properties of reduction, like finite developments and standardization, the Reduction under Substitution (RuS) technique by Diederik van Daalen [65], and its consequences [26]. Chapter 2 is devoted to present three kinds of coinductively defined trees representing the operational behavior of λ-terms, namely Böhm trees, Berarducci trees and Lévy-Longo trees. We profit from the occasion to mention that coinduction is treated in this book using the modern approach suggested in [43]. Since coinduction has been around for decades, and it is nowadays better understood in the scientific community, we adopt a more informal style of coinductive reasoning that greatly improves the readability and benefits the reader. Intuitively, we can do this without compromising the soundness of our proofs because the definition of Böhm trees is inherently *productive*. At the end of the chapter, we introduce two distinct notions of extensional Böhm trees corresponding to η-Böhm trees and Nakajima trees (respectively). The reason why these notions are incompatible with Berarducci and Lévy-Longo trees becomes clear later. In Chapter 3 we recall the main results concerning the lattice of λ-theories [46], and describe the relationships existing among the different definitions of a model of λ-calculus. In particular, we revisit Koymans' construction of a combinatory algebra starting from a categorical model – giving a λ-model only when the underlying category is *well pointed* – in favor of the more general construction described in [16] that works in every Cartesian closed category.

2 Reduction

In this part we consider properties of reductions in several systems: the regular λ-calculus, its infinitary version, and the **S**-fragment of Combinatory Logic.

Chapter 4. Leaving a β-reduction plane. Any λ-term M belongs to some *β-reduction plane*, which is defined as follows. Given $P, Q \in \Lambda$, write $P \circlearrowleft_\beta Q$ if $P \twoheadrightarrow_\beta Q \twoheadrightarrow_\beta P$. Then, a *$\beta$-reduction plane* is any $\circlearrowleft_\beta$-equivalence class. Clearly, M belongs to its own plane $[M]_{\circlearrowleft_\beta}$. It is possible to *leave a plane $\mathcal{P}$ at point $M \in \mathcal{P}$* if there is an N such that $M \to_\beta N \notin \mathcal{P}$. In 1980, Jan Willem Klop conjectured that if one can leave a plane at one of its points, then such a plane can be left at any of its points [41]. A few years later, Hans Mulder [50] and Sekimoto and Hirokawa [58] have refuted this conjecture (independently). In this chapter, we present Mulder's counterexample because he constructs a λ-term M containing a free variable x, but β-reducing to a closed term: $M \twoheadrightarrow_\beta P \in \Lambda^o$. Therefore, in order to conclude that $P \notin [M]_{\circlearrowleft_\beta}$ it is sufficient to invoke the fact that a λ-term cannot create free variables along β-reduction. The counterexample in [58] is a closed term, whence proving that it actually leaves its plane requires a more subtle reasoning. Both counterexamples are minimal, in the sense that they contain a minimal amount of redexes (i.e. 3) and rely on the fact that λ-calculus satisfies the ξ-rule.

[3] In our intent, it should be readable without having the previous book at hand.

[4] Auxiliary definitions and results about category theory, domain theory and universal algebra are presented in a technical appendix.

Chapter 5. Optimal lambda reduction. A consequence of the Standardization Theorem is that, if a λ-term is normalizing, then its normal form can be reached by repeatedly contracting the leftmost-outermost redex. However, due to the duplication of redexes, the leftmost-outermost reduction strategy might not be optimal, in the sense that the normal form could be reached in a number of steps which is not minimal. In [45], Jean-Jacques Lévy introduced the notion of *redex family* to capture an intuitive idea of *optimal sharing* between "copies" of the same redex. By studying the causal history of redexes using a suitable labeled extension of λ-calculus it is possible to define an optimal reduction method for λ-calculus. Compared to the most recent presentation of this material [1], our discussion enters deeply into the technical details of the so-called *extraction method*, furnishes more examples, and discusses some implementations.

Chapter 6. Infinitary lambda calculus. In λ-calculus there are terms, like Turing's fixed point combinator, generating an infinite reduction sequence. Pushing this reduction to infinity, one generates the infinite term $\lambda f.f(f(f(\cdots)))$, which is an in-line depiction of its Böhm tree. Inspired by this phenomenon, Kennaway, Klop, Sleep, and de Vries introduced the *infinitary λ-calculus* [40], whose terms and reductions can possibly be infinite. The resulting infinitary term rewriting system is well defined and enjoys the unicity of its normal forms, but many properties fail like normalization and – most importantly – confluence. Berarducci showed that collapsing meaningless terms allows to restore confluence, a property recently proved in `Coq` by Czajka used coinductive methods [24], and induces a new model of λ-calculus based on Berarducci trees [10]. By modifying the notion of meaningless terms, one also retrieves Böhm trees and Lévy-Longo trees as infinitary normal forms of λ-terms. Adding η-reduction to Berarducci trees or Lévy-Longo trees breaks confluence again, while this notion of reduction is compatible with Böhm trees.

Chapter 7. Starlings. In this birdwatching chapter our binoculars are focused on the *starling* $\mathbf{S}$, living in Smullyan's enchanted forest of combinatory terms [59] and exhibiting the following behavior:

$$\mathbf{S}xyz \to_w xz(yz)$$

We consider the $\mathbf{S}$-fragment of Combinatory Logic, namely combinatory terms built up from application and $\mathbf{S}$ exclusively. Many of these have a weak normal form, like $\mathbf{SSSSSSS}$. Others do not, like $\mathbf{S}(\mathbf{SS})\mathbf{SSSS}$ and $\mathbf{SSS}(\mathbf{SSS})(\mathbf{SSS})$. Many properties that are undecidable in the context of λ-calculus and Combinatory Logic, become decidable in this setting. For instance, the question of whether the normalization of $\mathbf{S}$ is decidable was answered positively by Johannes Waldmann in his PhD thesis [68]. We also present original results by Vincent Padovani on the $\mathbf{S}$-fragment of Combinatory Logic, including:

- the fact that termination of head reduction is decidable;
- the existence of two non-interconvertible terms having the same Berarducci tree.

Whether the inter-convertibility $P =_w Q$ is decidable is still an open problem.

3 Conversion

In this part we consider properties of $\beta(\eta)$-conversion, although the classification here is rather subjective. Indeed, these properties could equivalently be formulated using the λ-theories $\boldsymbol{\lambda}, \boldsymbol{\lambda}\eta$, or even the associated term models. In general, given a λ-theory $\mathcal{T}$, it is equivalent to work with the associated equality $=_{\mathcal{T}}$ or in the term model $\mathcal{M}(\mathcal{T})$, while working in the closed term model $\mathcal{M}^o(\mathcal{T})$ is equivalent to consider *closed* λ-terms modulo $=_{\mathcal{T}}$.

Chapter 8. Perpendicular Lines Property. In the Cartesian plane $\mathbb{R}^2$ the two lines $\{(x,2) \mid x \in \mathbb{R}\}$ and $\{(3,y) \mid y \in \mathbb{R}\}$ are perpendicular. Translated to λ-terms, one says that $\{(X,\mathsf{I}) \mid X \in \Lambda\}$ and $\{(\mathsf{K},Y) \mid Y \in \Lambda\}$ are "perpendicular", and similarly in higher dimensions. The *perpendicular lines property* (PLP) states that if a λ-definable function F of k arguments is constant on k perpendicular lines, then F is constant everywhere. As usual, the validity of this property depends on the notion of equality which is considered, and on whether one focuses on closed terms, or open terms are also allowed. The validity of PLP has been shown for $\mathcal{M}(\mathcal{B})$ by using Berry's stability [5], for $\mathcal{M}^o(\mathcal{B})$ in the satellite book by using coinductive methods, and for $\mathcal{M}(\boldsymbol{\lambda})$ by Endrullis and de Vrijer using Reduction under Substitution [26]. Statman and Barendregt proved that $\mathcal{M}^o(\boldsymbol{\lambda}) \not\models$ PLP using variants of Plotkin terms [64].

Chapter 9. Bijectivity and invertibility in $\lambda\eta$. In set theory a function $f : X \to X$ is bijective if and only if it is invertible. More precisely, f is surjective whenever it is right-invertible, and injective whenever it is left-invertible. Now, given a λ-theory $\mathcal{T}$, every closed λ-term $F \in \Lambda^o$ can be considered as a function $F\colon \mathcal{M}^o(\mathcal{T}) \to \mathcal{M}^o(\mathcal{T})$, so it makes sense to wonder whether this correspondence still holds:

> Assuming that F is a bijection, can one conclude that F is $\mathcal{T}$-invertible?

The problem is more difficult because the inverses of F are required to be λ-definable. In other words, is there a λ-term $G \in \Lambda^o$ such that $F \circ G =_{\mathcal{T}} G \circ F =_{\mathcal{T}} \mathsf{I}$? For $\mathcal{T} = \boldsymbol{\lambda}$ the answer is positive because the only β-invertible closed λ-term is the identity I. This was shown in [12]. The invertibility problem for $\mathcal{T} = \boldsymbol{\lambda}\eta$ was first raised in [5, Exercise 21.4.9]. More than 10 years later, Enno Folkerts showed that this correspondence does hold [29]. As a consequence of this, and of combined results by Dezani and Bergstra-Klop, a closed λ-term is $\boldsymbol{\lambda}\eta$-invertible if and only if it is a finite hereditary permutator.

4 Theories

In this part we study sensible λ-theories, that is, theories that equate all unsolvable λ-terms. Such terms indeed correspond to "looping" programs deprived of any computational content.

Chapter 10. Sensible theories. In this chapter we discuss two celebrated problems in λ-calculus, known as the range property and the fixed point property.

- The *range property* for a λ-theory $\mathcal{T}$ states that a combinator F, seen as a total function $F : \mathcal{M}^o(\mathcal{T}) \to \mathcal{M}^o(\mathcal{T})$ has either an infinite range or a singleton range (in other words, it is a constant function). For $\mathcal{T} = \boldsymbol{\lambda}$, this property has been conjectured by Böhm in [11] and proved by Barendregt in [5, Theorem 17.1.16]. The proof constitutes a striking example of the power of the hyperconnectedness property enjoyed by the Visser topology [66]. It is easy to check that the λ-theory $\mathcal{B}$ generated by equating all λ-terms having the same Böhm tree satisfies the range property, and the same reasoning generalizes to all λ-theories $\mathcal{T}$ in the interval $\mathcal{B} \subseteq \mathcal{T} \subseteq \mathcal{H}^*$. Barendregt conjectured that $\mathcal{H}$ satisfies the range property in [5, Conjecture 20.2.8], and this problem remained open for 30 years. This conjecture was refuted by Polonsky in his PhD thesis [54], where a λ-term having range 2 modulo $=_{\mathcal{H}}$ is constructed.
- The *fixed point property* for a λ-theory $\mathcal{T}$ states that every combinator has either one or infinitely many pairwise $\mathcal{T}$-distinct closed fixed points. The question whether this property holds for $\mathcal{T} = \boldsymbol{\lambda}$ was first raised by [37] and appears as Problem 25 in the TLCA list of open problems [36]. In this chapter, we present a λ-term violating the fixed point property for every sensible λ-theory, a result first appeared in [47].

Chapter 11. The kite. As mentioned in the introduction, researchers are mostly interested in those λ-theories constituting the kite-shape diagram depicted on Page 2. In the seventies, Patrick Sallé conjectured that the observational theory $\mathcal{H}^+$ should be placed in the diagram between $\mathcal{B}\omega$ and $\mathcal{H}^*$, that is, $\mathcal{B}\omega \subsetneq \mathcal{H}^+ \subsetneq \mathcal{H}^*$. The second inclusion actually follows from the work of Lévy [44] and Hyland [33] proving that $\mathcal{H}^+$ and $\mathcal{H}^*$ respectively capture the equalities induced by η-Böhm trees and Nakajima trees. The former inclusion turned out to be false: in an FSCD article [38], Intrigila *et al.* proved that $\mathcal{B}\omega = \mathcal{H}^+$, thus refuting Sallé's conjecture. We describe the main ingredients used in the proof:

$\underline{\mathcal{B}\omega \subseteq \mathcal{H}^+}$. This inclusion follows from the fact that $\mathcal{H}^+$ satisfies the ω-rule. This is a consequence of a weak separation theorem, first proved in [14], and satisfied by $\mathcal{H}^*$-equivalent terms, that are however different in $\mathcal{H}^+$.

$\underline{\mathcal{H}^+ \subseteq \mathcal{B}\omega}$. This inclusion is the difficult one. The proof exploits Lévy's characterization of $\mathcal{H}^+$ in terms of η-Böhm trees, the property that the "η-supremum" of two λ-terms (if any) is always λ-definable, the capability of λ-terms to work on the "codes" of other λ-terms via Gödelization, and the lemma stating that every closed λ-term becomes unsolvable when fed enough copies of Ω. To understand how these ingredients can be mixed together to obtain a proof of this theorem, the reader will need to actually study the chapter.

5 Models

In this part we present some advances on the denotational semantics of λ-calculus. Chapter 12 contains some preliminaries that are needed to understand the subsequent chapters.

Chapter 12. Ordered models and theories. In general, a model of λ-calculus only induces an equational theory (λ-theory) through the kernel of its interpretation function. In practice, most of the models individually introduced in the literature live in some cpo-enriched Cartesian closed category, whence they also induce an inequational theory. In this chapter, we introduce the inequational theories of λ-calculus independently from denotational considerations, and show that they inherit notions like extensionality and sensibility from λ-theories. We focus on observational inequational theories:

$$M \sqsubseteq_{\mathcal{O}} N \iff \forall C[\,]\,.\,[\; C[M] \in \mathcal{O} \quad\Rightarrow\quad C[N] \in \mathcal{O}\;],$$

still depending on a set $\mathcal{O}$ of observables. E.g., $\mathcal{H}^+ \vdash M = N \Leftrightarrow M \sqsubseteq_{\mathrm{NF}} N \;\&\; N \sqsubseteq_{\mathrm{NF}} M$. We conclude the chapter by recalling the (in)equational theories of the most known denotational models, like Scott's $\mathcal{D}_\infty$, Engeler's graph model $\mathcal{E}$, Plotkin's $\mathcal{P}_\omega$ and the like.

Chapter 13. Filter models. Intersection type assignment systems, introduced by [21], allow to give a logical description of several operational properties of λ-terms, like solvability and various forms of normalization. Moreover, thanks to the celebrated Stone's duality, they correspond to a class of denotational models of λ-calculus, called *filter models*. This nomenclature derives from the fact that the denotation of a λ-term is given by the filter of its types. We show that classical lattice models, whose construction mimics the one of Scott's $\mathcal{D}_\infty$, can be presented as filter models. Others examples are the original filter model $\mathcal{F}_{\mathrm{BCD}}$ defined by Barendregt, Coppo and Dezani [8] and the model $\mathcal{F}_{\mathrm{CDZ}}$ by Coppo, Dezani and Zacchi [22]. We mainly focus on the latter since it has been largely overlooked in the literature, with the notable exception of [55]. Exploiting Girard/Tait's reducibility candidates, we show that $\mathcal{F}_{\mathrm{CDZ}}$ satisfies an Approximation Theorem. Finally, using Lévy's extensional approximants, we prove that it is (in)equationally fully abstract for $\mathcal{H}^+$, a result first established in [22].

Chapter 14. Relational models. In the eighties Jean-Yves Girard realized that the category of sets and relations constitutes a simple quantitative semantics of Linear Logic [30], where the promotion $!A$ is given by the set of all finite multisets over A. The relational semantics of λ-calculus, obtained by applying the coKleisli construction, has been largely studied in the last decades because of its peculiar properties. First, its quantitative features allow to expose semantically intensional properties of λ-terms, like the amount of head-reduction steps needed to reach their head normal form. This also allows to endow fundamental results, like Approximation Theorems, with easy inductive proofs bypassing the usual techniques based on Tait's computability. Second, relational models can be expressed through tensor type[5] assignment systems whose inhabitation problem is decidable. Finally, the fact that it is a non-well-pointed category contributes to justify, together with categories of games, the interest in non well-pointed categorical models.

Chapter 15. Church algebras for λ-calculus. Combinatory algebras are considered algebraically pathological because they are never commutative, associative, finite or recursive. In fact, at first sight, they seem to have little in common with the mathematical structures that are usually considered in universal algebra. Salibra viewed these topics from a wider perspective and introduced the variety of *Church algebras* [48], namely algebras possessing two distinguished nullary terms representing the truth values, and a ternary term representing the if-then-else conditional construct, which is ubiquitous in programming languages. Beyond combinatory algebras, this class includes all Boolean algebras, Heyting algebras and rings with unity. Manzonetto and Salibra also proved that combinatory algebras satisfy a Representation Theorem stating that every combinatory algebra can be decomposed as a direct product of directly indecomposable combinatory algebras. It is therefore natural to study the *indecomposable semantics* of λ-calculus, namely the class of λ-models that are indecomposable in this sense. It turns out that this class is large enough to include all the main semantics of λ-calculus, but also largely incomplete: there is a wealth of λ-theories whose models must be decomposable. This furnishes a uniform algebraic proof of incompleteness for the main semantics.

6 Open Problems

In the last part of the book, we present some open problems in the hope that the next generation of scientists will solve them (and possibly write a book, forty years from now). Some are longstanding open problems or conjectures already present in the TLCA or RTA lists, we simply wish to draw the attention on them. Others arose during our discussions. We present here a few problems that should be of interest for the FSCD community.

6.1 Are there hyper-recurrent λ-terms?

In the article [63], Statman presents several notions of combinators that are not supposed to exist in λ-calculus, but whose existence is difficult to disprove. This list includes hyper-recurrent terms, uniform universal generators, and double fixed point combinators.

A closed λ-term M is called: *recurrent* if $M \twoheadrightarrow_\beta N$ entails $N \twoheadrightarrow_\beta M$; *hyper-recurrent* if, for every $N \in \Lambda^o$, $N =_\beta M$ implies that N is recurrent.

▶ **Problem 1** (Statman 1993). *Do hyper-recurrent λ-terms exist?*

[5] Intuitively, relevant intersection type systems where $\wedge$ is a non-idempotent operator, i.e. $\sigma \wedge \sigma \neq \sigma$.

This problem appears as Problem 52 in the RTA list of open problems [62]. Statman proved that hyper-recurrent combinators do not exist in Combinatory Logic [60], however the two calculi are known to satisfy different properties as term rewriting systems. For instance, in Combinatory Logic based on $\{\mathsf{S}, \mathsf{K}, \mathsf{I}\}$ there are no pure cycles, as shown by [28].

6.2 Is there a double fixed point combinator?

Consider the λ-term $\boldsymbol{\delta} = \lambda yx.x(yx)$. It was remarked by Böhm and van der Mey that Y is a fixed point combinator if and only if $\boldsymbol{\delta}Y =_\beta Y$ holds. It follows that, if Y is a fixed point combinator, then both $\boldsymbol{\delta}Y$ and $Y\boldsymbol{\delta}$ are fixed point generators. A *double fixed point combinator* is a λ-term Y satisfying

$$\boldsymbol{\delta}Y =_\beta Y =_\beta Y\boldsymbol{\delta}.$$

▶ **Problem 2** (Statman 1993). *Does there exist a double fixed point combinator?*

This problem appears as Problem 52 in the RTA list of open problems [61], and it is marked as "solved" since the appearance of [35]. However, in 2011, Endrullis has discovered a gap in a crucial case of the argument and the problem should therefore be considered as open. Klop considers this problem one of the most interesting problems in term rewriting, and Endrullis *et al.* [27] have developed a clocked mechanism in the hope of distinguishing every Y from $Y\boldsymbol{\delta}$, but their attempts were unsuccessful. For more information about this problem and other suggestions for a proof strategy, we refer to [47].

6.3 Does Combinatory Logic satisfy the Plane Property?

We consider here the Combinatory Logic with the basis of combinators $\{\mathbf{K}, \mathbf{S}\}$.

▶ **Problem 3** (Jan Willem Klop 1980). *If a combinatory term can leave a plane at one of its points, can such a plane be left at any of its points?*

This problem was first raised in [41], and a positive answer was conjectured. We mentioned before that the λ-calculus does not satisfy this property, but also that the counterexamples rely on the fact that λ-calculus satisfies the ξ-rule:

$$\frac{M = N}{\lambda x.M = \lambda x.N}\ (\xi)$$

Since Combinatory Logic does not satisfy this rule, the constructions in [50, 58] do not generalize to this setting.

6.4 Is the word problem for S decidable?

The following is another longstanding open problem concerning the $\mathbf{S}$-fragment of Combinatory Logic. Recall that the problems of determining whether an $\mathbf{S}$-term is strongly normalizing or head normalizing are both decidable.

▶ **Problem 4** (The word problem for $\mathbf{S}$). *Given* $\mathbf{S}$*-terms* P, Q*, is the problem of determining whether* $P =_w Q$ *decidable?*

First raised in [3]. Notice that, because of Padovani's counterexample [9, Theorem 7.49], the w-conversion $=_w$ does not coincide with the equality induced by Berarducci trees.

6.5 Are there denotational models of other observational theories?

Among the inequational theories defined in Chapter 12, the following have never been seriously studied in the literature:

$$
\begin{array}{lcll}
M \sqsubseteq_{m1} N & \iff & \forall C[\,]\,.\,[\; C[M] \text{ has a } \beta\text{-nf} & \Rightarrow \quad C[N] \text{ has the same } \beta\text{-nf} \;] \\
M \sqsubseteq_{[\mathsf{I}]_\beta} N & \iff & \forall C[\,]\,.\,[\; C[M] =_\beta \mathsf{I} & \Rightarrow \quad C[N] =_\beta \mathsf{I} \;]
\end{array}
$$

As a warm-up exercise, consider $X = \lambda zx.xzx$ and $Y = \lambda zx.xz(\lambda y.xy)$ and convince yourself that the following inequalities hold: $X\Omega \sqsubseteq_{m1} Y\Omega$ and $Y\Omega \sqsubseteq_{m1} X\Omega$.

▶ **Problem 5** (Manzonetto 2022). *Are there (in)equationally fully abstract denotational models for $\sqsubseteq_{m1}$ and $\sqsubseteq_{[\mathsf{I}]_\beta}$?*

Preliminary investigations made by Manzonetto in collaboration with Barbarossa, Breuvart and Kerinec suggest that fully abstract models for these theories might exist in the strongly stable semantics defined by Bucciarelli and Ehrhard [15].

6.6 Is the λ-theory $\mathcal{H}\omega$ Π^1_1-complete?

In [7], Barendregt *et al.* proved that for any Π^1_1-predicate P there exist closed λ-terms $\mathsf{B}^n_0, \mathsf{B}^n_1$ such that

$$
\mathrm{P}(n) \quad \Rightarrow \quad \mathcal{H}\omega \vdash \mathsf{B}^n_0 = \mathsf{B}^n_1,
$$

in which the fact that $\mathrm{P}(n)$ holds seems to have been properly used (which is not the case if one takes e.g. $\mathsf{B}^n_0 = \mathsf{I} = \mathsf{B}^n_1$). See also [5, Theorem 17.4.14]. This result motivated the conjecture that the λ-theory $\mathcal{H}\omega$ is Π^1_1-complete. This conjecture first appeared in [7] and was subsequently stated in [5, Conjecture 17.4.15]. A confirmation of Barendregt's conjecture was proposed by Intrigila and Statman in [39], but their proof contains a few issues that we explain in detail in [9, §6.3]. The problem should be then considered open.

▶ **Problem 6** (Barendregt 1978). *Is the λ-theory $\mathcal{H}\omega$ Π^1_1-complete?*

This is the last open problem from Barendregt's book [5] which is still standing.

Conclusions

I want to conclude by sharing some personal impressions about this experience. Writing a book on my research domain has been an incredibly enriching journey from a scientific perspective. Not only it gave me the opportunity to pick from Henk's mathematical wisdom, but it allowed me to study the state of the art of research in the field from a broader viewpoint. This brought to my attention many results that I wasn't aware of, and forced me to enter into the technicalities of proofs that I previously studied only superficially, eventually becoming familiar with the associated proof-techniques.

I believe that too often researchers get caught, not by their fault, in a "publish or perish" situation in which finding the next result is more profitable in terms of their academic career than studying existing results. I believe it is important, when a research domain reaches a stable point of maturity, to take the time to organize the material in a clear and well-structured text, so that the whole community can benefit from the effort.

References

1 Andrea Asperti and Stefano Guerrini. *The Optimal Implementation of Functional Programming Languages*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998.

2 Andrea Asperti and Giuseppe Longo. *Categories, types and structures: an introduction to category theory for the working computer scientist*. MIT Press, Cambridge, MA, 1991.

3 Henk P. Barendregt. RTA list, Problem #97: Is the word problem for the S-combinator decidable?, 1975. See `https://www.win.tue.nl/rtaloop/problems/97.html`.

4 Henk P. Barendregt. The type free lambda calculus. In J. Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 1091–1132. North-Holland, Amsterdam, 1977.

5 Henk P. Barendregt. *The lambda-calculus, its syntax and semantics*. Number 103 in Studies in Logic and the Foundations of Mathematics. North-Holland, revised edition, 1984.

6 Henk P. Barendregt. Constructive proofs of the range property in lambda calculus. *Theoretical Computer Science*, 121(1–2):59–69, 1993. A collection of contributions in honour of Corrado Böhm on the occasion of his 70th birthday.

7 Henk P. Barendregt, Jan A. Bergstra, Jan Willem Klop, and Henri Volken. Degrees of sensible lambda theories. *J. Symb. Log.*, 43(1):45–55, 1978.

8 Henk P. Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *J. Symb. Log.*, 48(4):931–940, 1983. `doi:10.2307/2273659`.

9 Henk P. Barendregt and Giulio Manzonetto. *A Lambda Calculus Satellite*. College Publications, 2022. URL: `https://www.collegepublications.co.uk/logic/mlf/?00035`.

10 Alessandro Berarducci. Infinite λ-calculus and non-sensible models. In Marcel Dekker, editor, *Logic and Algebra (Pontignano, 1994)*, volume 180, pages 339–377, New York, 1996.

11 Corrado Böhm. Alcune proprietà delle forme β-η-normali nel λ-K-calcolo. *Pubblicazioni dell'istituto per le applicazioni del calcolo*, 696:1–19, 1968. Lavoro eseguito all'INAC.

12 Corrado Böhm and Mariangiola Dezani-Ciancaglini. Combinatorial problems, combinator equations and normal forms. In Jacques Loeckx, editor, *Automata, Languages and Programming, 2nd Colloquium, University of Saarbrücken, Germany, July 29 - August 2, 1974, Proceedings*, volume 14 of *Lecture Notes in Computer Science*, pages 185–199. Springer, 1974.

13 Corrado Böhm and Wolf Gross. Introduction to the CUCH. *Automata theory*, pages 35–65, 1966. Reprinted in Pubblicazioni dell'Istituto Nazionale per le Applicazioni del Calcolo, ser. 11 no. 669, Rome 1966.

14 Flavien Breuvart, Giulio Manzonetto, Andrew Polonsky, and Domenico Ruoppolo. New results on Morris's observational theory: The benefits of separating the inseparable. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, volume 52 of *LIPIcs*, pages 15:1–15:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. `doi:10.4230/LIPIcs.FSCD.2016.15`.

15 Antonio Bucciarelli and Thomas Ehrhard. Sequentiality and strong stability. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 138–145. IEEE Computer Society, 1991. `doi:10.1109/LICS.1991.151638`.

16 Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. Not enough points is enough. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 298–312. Springer, 2007. `doi:10.1007/978-3-540-74915-8_24`.

17 Felice Cardone and J. Roger Hindley. Lambda-calculus and combinators in the 20th century. In Dov M. Gabbay and John Woods, editors, *Logic from Russell to Church*, volume 5 of *Handbook of the History of Logic*, pages 723–817. Elsevier, 2009. `doi:10.1016/S1874-5857(09)70018-4`.

18 Alonzo Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 33(2):346–366, 1932.

19 Alonzo Church. A set of postulates for the foundation of logic (second paper). *Annals of Mathematics*, 34:839–864, 1933.

20 Alonzo Church and J. B. Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39(3):472–482, 1936.

21 Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980.

22 Mario Coppo, Mariangiola Dezani-Ciancaglini, and Maddalena Zacchi. Type theories, normal forms and $\mathcal{D}_\infty$-lambda-models. *Inf. Comput.*, 72(2):85–116, 1987. doi:10.1016/0890-5401(87)90042-3.

23 Haskell B. Curry and Robert Feys. *Combinatory logic. Volume I.* Number 1 in Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1958.

24 Lukasz Czajka. A new coinductive confluence proof for infinitary lambda calculus. *Log. Methods Comput. Sci.*, 16(1), 2020. doi:10.23638/LMCS-16(1:31)2020.

25 David Edward. *The Church-Rosser Theorem.* Ph.D. thesis, Cornell Univ., 1965. Informally circulated 1963. 673 pp. Obtainable from University Microfilms Inc., Ann Arbor, Michigan, U.S.A., Publication No. 66-41.

26 Jörg Endrullis and Roel de Vrijer. Reduction under substitution. In Andrei Voronkov, editor, *Rewriting Techniques and Applications, 19th International Conference, RTA 2008, Hagenberg, Austria, July 15-17, 2008, Proceedings*, volume 5117 of *Lecture Notes in Computer Science*, pages 425–440. Springer, 2008.

27 Jörg Endrullis, Dimitri Hendriks, Jan Willem Klop, and Andrew Polonsky. Clocked lambda calculus. *Math. Struct. Comput. Sci.*, 27(5):782–806, 2017. doi:10.1017/S0960129515000389.

28 Jörg Endrullis, Jan Willem Klop, and Andrew Polonsky. Reduction Cycles in Lambda Calculus and Combinatory Logic. In Jan van Eijck, Rosalie Iemhoff, and Joost J. Joosten, editors, *Liber Amicorum Alberti — A Tribute to Albert Visser*. College Publications, 2016.

29 Enno Folkerts. *Kongruenz von unlösbaren Lambda-Termen.* Ph.D. thesis, University WWU-Münster, 1995. In German.

30 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.

31 J. Roger Hindley and Giuseppe Longo. Lambda calculus models and extensionality. *Z. Math. Logik Grundlag. Math*, 26:289–310, 1980.

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

33 J. Martin E. Hyland. A syntactic characterization of the equality in some models for the λ-calculus. *Journal London Mathematical Society (2)*, 12(3):361–370, 1975.

34 Martin Hyland. A simple proof of the Church-Rosser Theorem. Technical report, Oxford University, 1973. 7 pp.

35 Benedetto Intrigila. Non-existent Statman's double fixedpoint combinator does not exist, indeed. *Inf. Comput.*, 137(1):35–40, 1997. doi:10.1006/inco.1997.2633.

36 Benedetto Intrigila. TLCA list, Problem #25: How many fixed points can a combinator have?, 2000. See http://tlca.di.unito.it/opltlca/. First raised in [37].

37 Benedetto Intrigila and E. Biasone. On the number of fixed points of a combinator in lambda calculus. *Mathematical Structures in Computer Science*, 10(5):595–615, 2000.

38 Benedetto Intrigila, Giulio Manzonetto, and Andrew Polonsky. Refutation of Sallé's longstanding conjecture. 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 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.FSCD.2017.20.

39 Benedetto Intrigila and Richard Statman. Solution of a problem of Barendregt on sensible λ-theories. *Log. Methods Comput. Sci.*, 2(4), 2006. doi:10.2168/LMCS-2(4:5)2006.

40 Richard Kennaway, Jan Willem Klop, M. Ronan Sleep, and Fer-Jan de Vries. Infinitary lambda calculus. *Theor. Comput. Sci.*, 175(1):93–125, 1997. `doi:10.1016/S0304-3975(96)00171-5`.

41 Jan Willem Klop. Reduction cycles in Combinatory Logic. In Hindley and Seldin, editors, *Essays on Combinatory Logic, Lambda-Calculus, and Formalism*, pages 193–214. Academic Press, San Diego, 1980.

42 Christiaan Peter Jozef Koymans. Models of the lambda calculus. *Inf. Control.*, 52(3):306–332, 1982. `doi:10.1016/S0019-9958(82)90796-3`.

43 Dexter Kozen and Alexandra Silva. Practical coinduction. *Math. Struct. Comput. Sci.*, 27(7):1132–1152, 2017.

44 Jean-Jacques Lévy. An algebraic interpretation of the $\lambda\beta$K-calculus; and an application of a labelled λ-calculus. *Theor. Comput. Sci.*, 2(1):97–114, 1976. `doi:10.1016/0304-3975(76)90009-8`.

45 Jean-Jacques Lévy. *Réductions correctes et optimales dans le λ-calcul.* Thèse d'état, Université Paris-Diderot (Paris 7), 1978. In French.

46 Stefania Lusin and Antonino Salibra. The lattice of lambda theories. *J. Log. Comput.*, 14(3):373–394, 2004.

47 Giulio Manzonetto, Andrew Polonsky, Alexis Saurin, and Jakob Grue Simonsen. The fixed point property and a technique to harness double fixed point combinators. *Journal of Logic and Computation*, 2019. `doi:10.1093/logcom/exz013`.

48 Giulio Manzonetto and Antonino Salibra. From lambda-calculus to universal algebra and back. In Edward Ochmanski and Jerzy Tyszkiewicz, editors, *Mathematical Foundations of Computer Science 2008, 33rd International Symposium, MFCS 2008, Torun, Poland, August 25-29, 2008, Proceedings*, volume 5162 of *Lecture Notes in Computer Science*, pages 479–490. Springer, 2008. `doi:10.1007/978-3-540-85238-4_39`.

49 James Hiram Morris. *Lambda calculus models of programming languages.* Ph.D. thesis, Massachusetts Institute of Technology (MIT), 1968.

50 Hans Mulder. On a conjecture by J.W. Klop. Unpublished, 1986.

51 Rob P. Nederpelt, J. Herman Geuvers, and Roel C. de Vrijer, editors. *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1994.

52 Gordon D. Plotkin. A set-theoretical definition of application. Technical Report MIP-R-95, School of artificial intelligence, 1971.

53 Gordon D. Plotkin. The lambda-calculus is ω-incomplete. *Journal of Symbolic Logic*, 39(2):313–317, 1974.

54 Andrew Polonsky. *Proofs, Types and Lambda Calculus.* Ph.D. thesis, University of Bergen, Norway, 2011.

55 Simona Ronchi Della Rocca and Luca Paolini. *The Parametric Lambda Calculus: A Metamodel for Computation.* Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, 2004.

56 Dana S. Scott. Continuous lattices. In Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, volume 274 of *Lecture Notes in Mathematics*, pages 97–136. Springer, 1972.

57 Dana S. Scott. The language LAMBDA (abstract). *J. Symb. Log.*, 39:425–427, 1974.

58 Shoji Sekimoto and Sachio Hirokawa. One-step recurrent terms in lambda-beta-calculus. *Theor. Comput. Sci.*, 56:223–231, 1988. `doi:10.1016/0304-3975(88)90079-5`.

59 Raymond Smullyan. *To Mock A Mockingbird.* Alfred A. Knopf, New York, 1985.

60 Richard Statman. There is no hyper–recurrent S,K combinator. Technical Report 91–133, Department of Mathematics, Carnegie Mellon University, Pittsburgh, PA, 1991.

61 Richard Statman. RTA list, Problem #52: Is there a fixed point combinator Y for which $Y \leftrightarrow^* Y(SI)$?, 1993. See `https://www.win.tue.nl/rtaloop/`. First raised in [63].

62 Richard Statman. RTA list, Problem #53: Are there hyper-recurrent combinators?, 1993. See `https://www.win.tue.nl/rtaloop/`. First raised in [63].

63 Richard Statman. Some examples of non-existent combinators. *Theor. Comput. Sci.*, 121(1&2):441–448, 1993.

64 Richard Statman and Henk P. Barendregt. Applications of Plotkin-terms: partitions and morphisms for closed terms. *Journal of Functional Programming*, 9(5):565–575, 1999.

65 Diederik van Daalen. *The Language Theory of Automath.* Ph.D. thesis, Technical University Eindhoven, 1980. Large parts of this thesis, including the treatment of RuS, have been reproduced in [51].

66 Albert Visser. Numerations, λ-calculus, and arithmetic. In Hindley and Seldin, editors, *Essays on Combinatory Logic, Lambda-Calculus, and Formalism*, pages 259–284. Academic Press, San Diego, 1980.

67 Christopher P. Wadsworth. The relation between computational and denotational properties for Scott's $\mathcal{D}_\infty$-models of the lambda-calculus. *SIAM J. Comput.*, 5(3):488–521, 1976. doi:10.1137/0205036.

68 Johannes Waldmann. *The Combinator S.* Ph.D. thesis, Friedrich-Schiller-Universität Jena, 1998.

Termination of Term Rewriting: Foundation, Formalization, Implementation, and Competition

Akihisa Yamada ✉
National Institute of Advanced Industrial Science and Technology, Tokyo, Japan

Abstract

Automated termination analysis is a central topic in the research of term rewriting. In this talk, I will first review the theoretical foundation of termination of term rewriting, leading to the recently established tuple interpretation method. Then I will present an Isabelle/HOL formalization of the theory. Although the formalization is based on the existing library IsaFoR (Isabelle Formalization of Rewriting), the present work takes another approach of representing relations (predicates rather than sets) so that the notation is more human friendly. Then I will present a unified implementation of the termination analysis techniques via SMT encoding, leading to the termination prover NaTT. Many tools have been developed for termination analysis and have been competing annually in termCOMP (Termination Competition) for two decades. At the end of the talk, I will share my experience in organizing termCOMP in the last five years.

2012 ACM Subject Classification Theory of computation → Rewrite systems; Theory of computation → Automated reasoning

Keywords and phrases Term rewriting, Termination, Isabelle/HOL, Competition

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.4

Category Invited Talk

1 Foundation

Ensuring the termination [10] of term rewrite systems (TRSs) has many important applications and actively studied for a half century. A foundational way of ensuring termination used to be *reduction orders*, in short, overestimating the rewrite step by a term ordering whose termination (well-foundedness) is known. Later, the dependency pair method [1] generalized reduction orders to *reduction pairs* [1, 17, 15], where, in short, one part overestimates the rewrite step and the other part ensures termination.

The semantic method (interpretation method) [29] for defining reduction orders/pairs uses universal algebras, mapping terms into a set with a known terminating relation, such as $\langle \mathbb{N}, > \rangle$. Instances of semantic approaches are polynomial interpretations [27], matrix interpretations [18, 11], max-polynomial interpretations [12], and arctic interpretations [24]. Tuple interpretations [43, 23] encompass these instances, and the use of *derivers* [41, 28] allows to uniformly prove monotonicity (essential for overestimating rewrite steps) [43].

The syntactic approach such as recursive path ordering [9] and lexicographic path ordering [21] analyzes the term structure purely syntactically, and usually ensure termination by Kruskal's tree theorem [26]. The Knuth-Bendix ordering [22] mixes the semantic and syntactic approaches. The weighted path order (WPO) [45] gives a unified way of capturing syntactic and semantic methods.

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 4; pp. 4:1–4:5
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2 Formalization

There are number of methods for ensuring termination, and they are implemented in automated termination analysis tools such as AProVE [13], TTT2 [25], matchbox [40], MU-TERM [16], etc. However, it cannot be taken for granted that the implementations and the paper proofs are correct. Therefore there have been efforts in formally verifying the paper proofs and tool outputs using proof assistants such as Coq [4] and Isabelle/HOL [30].

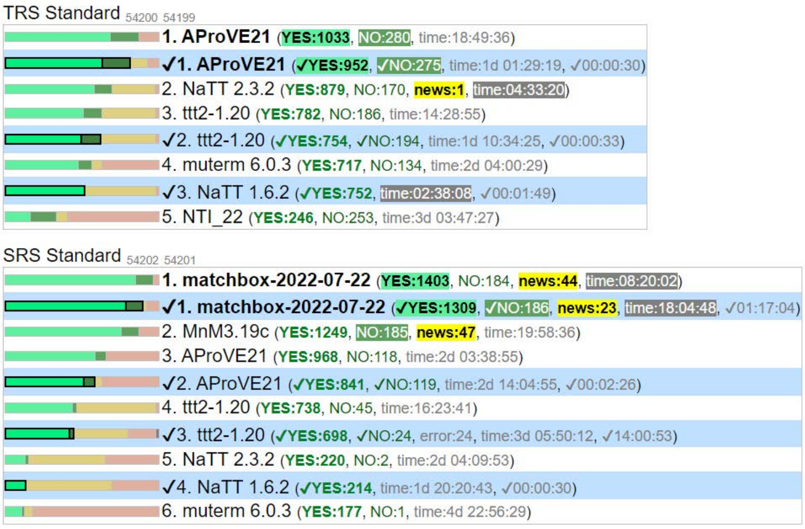
Coccinelle/CiME3 [8] and CoLoR/Rainbow [5] are pairs whose first components are Coq libraries of proofs for termination methods, and the second components are tools that translate termination tool's outputs into Coq theories whose validity are then checked by Coq. IsaFoR/CeTA [38] is another pair, whose first component is an Isabelle/HOL library of termination methods (and more), and the second is a tool that checks the tool outputs. The correctness of the latter is verified by Isabelle; therefore achieves the same level of assurance as the Coq ones without needing to execute the proof assistants in the run time. In this talk I will present a recent development based on IsaFoR for sorted term rewriting and algebras, which formalizes the correctness of the deriver-based tuple interpretation method.

3 Implementation

NaTT started as an OCaml [31] code for experimenting WPO. This implementation follows the "lazy" approach [47, 32, 48, 7] and delegates "solving" tasks to SMT solvers [3]. The code was turned into a full termination prover by adding other basic techniques and improving efficiency by being more lazy in calling SMT solvers [44]: for instance, in the SMT interface of NaTT, the OCaml code `x *^ Delay (fun _ -> hardFormula)` represents an SMT formula for x multiplied by *hardFormula*, but *hardFormula* will not be evaluated if $x = 0$ is known. NaTT participates in the Termination Competition (since "full run" of 2013) and keeps winning the second place in the TRS Standard category after the champion, AProVE. Since then more techniques are implemented into NaTT [19, 46, 2, 35, 43, 42], making the implementation harder and harder to maintain. In the meantime, the C++ programming language kept evolving, and hence I fully reimplement NaTT in the modern C++20 [20]. For instance, in the new C++ implementation, the above lazy formula is achieved by `x * []{ return hardFormula; }`.

4 Competition

At the International Workshop on Termination (WST) 2003, Albert Rubio organized a session where termination tool developers demonstrate their tools on problems written on a blackboard. The community decided to turn this event into the International Termination Competition (termCOMP) [14]. Benchmarks are collected, using an agreed textual format called the WST format [6], into the Termination Problem DataBase (TPDB) [39]. Editions from 2004 to 2007 were run on local servers organized by Claude Marché, and from 2008 to 2013 are by René Thiemann. In 2009, the benchmark format has been changed to an XML format (see [39]). From 2014 to 2017 are organized by Johannes Waldmann, who migrated the tool execution platform to the StarExec environment [36], and wrote the web server code, star-exec-presenter [33], using Haskell and SQL. In 2018 I succeeded the organization and developed starexec-master [34] web frontend in pure PHP (so that most web servers and engineers can understand). Since then I have gradually improve visuals and collected metrics (Figure 1 on page 3), keep past results in a public repository [37], and migrated TPDB also into GitHub. In the talk I would like to share some lessons learned by organizing the competition.



TRS Standard 54200 54199

1. AProVE21 (YES:1033, NO:280, time:18:49:36)
✓1. AProVE21 (✓YES:952, ✓NO:275, time:1d 01:29:19, ✓00:00:30)
2. NaTT 2.3.2 (YES:879, NO:170, news:1, time:04:33:20)
3. ttt2-1.20 (YES:782, NO:186, time:14:28:55)
✓2. ttt2-1.20 (✓YES:754, ✓NO:194, time:1d 10:34:25, ✓00:00:33)
4. muterm 6.0.3 (YES:717, NO:134, time:2d 04:00:29)
✓3. NaTT 1.6.2 (✓YES:752, time:02:38:08, ✓00:01:49)
5. NTI_22 (YES:246, NO:253, time:3d 03:47:27)

SRS Standard 54202 54201

1. matchbox-2022-07-22 (YES:1403, NO:184, news:44, time:08:20:02)
✓1. matchbox-2022-07-22 (✓YES:1309, ✓NO:186, news:23, time:18:04:48, ✓01:17:04)
2. MnM3.19c (YES:1249, NO:185, news:47, time:19:58:36)
3. AProVE21 (YES:968, NO:118, time:2d 03:38:55)
✓2. AProVE21 (✓YES:841, ✓NO:119, time:2d 14:04:55, ✓00:02:26)
4. ttt2-1.20 (YES:738, NO:45, time:16:23:41)
✓3. ttt2-1.20 (✓YES:698, ✓NO:24, error:24, time:3d 05:50:12, ✓14:00:53)
5. NaTT 2.3.2 (YES:220, NO:2, time:2d 04:09:53)
✓4. NaTT 1.6.2 (✓YES:214, time:1d 20:20:43, ✓00:00:30)
6. muterm 6.0.3 (YES:177, NO:1, time:4d 22:56:29)

Figure 1 An excerpt of results of termCOMP 2022. Rows with check mark indicates the certified configurations. The "news" scores indicate improvements over the virtual best solver of the past, that is, the number of open problems closed by the solver this year. Those scores were used to award the special "advancing-the-state-of-the-art" medals.

References

1 T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theor. Compt. Sci.*, 236(1-2):133–178, 2000. doi:0.1016/S0304-3975(99)00207-8.

2 Martin Avanzini, Ugo Dal Lago, and Akihisa Yamada. On probabilistic term rewriting. *Sci. Comput. Program.*, 185, 2020. doi:10.1016/j.scico.2019.102338.

3 Clark W. Barrett and Cesare Tinelli. Satisfiability modulo theories. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 305–343. Springer, 2018. doi:10.1007/978-3-319-10575-8_11.

4 Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

5 Frédéric Blanqui and Adam Koprowski. CoLoR: a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates. *Math. Struct. Comput. Sci.*, 21(4):827–859, 2011. doi:10.1017/S0960129511000120.

6 H. Zantema C. Marché, A. Rubio. Termination problem data base: format of input files, 2005. URL: https://www.lri.fr/~marche/tpdb/format.html.

7 M. Codish, J. Giesl, P. Schneider-Kamp, and R. Thiemann. SAT solving for termination proofs with recursive path orders and dependency pairs. *J. Autom. Reasoning*, 49(1):53–93, 2012.

8 Evelyne Contejean, Pierre Courtieu, Julien Forest, Olivier Pons, and Xavier Urbain. Automated certified proofs with cime3. In Manfred Schmidt-Schauß, editor, *RTA 2011*, volume 10 of *LIPIcs*, pages 21–30. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.RTA.2011.21.

9 Nachum Dershowitz. Orderings for term-rewriting systems. *Theor. Compt. Sci.*, 17(3):279–301, 1982. doi:10.1016/0304-3975(82)90026-3.

10 Nachum Dershowitz. Termination of rewriting. *J. Symb. Comput.*, 3(1–2):69–115, 1987.

11 Jörg Endrullis, Johannes Waldmann, and Hans Zantema. Matrix interpretations for proving termination of term rewriting. *J. Autom. Reason.*, 40(2-3):195–220, 2008. doi:10.1007/s10817-007-9087-9.

12 Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl. Maximal termination. In Andrei Voronkov, editor, *RTA 2008*, volume 5117 of *LNCS*, pages 110–125. Springer, 2008. doi:10.1007/978-3-540-70590-1_8.

13 Jürgen Giesl, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann. Proving termination of programs automatically with AProVE. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *IJCAR 2014*, volume 8562 of *LNCS*, pages 184–191. Springer, 2014. doi:10.1007/978-3-319-08587-6_13.

14 Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. The termination and complexity competition. In Dirk Beyer, Marieke Huisman, Fabrice Kordon, and Bernhard Steffen, editors, *TACAS 2019, Part III*, volume 11429 of *LNCS*, pages 156–166. Springer, 2019. doi:10.1007/978-3-030-17502-3_10.

15 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. Mechanizing and improving dependency pairs. *J. Autom. Reason.*, 37(3):155–203, 2006. doi:10.1007/s10817-006-9057-7.

16 Raúl Gutiérrez and Salvador Lucas. mu-term: Verify termination properties automatically (system description). In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *IJCAR 2020 (2)*, volume 12167 of *LNCS*, pages 436–447. Springer, 2020. doi:10.1007/978-3-030-51054-1_28.

17 Nao Hirokawa and Aart Middeldorp. Dependency pairs revisited. In Vincent van Oostrom, editor, *RTA 2004*, volume 3091 of *LNCS*, pages 249–268. Springer, 2004. doi:10.1007/978-3-540-25979-4_18.

18 Dieter Hofbauer and Johannes Waldmann. Termination of string rewriting with matrix interpretations. In Frank Pfenning, editor, *RTA 2006*, volume 4098 of *LNCS*, pages 328–342. Springer, 2006. doi:10.1007/11805618_25.

19 José Iborra, Naoki Nishida, Germán Vidal, and Akihisa Yamada. Relative termination via dependency pairs. *J. Autom. Reason.*, 58(3):391–411, 2017. doi:10.1007/s10817-016-9373-5.

20 Programming languages: C++, ISO/IEC 14882:2020.

21 Sam Kamin and Jean-Jacques Lévy. Two generalizations of the recursive path ordering, 1980. Unpublished note.

22 Donald E. Knuth and Peter Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, New York, 1970. doi:10.1016/B978-0-08-012975-4.50028-X.

23 Cynthia Kop and Deivid Vale. Tuple interpretations for higher-order complexity. In Naoki Kobayashi, editor, *FSCD 2021*, volume 195 of *LIPIcs*, pages 31:1–31:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSCD.2021.31.

24 Adam Koprowski and Johannes Waldmann. Max/plus tree automata for termination of term rewriting. *Acta Cybern.*, 19(2):357–392, 2009.

25 Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean Termination Tool 2. In Ralf Treinen, editor, *RTA 2009*, volume 5595 of *LNCS*, pages 295–304. Springer, 2009. doi:10.1007/978-3-642-02348-4_21.

26 J.B. Kruskal. Well-quasi-ordering, the tree theorem, and Vazsonyia's conjecture. *Transactions of the American Mathematical Society*, 95(2):210–225, 1960.

27 D. Lankford. Canonical algebraic simplification in computational logic. Technical Report ATP-25, University of Texas, 1975.

28 Salvador Lucas and Raúl Gutiérrez. Automatic synthesis of logical models for order-sorted first-order theories. *J. Autom. Reason.*, 60(4):465–501, 2018. doi:10.1007/s10817-017-9419-3.

29 Z. Manna and S. Ness. On the termination of Markov algorithms. In *the 3rd Hawaii International Conference on System Science*, pages 789–792, 1970.

30 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002. doi:10.1007/3-540-45949-9.

31 Ocaml. URL: https://v2.ocaml.org/.

32 Peter Schneider-Kamp, René Thiemann, Elena Annov, Michael Codish, and Jürgen Giesl. Proving termination using recursive path orders and SAT solving. In Boris Konev and Frank Wolter, editors, *FroCoS 2007*, volume 4720 of *LNCS*, pages 267–282. Springer, 2007. doi:10.1007/978-3-540-74621-8_18.

33 star-exec-presenter (software), 2014. URL: https://github.com/jwaldmann/star-exec-presenter/.

34 starexec-master (software). URL: https://github.com/TermCOMP/starexec-master.

35 Christian Sternagel and Akihisa Yamada. Reachability analysis for termination and confluence of rewriting. In Tomás Vojnar and Lijun Zhang, editors, *TACAS (1) 2019*, volume 11427 of *LNCS*, pages 262–278. Springer, 2019. doi:10.1007/978-3-030-17462-0_15.

36 Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. StarExec: A cross-community infrastructure for logic solving. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *IJCAR 2014*, volume 8562 of *LNCS*, pages 367–373. Springer, 2014. doi:10.1007/978-3-319-08587-6_28.

37 Termination competitions results (repository). URL: https://termcomp.github.io/.

38 René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *TPHOLs 2009*, volume 5674 of *LNCS*, pages 452–468. Springer, 2009. doi:10.1007/978-3-642-03359-9_31.

39 The termination problem data base. URL: http://termination-portal.org/wiki/TPDB.

40 Johannes Waldmann. Matchbox: A tool for match-bounded string rewriting. In Vincent van Oostrom, editor, *RTA 2004*, volume 3091 of *LNCS*, pages 85–94. Springer, 2004. doi:10.1007/978-3-540-25979-4_6.

41 T.J. Watson, J.A. Goguen, J.W. Thatcher, and E.G. Wagner. An initial algebra approach to the specification, correctness, and implementation of abstract data types. In *Current Trends in Programming Methodology*. Prentice Hall, 1976.

42 Akihisa Yamada. Term orderings for non-reachability of (conditional) rewriting. In Jasmin Blanchette, Laura Kovács, and Dirk Pattinson, editors, *IJCAR 2022*, volume 13385 of *LNCS*, pages 248–267. Springer, 2022. doi:10.1007/978-3-031-10769-6_15.

43 Akihisa Yamada. Tuple interpretations for termination of term rewriting. *J. Autom. Reason.*, 66(4):667–688, 2022. doi:10.1007/s10817-022-09640-4.

44 Akihisa Yamada, Keiichirou Kusakari, and Toshiki Sakabe. Nagoya termination tool. In Gilles Dowek, editor, *RTA-TLCA 2014*, volume 8560 of *LNCS*, pages 466–475. Springer, 2014. doi:10.1007/978-3-319-08918-8_32.

45 Akihisa Yamada, Keiichirou Kusakari, and Toshiki Sakabe. A unified ordering for termination proving. *Sci. Comput. Program.*, 111:110–134, 2015. doi:10.1016/j.scico.2014.07.009.

46 Akihisa Yamada, Christian Sternagel, René Thiemann, and Keiichirou Kusakari. AC dependency pairs revisited. In Jean-Marc Talbot and Laurent Regnier, editors, *CSL 2016*, volume 62 of *LIPIcs*, pages 8:1–8:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CSL.2016.8.

47 Harald Zankl. *Lazy Termination Analysis*. PhD thesis, University of Innsbruck, 2009.

48 Harald Zankl, Nao Hirokawa, and Aart Middeldorp. KBO orientability. *J. Autom. Reasoning*, 43(2):173–201, 2009.

Homotopy Type Theory as Internal Languages of Diagrams of ∞-Logoses

Taichi Uemura
Stockholm University, Sweden

Abstract

We show that certain diagrams of ∞-logoses are reconstructed in internal languages of their oplax limits via lex, accessible modalities, which enables us to use plain homotopy type theory to reason about not only a single ∞-logos but also a diagram of ∞-logoses. This also provides a higher dimensional version of Sterling's synthetic Tait computability – a type theory for higher dimensional logical relations.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Denotational semantics; Theory of computation → Categorical semantics

Keywords and phrases Homotopy type theory, ∞-logos, ∞-topos, oplax limit, Artin gluing, modality, synthetic Tait computability, logical relation

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.5

Related Version *Full Version*: `https://arxiv.org/abs/2212.02444` [33]

Funding *Taichi Uemura*: Supported by KAW Grant "Type Theory for Mathematics and Computer Science" investigated by Thierry Coquand and Peter LeFanu Lumsdaine.

Acknowledgements The author thanks Jonathan Sterling for useful conversations on the current work. The author also thanks anonymous referees for corrections, comments, and suggestions.

1 Introduction

Homotopy type theory [31] is a type theory where one can do homotopy theory. It extends Martin-Löf's type theory [19] by the *univalence axiom* and *higher inductive types*. The former forces types to behave like spaces rather than sets, and the latter allow us to build types representing spaces such as spheres and tori.

An ∞-*logos*, also known as an ∞-*topos* [16, 2][1], is another place to do homotopy theory, among other aspects of it. An ∞-logos is an $(\infty, 1)$-category that looks like the $(\infty, 1)$-category of spaces just as an ordinary logos is a category that looks like the category of sets.

Homotopy type theory and ∞-logoses are closely related. Shulman [26] has shown that any ∞-logos is presented by a structure that admits an interpretation of homotopy type theory. In other words, homotopy type theory is an *internal language* of an ∞-logos. Any theorem proved in homotopy type theory can be translated in an arbitrary ∞-logos. For example, the proof of the Blakers-Massey connectivity theorem in homotopy type theory [10] has led to a new generalized Blakers-Massey theorem that holds in an arbitrary ∞-logos [1].

An ∞-logos, however, does not live alone. ∞-logoses are often connected by functors which are also connected by natural transformations. Plain homotopy type theory is, at first sight, not sufficient to reason about a diagram of ∞-logoses, because the actions of

[1] The term ∞-logos is Anel and Joyal's terminology [2] for ∞-topos considered as an algebraic structure rather than a geometric object. A morphism of ∞-logoses is always considered in the direction of the inverse image functor. We use this terminology to clarify the direction of morphisms when speaking about (co)limits of ∞-logoses.

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 5; pp. 5:1–5:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the functors and natural transformations are not internalized to type theory. Even worse, it is impossible to naively internalize some diagrams: some internal adjunction leads a contradiction [14]; there are only trivial internal idempotent comonads [25].

While there is no chance of naive internalization of such interesting but problematic diagrams to plain homotopy type theory, some other diagrams can be internalized in a clever way pointed out by Shulman[2]. A minimal non-trivial example is a diagram consisting of two ∞-logoses and a lex, accessible functor between them in one direction. The two ∞-logoses are *lex, accessible localizations* of another ∞-logos obtained by the *Artin gluing* for the functor, and the functor is reconstructed by composing the inclusion from one localization and the reflector to the other. Moreover, this reconstruction is *internal* to the glued ∞-logos, because lex, accessible localizations of an ∞-logos correspond to *lex, accessible modalities* in its internal language. Hence, plain homotopy type theory as an internal language of the glued ∞-logos is sufficient to reason about the original diagram.

In this paper, we propose a class of shapes of diagrams of ∞-logoses for which the internal reconstruction technique explained in the previous paragraph works. We call shapes in the proposed class *mode sketches.* Our main result is summarized as follows. Let $\mathfrak{M}$ be a mode sketch. $\mathfrak{M}$ is regarded as a presentation of an $(\infty, 2)$-category. Then:

1. We associate to $\mathfrak{M}$ certain axioms in type theory, one of which is to postulate some lex, accessible modalities from which one can construct a diagram of ∞-logoses internally to type theory (Sections 3.1 and 3.2).
2. For any diagram $\mathcal{L}$ indexed over $\mathfrak{M}$ consisting of ∞-logoses and lex, accessible functors, the oplax limit of $\mathcal{L}$ is an ∞-logos that satisfies the axioms associated to $\mathfrak{M}$, and the diagram obtained in the internal language of the oplax limit corresponds to the original diagram $\mathcal{L}$. Conversely, any ∞-logos that satisfies the axioms associated to $\mathfrak{M}$ is obtained by this oplax limit construction (Theorem 48).

Recall that the *oplax limit* of a diagram of $(\infty, 1)$-categories is a generalization of the Artin gluing [35, 24].

1.1 Synthetic Tait computability

This work is closely related to Sterling's *synthetic Tait computability* [29, 27]. It is a technique of constructing *logical relations* using an internal language for the Artin gluing. A logos obtained by the Artin gluing is always equipped with a distinguished proposition in its internal language. The two lex, accessible modalities associated to the glued logos are the *open* and *closed* modalities associated to the proposition. The *fracture and gluing theorem* asserts that every type in the internal language is canonically fractured into an open type and a closed (unary, proof-relevant) relation on it which are glued back together. The internal language for the Artin gluing is thus a type theory with an indeterminate proposition in which *types are relations* and provides a synthetic method of constructing logical relations used in the study of type theories and programming languages. Applications include normalization for complex type theories [28, 8].

We relate synthetic Tait computability and mode sketches. The core axiom for synthetic Tait computability is to postulate some indeterminate propositions. Note that, although the original synthetic Tait computability is based on extensional type theory, postulating propositions makes sense also in homotopy type theory. We show that part of the axioms associated to a mode sketch is equivalent to postulating a lattice of propositions (Theorem 37).

[2] `https://golem.ph.utexas.edu/category/2011/11/internalizing_the_external_or.html`

Mode sketches thus provide an alternative synthetic method of constructing logical relations. This is also natural from Shulman's point of view [24] that interpretations of type theory in oplax limits are generalized logical relations. Since we work in homotopy type theory, what we get is actually *higher-dimensional logical relations*, and our primary application of mode sketches in upcoming paper(s) [34] will be normalization for ∞-*type theories* introduced by Nguyen and Uemura [20] as a higher-dimensional generalization of type theories.

1.2 Contributions

Our main result is Theorem 48: for every mode sketch $\mathfrak{M}$, the models of the axioms associated to $\mathfrak{M}$ are precisely the oplax limits of diagrams of ∞-logoses indexed over $\mathfrak{M}$. This allows us to reason about a diagram of ∞-logoses in plain homotopy type theory. We also relate mode sketches to synthetic Tait computability (Theorem 37). Mode sketches provide a higher-dimensional version of synthetic Tait computability.

A minor result is an improvement of the *fracture and gluing theorem* of Rijke, Shulman, and Spitters [22, Theorem 3.50]. It gives a construction of a canonical join of two strongly disjoint lex modalities. We show that this construction preserves accessibility as well (Proposition 14).

1.3 Organization

In Section 2, we review the theory of modalities in homotopy type theory [22]. Our focus is on the poset of lex, accessible modalities and on the open and closed modalities associated to propositions.

Sections 3 and 4 are the core of the paper. We introduce the notion of a *mode sketch* (Definition 25). For every mode sketch, we introduce two equivalent sets of axioms to encode a certain diagram of universes. One postulates some lex, accessible modalities while the other postulates a lattice of propositions. The open and closed modalities give a construction of the former from the latter which we show is an equivalence (Theorem 37). The latter is a higher dimensional analogue of Sterling's synthetic Tait computability [27].

In Section 5, we give a sketch of proof of our main result (Theorem 48): for any mode sketch, the space of ∞-logoses satisfying the axioms associated to the mode sketch is equivalent to the space of diagrams of ∞-logoses and lex, accessible functors indexed over the mode sketch. For reasons of space, details are not presented in this version. See [33] for full details.

1.4 Preliminaries

We assume that the reader is familiar with homotopy type theory [31]. By *homotopy type theory* we mean dependent type theory with (dependent) function types, (dependent) pair types, a unit type, identity types, univalent universes $\mathcal{U} : \Uparrow\mathcal{U} : \Uparrow^2\mathcal{U} : \dots$, and all higher inductive types we need. The universe $\Uparrow^n\mathcal{U}$ is usually written as $\mathcal{U}_n$, but the latter conflicts with the notation for the subuniverse of modal types. The notation $\Uparrow^n\mathcal{U}$ also indicates that large types are interpreted in universe enlargements of an ∞-logos; see Section 5. We mainly follow the HoTT Book [31] for terminologies and notations in homotopy type theory.

1.5 Related work

An earlier version of *cohesive homotopy type theory* [23] uses modalities in plain homotopy type theory to internalize a series of adjunctions that arises in Lawvere's axiomatic cohesion [13]. However, because naive internalization of adjunctions do not work well [14, 25], the axiomatization is tricky and not ideal to work with. The newer version of cohesive homotopy type theory [25] instead extends homotopy type theory by another layer of context and new modal operators. The resulting type theory works well for axiomatic cohesion but is complicated compared to plain homotopy type theory. It is also too optimized for axiomatic cohesion.

A more general framework for internal diagrams is *multimodal dependent type theory* [9]. It is roughly a family of type theories related to each other via modal operators and interpreted in a diagram of presheaf categories among a more general notion of model. The shape of diagram is specified directly by an arbitrary 2-category which is called a *mode theory* in this context. Our terminology "mode sketch" is chosen to mean a sketch of a mode theory. Multimodal dependent type theory is potentially an internal language for diagrams of ∞-logoses, but for this one would have to rectify not only ∞-logoses but also functors and natural transformations between them.

Our work brings back the ideas of earlier cohesive homotopy type theory [23]. Although it might not be the best type theory, it has a lot of advantages: modalities are internal to plain homotopy type theory, and thus all results are ready to formalize in existing proof assistants; keeping type theory simple is also important in informal use of type theory in which the correctness of application of inference rules is not checked by computer; the semantics is clear, since the ∞-logos semantics of homotopy type theory is well-established [4, 3, 24, 26]; it also opens the door to internalization of more general diagrams in a uniform way, which is the motivation for the current work.

2 Modalities in homotopy type theory

We review the theory of *modalities* in homotopy type theory [22]. In this section, we work in homotopy type theory. A modality is in short a reflective subuniverse closed under pair types.

▶ **Definition 1.** *A* subuniverse $\mathfrak{m}$ *is a a function* $\mathsf{In}_{\mathfrak{m}} : \mathcal{U} \to \Uparrow\mathcal{U}$ *such that* $\mathsf{In}_{\mathfrak{m}}(A)$ *is a proposition for all* $A : \mathcal{U}$*. A type* A *satisfying* $\mathsf{In}_{\mathfrak{m}}(A)$ *is called* $\mathfrak{m}$-modal*. We define a subtype* $\mathcal{U}_{\mathfrak{m}} \subset \mathcal{U}$ *to be* $\{A : \mathcal{U} \mid \mathsf{In}_{\mathfrak{m}}(A)\}$*.*

▶ **Definition 2.** *A subuniverse* $\mathfrak{m}$ *is* reflective *if it is equipped with functions* $\bigcirc_{\mathfrak{m}} : \mathcal{U} \to \mathcal{U}_{\mathfrak{m}}$ *and* $\eta_{\mathfrak{m}} : \prod_{A:\mathcal{U}} A \to \bigcirc_{\mathfrak{m}} A$ *such that that the precomposition* $\lambda f.f \circ \eta_{\mathfrak{m}}(A) : (\bigcirc_{\mathfrak{m}} A \to B) \to (A \to B)$ *is an equivalence for any* $B : \mathcal{U}_{\mathfrak{m}}$*. Note that such a pair* $(\bigcirc_{\mathfrak{m}}, \eta_{\mathfrak{m}})$ *is unique.*

▶ **Definition 3.** *A reflective subuniverse* $\mathfrak{m}$ *is a* modality *if* $\mathsf{In}_{\mathfrak{m}}$ *is closed under pair types, that is, for* $A : \mathcal{U}$ *and* $B : A \to \mathcal{U}$*, if* $\mathsf{In}_{\mathfrak{m}}(A)$ *and* $\prod_{a:A} \mathsf{In}_{\mathfrak{m}}(B(a))$*, then* $\mathsf{In}_{\mathfrak{m}}(\sum_{a:A} B(a))$*.*

An important class of modalities is *accessible* modalities which are roughly modalities "presented by small data".

▶ **Definition 4.** *For types* $A, B : \mathcal{U}$*, we define*

$$(A \perp B) \equiv \mathsf{IsEquiv}(\lambda(b : B).\lambda(\_ : A).b).$$

Note that $\lambda(b : B).\lambda(\_ : A).b$ *is a function of type* $B \to (A \to B)$*. For a subuniverse* $\mathfrak{m}$*, we define subuniverses* $\mathfrak{m}^{\perp}$ *and* ${}^{\perp}\mathfrak{m}$ *by*

$$\mathsf{In}_{\mathfrak{m}^{\perp}}(B) \equiv \prod_{A:\mathcal{U}_{\mathfrak{m}}} A \perp B$$
$$\mathsf{In}_{{}^{\perp}\mathfrak{m}}(A) \equiv \prod_{B:\mathcal{U}_{\mathfrak{m}}} A \perp B.$$

▶ **Definition 5.** *A* null generator μ *consists of* $I_\mu : \mathcal{U}$ *and* $Z_\mu : I_\mu \to \mathcal{U}$*. We write* $\mathsf{NullGen}$ *for the type of null generators. Given a null generator* μ*, we define a subuniverse* $\mathfrak{Null}(\mu)$ *by* $\mathsf{In}_{\mathfrak{Null}(\mu)}(A) \equiv \prod_{i:I_\mu} Z_\mu(i) \perp A$*. It is shown that* $\mathfrak{Null}(\mu)$ *is a modality using a higher inductive type [22, Theorem 2.19]. A modality* $\mathfrak{m}$ *is* accessible *if it is in the image of* $\mathfrak{Null}$*, that is,* $\| \sum_{\mu:\mathsf{NullGen}} \mathfrak{m} = \mathfrak{Null}(\mu) \|$.

Another important class of modalities is *lex* modalities.

▶ **Definition 6.** *For a modality* $\mathfrak{m}$*, a type* $A : \mathcal{U}$ *is* $\mathfrak{m}$-connected *if* $\bigcirc_{\mathfrak{m}} A$ *is contractible. This is equivalent to* $\mathsf{In}_{\perp\mathfrak{m}}(A)$ *by [22, Corollary 1.37].*

▶ **Definition 7.** *A modality* $\mathfrak{m}$ *is* lex *if for any* $\mathfrak{m}$*-connected type* $A : \mathcal{U}$*, the identity type* $a_1 = a_2$ *is* $\mathfrak{m}$*-connected for any* $a_1, a_2 : A$.

Modalities that are both lex and accessible are of particular importance because they correspond to subtoposes of an ∞-topos under the interpretation of types as sheaves on the ∞-topos. From now on, we are mostly interested lex, accessible modalities, so we give them a short name.

▶ **Terminology 8.** LAM is an acronym for lex, accessible modality.

Fundamental examples of LAMs are *open* and *closed* modalities which correspond to open and closed, respectively, subtoposes.

▶ **Construction 9.** Let P be a proposition. We define the *open modality* $\mathfrak{Op}(P)$ by $\bigcirc_{\mathfrak{Op}(P)} A \equiv (P \to A)$ and $\eta_{\mathfrak{Op}(P)}(A, a) \equiv \lambda\_.a$. It is lex and accessible by [22, Example 2.24 and Example 3.10]. We also define the *closed modality* $\mathfrak{Cl}(P)$ by $\mathsf{In}_{\mathfrak{Cl}(P)}(A) \equiv (P \to \mathsf{IsContr}(A))$. It is lex and accessible by [22, Example 2.25 and Example 3.14]. Note that $\mathfrak{Cl}(P) = {}^{\perp}\mathfrak{Op}(P)$ [22, Example 1.31].

2.1 The poset of lex, accessible modalities

We have the posets

$$\mathsf{SU} \supset \mathsf{RSU} \supset \mathsf{Mdl} \supset \mathsf{AccMdl} \supset \mathsf{LAM}$$

of subuniverses, reflective subuniverses, modalities, accessible modalities, and lex, accessible modalities, respectively, where all the inclusions are full. We also have the full subposet $\mathsf{Lex} \subset \mathsf{Mdl}$ of lex modalities. By definition, $\mathsf{LAM} = \mathsf{Lex} \cap \mathsf{AccMdl}$. We study the poset LAM in more detail.

▶ **Definition 10** ([22, Theorem 3.25])**.** *Let* $I : \mathcal{U}$ *and* $\mathfrak{m} : I \to \mathsf{LAM}$*. A* canonical meet $\bigwedge_{i:I} \mathfrak{m}(i)$ *is a LAM that is the meet of* $\mathfrak{m}(i)$*'s in* SU*. A* canonical join $\bigvee_{i:I} \mathfrak{m}(i)$ *is a LAM satisfying that a type* $A : \mathcal{U}$ *is* $(\bigvee_{i:I} \mathfrak{m}(i))$*-connected if and only if it is* $\mathfrak{m}(i)$*-connected for all* $i : I$*. Note that a canonical join is the join in* Mdl.

▶ **Example 11.** The *top modality* $\mathfrak{Top}$, for which all the types are modal, is the canonical meet of the empty family. The *bottom modality* $\mathfrak{Bot}$, for which only the contractible types are modal, is the canonical join of the empty family.

The canonical meet of an arbitrary family of LAMs exists [22, Theorem 3.29 and Remark 3.23]. Canonical joins are less understood than canonical meets. One important case when canonical joins exist and can be computed is the following.

▶ **Definition 12.** *Let* $\mathfrak{m}$ *and* $\mathfrak{n}$ *be LAMs.* $\mathfrak{n}$ *is* strongly disjoint from $\mathfrak{m}$ *if any* $\mathfrak{m}$*-modal type is* $\mathfrak{n}$*-connected or equivalently if* $\mathfrak{m} \leq {}^{\perp}\mathfrak{n}$ *in* SU*.*

▶ **Proposition 13** (Fracture and gluing theorem)**.** *Let* $\mathfrak{m}$ *and* $\mathfrak{n}$ *be LAMs such that* $\mathfrak{m} \leq {}^{\perp}\mathfrak{n}$*.*

1. *The canonical join* $\mathfrak{m} \vee \mathfrak{n}$ *exists.*
2. *A type* A *is* $(\mathfrak{m} \vee \mathfrak{n})$*-modal if and only if the function* $\eta_{\mathfrak{n}}(A) : A \to \bigcirc_{\mathfrak{n}} A$ *has* $\mathfrak{m}$*-modal fibers.*
3. $\mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}} \simeq \sum_{A:\mathcal{U}_{\mathfrak{m}}} \sum_{B:\mathcal{U}_{\mathfrak{n}}} A \to \bigcirc_{\mathfrak{m}} B$*.*

In the special case when $\mathfrak{m} = {}^{\perp}\mathfrak{n}$*, we have* $\mathfrak{m} \vee \mathfrak{n} = \mathfrak{Top}$*.*

Proof. All but the accessibility of $\mathfrak{m} \vee \mathfrak{n}$ are proved by Rijke, Shulman, and Spitters [22, Theorem 3.50]. We will prove the accessibility of $\mathfrak{m} \vee \mathfrak{n}$ in Proposition 14 using an open modality. ◀

2.2 Accessibility of the canonical join

Let us fill the gap in the proof of Proposition 13. This subsection is devoted to proving the following.

▶ **Proposition 14.** *Let* $\mathfrak{m}$ *and* $\mathfrak{n}$ *be LAMs such that* $\mathfrak{m} \leq {}^{\perp}\mathfrak{n}$*. Then the canonical join* $\mathfrak{m} \vee \mathfrak{n}$ *(in* Lex*) is accessible.*

We have to find a null generator for $\mathfrak{m} \vee \mathfrak{n}$. A natural guess is the following.

▶ **Construction 15.** Let μ and ν be null generators. We define a null generator $\mu \star \nu$ by $I_{\mu\star\nu} \equiv I_\mu \times I_\nu$ and $Z_{\mu\star\nu}(i,j) \equiv Z_\mu(i) \star Z_\nu(j) \equiv Z_\mu(i) +_{Z_\mu(i)\times Z_\nu(j)} Z_\nu(j)$.

▶ **Lemma 16.** *Let* $\mathfrak{m}$ *and* $\mathfrak{n}$ *be LAMs, and let* μ *and* ν *be null generators for* $\mathfrak{m}$ *and* $\mathfrak{n}$*, respectively. Then* $Z_{\mu\star\nu}(i,j)$ *is both* $\mathfrak{m}$*-connected and* $\mathfrak{n}$*-connected for all* $i : I_\mu$ *and* $j : I_\nu$*.*

Proof. Recall that a function is $\mathfrak{m}$-*connected* if its fibers are $\mathfrak{m}$-connected and that the class of $\mathfrak{m}$-connected functions is the left class of a (stable) orthogonal factorization system [22, Theorem 1.34]. Then the claim follows by the pushout stability and the right cancellability of connected functions. ◀

Lemma 16 shows $\mathfrak{m} \vee \mathfrak{n} \leq \mathfrak{Null}(\mu \star \nu)$ for arbitrary accessible modalities $\mathfrak{m}$ and $\mathfrak{n}$ and for arbitrary choices of μ and ν. We know neither if the other direction holds in general for some choices of μ and ν nor if $\mathfrak{Null}(\mu \star \nu)$ is independent of μ and ν. Note that Finster [7] observed that $\mathfrak{Null}(\mu \star \nu)$ is lex whenever $\mathfrak{Null}(\mu)$ and $\mathfrak{Null}(\nu)$ are lex. In the special case when $\mathfrak{m} \leq {}^{\perp}\mathfrak{n}$, the idea of the proof of $\mathfrak{m} \vee \mathfrak{n} = \mathfrak{Null}(\mu \star \nu)$ is to show that $\mathfrak{n}$ is an *open modality* within the subuniverse of $\mathfrak{Null}(\mu \star \nu)$-modal types.

▶ **Lemma 17.** *Let* $\mathfrak{m}$ *and* $\mathfrak{n}$ *be LAMs such that* $\mathfrak{m} \leq {}^{\perp}\mathfrak{n}$*. Then* $\mathfrak{n} \leq \mathfrak{Op}(\bigcirc_{\mathfrak{m}} \mathbf{0})$*.*

Proof. This is because $\bigcirc_{\mathfrak{m}} \mathbf{0}$ is $\mathfrak{n}$-connected by assumption. ◀

▶ **Lemma 18.** *Let* $\mathfrak{m}$ *and* $\mathfrak{n}$ *be LAMs such that* $\mathfrak{m} \leq {}^{\perp}\mathfrak{n}$*. Suppose that* μ *and* ν *are null generators for* $\mathfrak{m}$ *and* $\mathfrak{n}$*, respectively, and that* μ *admits a function* $f : \bigcirc_{\mathfrak{m}} \mathbf{0} \to I_\mu$ *such that* $\mathbf{0} \simeq Z_\mu(f(i))$ *for all* $i : \bigcirc_{\mathfrak{m}} \mathbf{0}$*. Then* $\bigcirc_{\mathfrak{Op}(\bigcirc_{\mathfrak{m}} \mathbf{0})} A$ *is* $\mathfrak{n}$*-modal for any* $\mathfrak{Null}(\mu \star \nu)$*-modal type* A*. Consequently, the canonical function* $\bigcirc_{\mathfrak{Op}(\bigcirc_{\mathfrak{m}} \mathbf{0})} A \to \bigcirc_{\mathfrak{n}} A$ *induced by Lemma 17 is an equivalence for any* $\mathfrak{Null}(\mu \star \nu)$*-modal type* A*.*

Proof. We show that $\bigcirc_{\mathfrak{Op}(\bigcirc_{\mathfrak{m}} \mathbf{0})} A \equiv (\bigcirc_{\mathfrak{m}} \mathbf{0} \to A)$ is $\mathfrak{n}$-modal. Since ν is a null generator for $\mathfrak{n}$, it suffices to show that $Z_\nu(j) \perp (\bigcirc_{\mathfrak{m}} \mathbf{0} \to A)$ for all $j : I_\nu$. This is equivalent to that $Z_\nu(j) \perp A$ under an assumption $i : \bigcirc_{\mathfrak{m}} \mathbf{0}$. This holds since $Z_\nu(j) \simeq \mathbf{0} \star Z_\nu(j) \simeq Z_\mu(f(i)) \star Z_\nu(j)$ and since A is $\mathfrak{Null}(\mu \star \nu)$-modal. ◀

▶ **Lemma 19.** *Let $\mathfrak{m}$ and $\mathfrak{n}$ be LAMs such that $\mathfrak{m} \leq {}^{\perp}\mathfrak{n}$. Suppose that μ and ν are null generators for $\mathfrak{m}$ and $\mathfrak{n}$, respectively, and that ν has an element $j : I_\nu$ such that $Z_\nu(j) \simeq \bigcirc_{\mathfrak{m}} \mathbf{0}$. Then, if a type A is $\mathfrak{Null}(\mu \star \nu)$-modal and $\mathfrak{Op}(\bigcirc_{\mathfrak{m}} \mathbf{0})$-connected, then it is $\mathfrak{m}$-modal.*

Proof. We show that $Z_\mu(i) \perp A$ for all $i : I_\mu$. By the definition of $\star$, we have the following pullback square.

$$\begin{tikzcd}
(Z_\mu(i) \star \bigcirc_{\mathfrak{m}} \mathbf{0} \to A) \arrow[r, "\simeq"] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & (Z_\mu(i) \to A) \arrow[d] \\
(\bigcirc_{\mathfrak{m}} \mathbf{0} \to A) \arrow[r, "\simeq"'] & (Z_\mu(i) \to \bigcirc_{\mathfrak{m}} \mathbf{0} \to A)
\end{tikzcd}$$

Since A is $\mathfrak{Op}(\bigcirc_{\mathfrak{m}} \mathbf{0})$-connected, the domain and codomain of the bottom function are contractible, and thus the bottom function is an equivalence. It then follows that the top function is also an equivalence. Since A is $\mathfrak{Null}(\mu \star \nu)$-modal and since $Z_\nu(j) \simeq \bigcirc_{\mathfrak{m}} \mathbf{0}$, we have $A \simeq (Z_\mu(i) \star \bigcirc_{\mathfrak{m}} \mathbf{0} \to A) \simeq (Z_\mu(i) \to A)$, and thus $Z_\mu(i) \perp A$. ◀

Proof of Proposition 14. Let μ and ν be null generators for $\mathfrak{m}$ and $\mathfrak{n}$, respectively. Note that adjoining a family of connected types to a null generator does not change the modality presented by the null generator. Under an assumption $i : \bigcirc_{\mathfrak{m}} \mathbf{0}$, the empty type $\mathbf{0}$ becomes $\mathfrak{m}$-connected, and thus we may assume that μ includes the type family $\lambda(\_ : \bigcirc_{\mathfrak{m}} \mathbf{0}).\mathbf{0}$. Since $\bigcirc_{\mathfrak{m}} \mathbf{0}$ is $\mathfrak{n}$-connected by assumption, we may assume that ν includes the type family $\lambda(\_ : \mathbf{1}).\bigcirc_{\mathfrak{m}} \mathbf{0}$.

We show that $\mathfrak{Null}(\mu \star \nu) = \mathfrak{m} \vee \mathfrak{n}$. By Lemma 16, $\mathfrak{m} \vee \mathfrak{n} \leq \mathfrak{Null}(\mu \star \nu)$. For the other direction, suppose that A is a $\mathfrak{Null}(\mu \star \nu)$-modal type. By [22, Theorem 3.50], it suffices to show that $\eta_{\mathfrak{n}}(A) : A \to \bigcirc_{\mathfrak{n}} A$ has $\mathfrak{m}$-modal fibers. By Lemma 18, $\bigcirc_{\mathfrak{n}} A \simeq \bigcirc_{\mathfrak{Op}(\bigcirc_{\mathfrak{m}} \mathbf{0})} A$. Then the fibers of $\eta_{\mathfrak{n}}(A)$ are $\mathfrak{Op}(\bigcirc_{\mathfrak{m}} \mathbf{0})$-connected. Since both A and $\bigcirc_{\mathfrak{n}} A$ are $\mathfrak{Null}(\mu \star \nu)$-modal, the fibers of $\eta_{\mathfrak{n}}(A)$ are also $\mathfrak{Null}(\mu \star \nu)$-modal. Thus, by Lemma 19, $\eta_{\mathfrak{n}}(A)$ has $\mathfrak{m}$-modal fibers. ◀

As a by-product, we have the following.

▶ **Corollary 20.** *Let $\mathfrak{m}$ and $\mathfrak{n}$ be LAMs such that $\mathfrak{m} \leq {}^{\perp}\mathfrak{n}$. If $\mathfrak{m} \vee \mathfrak{n} = \mathfrak{Top}$, then $\mathfrak{m}$ and $\mathfrak{n}$ are the closed and open, respectively, modalities associated to the proposition $\bigcirc_{\mathfrak{m}} \mathbf{0}$.* ◀

3 Mode sketches

We introduce *mode sketches* as shapes of diagrams of subuniverses definable internally to type theory. We work in homotopy type theory through the section.

3.1 Internal diagrams induced by modalities

We consider postulating some LAMs to encode some diagram of subuniverses. The fundamental observation is that a pair of LAMs induces a canonical functor between them.

▶ **Construction 21.** Let $\mathfrak{m}$ and $\mathfrak{n}$ be LAMs. We define a function $\bigcirc_{\mathfrak{m}}^{\mathfrak{n}} : \mathcal{U}_{\mathfrak{n}} \to \mathcal{U}_{\mathfrak{m}}$ to be the restriction of $\bigcirc_{\mathfrak{m}}$ to $\mathcal{U}_{\mathfrak{n}} \subset \mathcal{U}$.

▶ Remark 22. We can say that $\bigcirc^{\mathfrak{n}}_{\mathfrak{m}}$ is a functor *externally*: we can construct a function $\prod_{A,B:\mathcal{U}_{\mathfrak{n}}}(A \to B) \to (\bigcirc^{\mathfrak{n}}_{\mathfrak{m}} A \to \bigcirc^{\mathfrak{n}}_{\mathfrak{m}} B)$ and every instance of the coherence laws. However, it is not known how to state that $\bigcirc^{\mathfrak{n}}_{\mathfrak{m}}$ is a functor internally to type theory, because defining the type of $(\infty, 1)$-categories in plain homotopy type theory is still an open problem.

We have two functors $\bigcirc^{\mathfrak{m}}_{\mathfrak{n}} : \mathcal{U}_{\mathfrak{m}} \to \mathcal{U}_{\mathfrak{n}}$ and $\bigcirc^{\mathfrak{n}}_{\mathfrak{m}} : \mathcal{U}_{\mathfrak{n}} \to \mathcal{U}_{\mathfrak{m}}$ for every pair of LAMs $\mathfrak{m}$ and $\mathfrak{n}$, but we are often interested in only one direction. It is thus useful to cut off one direction by postulating that $\mathfrak{m} \leq {}^{\perp}\mathfrak{n}$: by the definition of connectedness, $\bigcirc^{\mathfrak{m}}_{\mathfrak{n}}$ becomes constant at the unit type. The other direction $\bigcirc^{\mathfrak{n}}_{\mathfrak{m}} : \mathcal{U}_{\mathfrak{n}} \to \mathcal{U}_{\mathfrak{m}}$ remains non-trivial. Therefore, a pair $(\mathfrak{m}, \mathfrak{n})$ of LAMs such that $\mathfrak{m} \leq {}^{\perp}\mathfrak{n}$ encodes a functor $\mathcal{U}_{\mathfrak{n}} \to \mathcal{U}_{\mathfrak{m}}$. When $\mathfrak{n} \leq {}^{\perp}\mathfrak{m}$ is also assumed, $\mathcal{U}_{\mathfrak{m}}$ and $\mathcal{U}_{\mathfrak{n}}$ are considered unrelated.

Given more than two LAMs, we have canonical natural transformations between the canonical functors.

▶ **Construction 23.** Let $\mathfrak{m}_0, \mathfrak{m}_1, \mathfrak{m}_2$ be LAMs. We define

$$\eta^{\mathfrak{m}_0;\mathfrak{m}_2}_{\mathfrak{m}_1} : \prod_{A:\mathcal{U}_{\mathfrak{m}_2}} \bigcirc^{\mathfrak{m}_2}_{\mathfrak{m}_0} A \to \bigcirc^{\mathfrak{m}_1}_{\mathfrak{m}_0} \bigcirc^{\mathfrak{m}_2}_{\mathfrak{m}_1} A$$

by $\eta^{\mathfrak{m}_0;\mathfrak{m}_2}_{\mathfrak{m}_1}(A) \equiv \bigcirc_{\mathfrak{m}_0} \eta_{\mathfrak{m}_1}(A)$. This family of functions is natural in the sense that for any $A, B : \mathcal{U}_{\mathfrak{m}_2}$ and $f : A \to B$, we have a homotopy filling the following square.

$$\begin{tikzcd}
\bigcirc^{\mathfrak{m}_2}_{\mathfrak{m}_0} A \arrow[r, "\eta^{\mathfrak{m}_0;\mathfrak{m}_2}_{\mathfrak{m}_1}(A)"] \arrow[d, "\bigcirc^{\mathfrak{m}_2}_{\mathfrak{m}_0} f"'] & \bigcirc^{\mathfrak{m}_1}_{\mathfrak{m}_0} \bigcirc^{\mathfrak{m}_2}_{\mathfrak{m}_1} A \arrow[d, "\bigcirc^{\mathfrak{m}_1}_{\mathfrak{m}_0} \bigcirc^{\mathfrak{m}_2}_{\mathfrak{m}_1} f"] \\
\bigcirc^{\mathfrak{m}_2}_{\mathfrak{m}_0} B \arrow[r, "\eta^{\mathfrak{m}_0;\mathfrak{m}_2}_{\mathfrak{m}_1}(B)"'] & \bigcirc^{\mathfrak{m}_1}_{\mathfrak{m}_0} \bigcirc^{\mathfrak{m}_2}_{\mathfrak{m}_1} B
\end{tikzcd}$$

Let $\mathfrak{m}_0, \mathfrak{m}_1, \mathfrak{m}_2, \mathfrak{m}_3$ be LAMs. By naturality, the following diagram commutes.

$$\begin{tikzcd}
\bigcirc^{\mathfrak{m}_3}_{\mathfrak{m}_0} \arrow[r, "\eta^{\mathfrak{m}_0;\mathfrak{m}_3}_{\mathfrak{m}_1}"] \arrow[d, "\eta^{\mathfrak{m}_0;\mathfrak{m}_3}_{\mathfrak{m}_2}"'] & \bigcirc^{\mathfrak{m}_1}_{\mathfrak{m}_0} \bigcirc^{\mathfrak{m}_3}_{\mathfrak{m}_1} \arrow[d, "\bigcirc^{\mathfrak{m}_1}_{\mathfrak{m}_0} \eta^{\mathfrak{m}_1;\mathfrak{m}_3}_{\mathfrak{m}_2}"] \\
\bigcirc^{\mathfrak{m}_2}_{\mathfrak{m}_0} \bigcirc^{\mathfrak{m}_3}_{\mathfrak{m}_2} \arrow[r, "\eta^{\mathfrak{m}_0;\mathfrak{m}_2}_{\mathfrak{m}_1} \bigcirc^{\mathfrak{m}_3}_{\mathfrak{m}_2}"'] & \bigcirc^{\mathfrak{m}_1}_{\mathfrak{m}_0} \bigcirc^{\mathfrak{m}_2}_{\mathfrak{m}_1} \bigcirc^{\mathfrak{m}_3}_{\mathfrak{m}_2}
\end{tikzcd}$$

For more than four LAMs, higher coherence laws are also satisfied. Hence, a tuple $(\mathfrak{m}_0, \dots, \mathfrak{m}_n)$ of LAMs such that $\mathfrak{m}_i \leq {}^{\perp}\mathfrak{m}_j$ for all $i < j$ encodes an n-simplex with vertices $\mathcal{U}_{\mathfrak{m}_i}$, edges $\bigcirc^{\mathfrak{m}_j}_{\mathfrak{m}_i} : \mathcal{U}_{\mathfrak{m}_j} \to \mathcal{U}_{\mathfrak{m}_i}$ for $i < j$, triangles

$$\begin{tikzcd}
\mathcal{U}_{\mathfrak{m}_i} & & \mathcal{U}_{\mathfrak{m}_k} \arrow[ll, "\bigcirc^{\mathfrak{m}_k}_{\mathfrak{m}_i}"'] \arrow[dl, "\bigcirc^{\mathfrak{m}_k}_{\mathfrak{m}_j}"] \\
& \mathcal{U}_{\mathfrak{m}_j} \arrow[ul, "\bigcirc^{\mathfrak{m}_j}_{\mathfrak{m}_i}"] \arrow[u, phantom, "\Downarrow \eta^{\mathfrak{m}_i;\mathfrak{m}_k}_{\mathfrak{m}_j}"] &
\end{tikzcd}$$

for $i < j < k$, and higher homotopies.

Shapes other than simplices are expressed by postulating invertibility of some of $\eta^{\mathfrak{m}_i;\mathfrak{m}_k}_{\mathfrak{m}_j}$'s. For example, let $\mathfrak{m}_0, \mathfrak{m}_1, \mathfrak{m}_2, \mathfrak{m}_3$ be LAMs and suppose that $\mathfrak{m}_i \leq {}^{\perp}\mathfrak{m}_j$ for all $i < j$, that $\mathfrak{m}_2 \leq {}^{\perp}\mathfrak{m}_1$, and that $\eta^{\mathfrak{m}_0;\mathfrak{m}_3}_{\mathfrak{m}_1}$ is invertible. We have a diagram

$$\begin{tikzcd}
 & \mathcal{U}_{\mathfrak{m}_1} \arrow[ddl, "\bigcirc^{\mathfrak{m}_1}_{\mathfrak{m}_0}"'] & \\
 & {} \arrow[u, Rightarrow, "\simeq", "\eta^{\mathfrak{m}_0;\mathfrak{m}_3}_{\mathfrak{m}_1}"'] & \\
\mathcal{U}_{\mathfrak{m}_0} & & \mathcal{U}_{\mathfrak{m}_3} \arrow[ll, "\bigcirc^{\mathfrak{m}_3}_{\mathfrak{m}_0}" description] \arrow[uul, "\bigcirc^{\mathfrak{m}_3}_{\mathfrak{m}_1}"'] \arrow[ddl, "\bigcirc^{\mathfrak{m}_3}_{\mathfrak{m}_2}"] \\
 & {} \arrow[d, Rightarrow, "\eta^{\mathfrak{m}_0;\mathfrak{m}_3}_{\mathfrak{m}_2}"'] & \\
 & \mathcal{U}_{\mathfrak{m}_2} \arrow[uul, "\bigcirc^{\mathfrak{m}_2}_{\mathfrak{m}_0}"] &
\end{tikzcd}$$

which is equivalent to a diagram of the form

$$\begin{tikzcd}
 & \mathcal{U}_{\mathfrak{m}_1} \arrow[dl, "\bigcirc^{\mathfrak{m}_1}_{\mathfrak{m}_0}"'] \arrow[dd, Rightarrow] & \\
\mathcal{U}_{\mathfrak{m}_0} & & \mathcal{U}_{\mathfrak{m}_3}. \arrow[ul, "\bigcirc^{\mathfrak{m}_3}_{\mathfrak{m}_1}"'] \arrow[dl, "\bigcirc^{\mathfrak{m}_3}_{\mathfrak{m}_2}"] \\
 & \mathcal{U}_{\mathfrak{m}_2} \arrow[ul, "\bigcirc^{\mathfrak{m}_2}_{\mathfrak{m}_0}"] &
\end{tikzcd}$$

We cannot, however, naively postulate some properties of the functors $\bigcirc^{\mathfrak{n}}_{\mathfrak{m}}$'s such as conservativity, fullness, faithfulness, adjointness, and invertibility. This is because the internal statements of these conditions are too strong due to stability under substitution, and indeed some "no-go" theorems on internalizing properties of functors are known [14, Theorem 5.1][25, Theorem 4.1].

▶ Remark 24. It is *possible* to postulate arbitrary properties of $\bigcirc^{\mathfrak{m}_j}_{\mathfrak{m}_i}$'s in the following way. We first postulate a "base" LAM $\mathfrak{Base}$ and assume $\mathfrak{Base} \leq {}^{\perp}\mathfrak{m}_i$ for all i. The universe $\mathcal{U}_{\mathfrak{Base}}$ is intended to be interpreted as the $(\infty, 1)$-category of spaces, so statements in $\mathcal{U}_{\mathfrak{Base}}$ will correspond to external statements. Since $\bigcirc_{\mathfrak{Base}} : \mathcal{U} \to \mathcal{U}_{\mathfrak{Base}}$ preserves finite limits, it takes $(\infty, 1)$-categories to $(\infty, 1)$-categories and functors to functors. We can then postulate any property on the induced functor $\bigcirc_{\mathfrak{Base}}\mathcal{U}_{\mathfrak{m}_j} \to \bigcirc_{\mathfrak{Base}}\mathcal{U}_{\mathfrak{m}_i}$. In fact, cohesive homotopy type theory [23] was first formulated in a similar fashion where the $\sharp$ modality plays the role of $\mathfrak{Base}$. However, since we only know that $\bigcirc_{\mathfrak{Base}}\mathcal{U}_{\mathfrak{m}_i}$ is an $(\infty, 1)$-category *externally*, this approach is not so convenient to work with especially for formalization in proof assistants. For this and some other reasons, the newer version of cohesive homotopy type theory [25] is a proper extension of homotopy type theory. Nevertheless, this adding-base approach is attractive since it keeps type theory simple and works for any kind of diagram.

3.2 Mode sketches

We introduce *mode sketches* as shapes of diagrams definable by the methodology explained in Section 3.1.

▶ **Definition 25.** *A* mode sketch $\mathfrak{M}$ *consists of the following data:*

- *a decidable finite poset* $I_{\mathfrak{M}}$*;*
- *a subset* $T_{\mathfrak{M}}$ *of triangles in* $I_{\mathfrak{M}}$*.*

Here, by a decidable *poset we mean a poset whose ordering relation $\leq$ is decidable. A type is* finite *if it is merely equivalent to the coproduct of n copies of* $\mathbf{1}$ *for some $n : \mathbb{N}$ [21, Definition 16.3.1]. The identity type on a finite type is decidable [21, Remark 16.3.2]. The strict ordering relation $i < j$ defined as $(i \leq j) \wedge (i \neq j)$ is also decidable. By a* triangle *in $I_{\mathfrak{M}}$ we mean an ordered triple $(i_0 < i_1 < i_2)$ of elements of $I_{\mathfrak{M}}$. A triangle in $T_{\mathfrak{M}}$ is called* thin.

▶ Remark 26. The definition of mode sketches also makes sense in the metatheory. Every mode sketch $\mathfrak{M}$ in the metatheory can be encoded in type theory since it is finite.

Let $\mathfrak{M}$ be a mode sketch and $\mathfrak{m} : \mathfrak{M} \to \mathsf{LAM}$ a function. We consider the following axioms.

▶ **Axiom A.** $\mathfrak{m}(i) \leq {}^{\perp}\mathfrak{m}(j)$ for any $j \not\leq i$ in $\mathfrak{M}$.

▶ **Axiom B.** For any triangle $(i_0 < i_1 < i_2) : T_{\mathfrak{M}}$, the natural transformation $\eta^{\mathfrak{m}(i_0);\mathfrak{m}(i_2)}_{\mathfrak{m}(i_1)} : \bigcirc^{\mathfrak{m}(i_2)}_{\mathfrak{m}(i_0)} \Rightarrow \bigcirc^{\mathfrak{m}(i_1)}_{\mathfrak{m}(i_0)} \bigcirc^{\mathfrak{m}(i_2)}_{\mathfrak{m}(i_1)}$ is invertible.

▶ **Axiom C.** The top modality is the canonical join $\bigvee_{\mathfrak{M}} \mathfrak{m}$.

▶ Remark 27. Assuming Axiom A, if $i < j$, then $\mathfrak{m}(i) \leq {}^{\perp}\mathfrak{m}(j)$.

Axioms A and B are motivated by the observation made in Section 3.1. That is, when $j \not\leq i$, the functor in the direction $\mathcal{U}_{\mathfrak{m}(i)} \to \mathcal{U}_{\mathfrak{m}(j)}$ is cut off. Our intended models constructed in Section 5 additionally satisfy Axiom C. This axiom is not so important in practical use, since our primary aim is to draw a diagram of ∞-logoses inside homotopy type theory, but Axiom C does nothing for this purpose. It is even better to work without Axiom C, because Axioms A and B are stable under restriction along a full inclusion $\mathfrak{M}' \subset \mathfrak{M}$ while Axiom C is not. Axiom C is meant to exclude models other than intended models.

▶ Remark 28. A mode sketch $\mathfrak{M}$ is regarded as a presentation of an $(\infty, 2)$-category. The strict ordering relation generates 1-cells $(i < j) : i \to j$, and the triangles $(i < j < k)$ generate 2-cells in the direction

$$\begin{tikzcd} & j \arrow[dr] \arrow[d, phantom, "\Downarrow"] & \\ i \arrow[ur] \arrow[rr] & & k. \end{tikzcd}$$

When the triangle is thin, the corresponding 2-cell is made invertible. Longer chains $(i_0 < i_1 < \cdots < i_n)$ present coherence. Formally, we regard $\mathfrak{M}$ as a *scaled simplicial set* [17], one of models for $(\infty, 2)$-categories, by taking the nerve and marking thin triangles as thin 2-simplices, and then reverse 2-cells. A function $\mathfrak{m} : \mathfrak{M} \to \mathsf{LAM}$ satisfying Axioms A–C is then considered as a diagram of subuniverses indexed over $\mathfrak{M}^{\mathrm{op}(1,2)}$, the $(\infty, 2)$-category obtained from $\mathfrak{M}$ by reversing the directions of 1-cells and 2-cells.

▶ **Example 29.** Every decidable finite poset is a mode sketch where no triangle is thin. The $(\infty, 2)$-category presented by it is obtained from the left adjoint of the Duskin nerve [6] by reversing 2-cells.

▶ **Example 30.** The *mode sketch for functors* is drawn as

$$\begin{tikzcd} 0 \arrow[r] & 1. \end{tikzcd}$$

Axiom A asserts $\mathfrak{m}(0) \leq {}^{\perp}\mathfrak{m}(1)$. Axiom B is empty since there is no triangle. Thus, we get the following diagram.

$$\begin{tikzcd}
\mathcal{U}_{\mathfrak{m}(0)} & \mathcal{U}_{\mathfrak{m}(1)} \arrow[l, "\bigcirc^{\mathfrak{m}(1)}_{\mathfrak{m}(0)}"']
\end{tikzcd}$$

Axiom C asserts $\mathfrak{m}(0) \vee \mathfrak{m}(1) = \mathfrak{Top}$.

▶ **Example 31.** The *mode sketch for triangles* is drawn as

$$\begin{tikzcd}
0 \arrow[rr] \arrow[dr] & \simeq & 2 \\
& 1 \arrow[ur] &
\end{tikzcd}$$

where "$\simeq$" indicates that the triangle is thin. Axiom A asserts $\mathfrak{m}(0) \leq {}^{\perp}\mathfrak{m}(1)$, $\mathfrak{m}(0) \leq {}^{\perp}\mathfrak{m}(2)$, and $\mathfrak{m}(1) \leq {}^{\perp}\mathfrak{m}(2)$. Axiom B asserts that $\eta^{\mathfrak{m}(0);\mathfrak{m}(2)}_{\mathfrak{m}(1)}$ is invertible. Thus, we have the following commutative triangle.

$$\begin{tikzcd}
\mathcal{U}_{\mathfrak{m}(0)} & & \mathcal{U}_{\mathfrak{m}(2)} \arrow[ll, "\bigcirc^{\mathfrak{m}(2)}_{\mathfrak{m}(0)}"'] \arrow[dl, "\bigcirc^{\mathfrak{m}(2)}_{\mathfrak{m}(1)}"] \\
& \mathcal{U}_{\mathfrak{m}(1)} \arrow[ul, "\bigcirc^{\mathfrak{m}(1)}_{\mathfrak{m}(0)}"] &
\end{tikzcd}$$

Axiom C asserts $\mathfrak{m}(0) \vee \mathfrak{m}(1) \vee \mathfrak{m}(2) = \mathfrak{Top}$. Notice that theorems for the mode sketch for functors proved without Axiom C also apply to the three edges in the above diagram. To keep this reusability, we should not assume Axiom C in practical use.

3.3 Intended models, internally

Let $\mathfrak{M}$ be a mode sketch. We can internally see what kind of an ∞-logos is a model of $\mathfrak{M}$. Here, by a model of $\mathfrak{M}$ we mean an ∞-logos that admits an interpretation of a postulated function $\mathfrak{m} : \mathfrak{M} \to \mathsf{LAM}$ satisfying Axioms A–C.

▶ **Example 32.** Consider the case when $\mathfrak{M}$ is the mode sketch for functors (Example 30). Proposition 13 implies that $\mathcal{U} \simeq \mathcal{U}_{\mathfrak{m}(0)\vee\mathfrak{m}(1)}$ is the *Artin gluing* for the functor $\bigcirc^{\mathfrak{m}(1)}_{\mathfrak{m}(0)} : \mathcal{U}_{\mathfrak{m}(1)} \to \mathcal{U}_{\mathfrak{m}(0)}$. Therefore, our intended models of $\mathfrak{M}$ are ∞-logoses obtained by the Artin gluing.

A generalization of the Artin gluing is *oplax limits*. In the setting of Example 32, $\mathcal{U}$ fits into the following *universal oplax cone* over the diagram $\mathcal{U}_{\mathfrak{m}(0)} \xleftarrow{\bigcirc^{\mathfrak{m}(1)}_{\mathfrak{m}(0)}} \mathcal{U}_{\mathfrak{m}(1)}$.

$$\begin{tikzcd}
& \mathcal{U} \arrow[dl] \arrow[dr] \arrow[d, phantom, "\Longrightarrow"] & \\
\mathcal{U}_{\mathfrak{m}(0)} & {} & \mathcal{U}_{\mathfrak{m}(1)} \arrow[ll, "\bigcirc^{\mathfrak{m}(1)}_{\mathfrak{m}(0)}"]
\end{tikzcd} \tag{1}$$

An oplax cone over a diagram is a kind of cone but every triangle formed by two projections and a functor in the diagram is only filled by a not necessarily invertible natural transformation in the direction of Diagram (1). The universal oplax cone or oplax limit is the terminal object in the $(\infty, 1)$-category of oplax cones.

▶ **Example 33.** Consider the case when $\mathfrak{M}$ is the mode sketch $\{0 \to 1 \to 2\}$ with no thin triangle. Iterating Proposition 13, we see that every type $A : \mathcal{U}$ is fractured into $A_0 : \mathcal{U}_{\mathfrak{m}(0)}$, $A_1 : \mathcal{U}_{\mathfrak{m}(1)}$, $A_2 : \mathcal{U}_{\mathfrak{m}(2)}$, $f_{01} : A_0 \to \bigcirc^{\mathfrak{m}(1)}_{\mathfrak{m}(0)} A_1$, $f_{02} : A_0 \to \bigcirc^{\mathfrak{m}(2)}_{\mathfrak{m}(0)} A_2$, $f_{12} : A_1 \to \bigcirc^{\mathfrak{m}(2)}_{\mathfrak{m}(1)} A_2$, and $p_{012} : \bigcirc^{\mathfrak{m}(1)}_{\mathfrak{m}(0)} f_{12} \circ f_{01} = \eta^{\mathfrak{m}(0);\mathfrak{m}(2)}_{\mathfrak{m}(1)}(A_2) \circ f_{02}$. Indeed, we have

$$\begin{aligned}
& \mathcal{U}_{\mathfrak{m}(0) \vee \mathfrak{m}(1) \vee \mathfrak{m}(2)} \\
\simeq \quad & \{\text{Proposition 13 for } \mathfrak{m}(0) \text{ and } \mathfrak{m}(1) \vee \mathfrak{m}(2)\} \\
& \textstyle\sum_{A_0 : \mathcal{U}_{\mathfrak{m}(0)}} \sum_{A_{12} : \mathcal{U}_{\mathfrak{m}(1) \vee \mathfrak{m}(2)}} A_0 \to \bigcirc_{\mathfrak{m}_0} A_{12} \\
\simeq \quad & \{\text{Proposition 13 for } \mathfrak{m}(1) \text{ and } \mathfrak{m}(2)\} \\
& \textstyle\sum_{A_0 : \mathcal{U}_{\mathfrak{m}(0)}} \sum_{A_1 : \mathcal{U}_{\mathfrak{m}(1)}} \sum_{A_2 : \mathcal{U}_{\mathfrak{m}(2)}} \sum_{f_{12} : A_1 \to \bigcirc_{\mathfrak{m}(1)} A_2} A_0 \to \bigcirc_{\mathfrak{m}(0)}(A_1 \times_{\bigcirc_{\mathfrak{m}(1)} A_2} A_2)
\end{aligned}$$

where the pullback is taken for $f_{12} : A_1 \to \bigcirc_{\mathfrak{m}(1)} A_2$ and $\eta_{\mathfrak{m}(1)}(A_2) : A_2 \to \bigcirc_{\mathfrak{m}(1)} A_2$. Since $\bigcirc_{\mathfrak{m}(0)}$ preserves pullbacks, the component $A_0 \to \bigcirc_{\mathfrak{m}(0)}(A_1 \times_{\bigcirc_{\mathfrak{m}(1)} A_2} A_2)$ corresponds to the components f_{01}, f_{02}, and p_{012}. Then $\mathcal{U}$ is the oplax limit of the diagram

$$\begin{tikzcd}
\mathcal{U}_{\mathfrak{m}(0)} & {} & \mathcal{U}_{\mathfrak{m}(2)} . \arrow[ll, "\bigcirc^{\mathfrak{m}(2)}_{\mathfrak{m}(0)}"'] \arrow[dl, "\bigcirc^{\mathfrak{m}(2)}_{\mathfrak{m}(1)}"] \\
& \mathcal{U}_{\mathfrak{m}(1)} \arrow[ul, "\bigcirc^{\mathfrak{m}(1)}_{\mathfrak{m}(0)}"] \arrow[from=1-2, to=2-2, Rightarrow, shorten=2mm, "\eta^{\mathfrak{m}(0);\mathfrak{m}(2)}_{\mathfrak{m}(1)}"]
\end{tikzcd} \tag{2}$$

This means that we have projections $A_i : \mathcal{U} \to \mathcal{U}_{\mathfrak{m}(i)}$ for all i, natural transformations

$$\begin{tikzcd}
& \mathcal{U} \arrow[dl, "A_i"'] \arrow[dr, "A_j"] & \\
\mathcal{U}_{\mathfrak{m}(i)} \arrow[rr, phantom, shift left=3, "\overset{\Longrightarrow}{f_{ij}}"] & & \mathcal{U}_{\mathfrak{m}(j)} \arrow[ll, "\bigcirc^{\mathfrak{m}(j)}_{\mathfrak{m}(i)}"]
\end{tikzcd}$$

for all $i < j$, and a homotopy

$$\begin{tikzcd}
& \mathcal{U} \arrow[dl, "A_0"'] \arrow[dr, "A_2"] \arrow[dd, "A_1" description] & \\
\mathcal{U}_{\mathfrak{m}(0)} \arrow[r, phantom, "\overset{\Rightarrow}{f_{01}}"] & {} \arrow[r, phantom, "\overset{\Rightarrow}{f_{12}}"] & \mathcal{U}_{\mathfrak{m}(2)} \arrow[dl, "\bigcirc^{\mathfrak{m}(2)}_{\mathfrak{m}(1)}"] \\
& \mathcal{U}_{\mathfrak{m}(1)} \arrow[ul, "\bigcirc^{\mathfrak{m}(1)}_{\mathfrak{m}(0)}"] &
\end{tikzcd}
\overset{p_{012}}{=}
\begin{tikzcd}
& \mathcal{U} \arrow[dl, "A_0"'] \arrow[dr, "A_2"] & \\
\mathcal{U}_{\mathfrak{m}(0)} & {} & \mathcal{U}_{\mathfrak{m}(2)} , \arrow[ll, "\bigcirc^{\mathfrak{m}(2)}_{\mathfrak{m}(0)}" description] \arrow[dl, "\bigcirc^{\mathfrak{m}(2)}_{\mathfrak{m}(1)}"] \\
& \mathcal{U}_{\mathfrak{m}(1)} \arrow[ul, "\bigcirc^{\mathfrak{m}(1)}_{\mathfrak{m}(0)}"] \arrow[from=1-2, to=2-2, phantom, "\overset{f_{02}}{\Longrightarrow}"] \arrow[from=2-2, to=3-2, Rightarrow, shorten=2mm, "\eta^{\mathfrak{m}(0);\mathfrak{m}(2)}_{\mathfrak{m}(1)}"] &
\end{tikzcd}$$

and these data form a universal oplax cone over Diagram (2).

Let us make the triangle $(0 < 1 < 2)$ thin so that the natural transformation $\eta^{\mathfrak{m}(0);\mathfrak{m}(2)}_{\mathfrak{m}(1)}$ becomes invertible. In this setting, $\mathcal{U}$ is still the oplax limit of Diagram (2), but the presentation can be simplified since the type of data (f_{02}, p_{012}) is contractible.

For a general mode sketch $\mathfrak{M}$, we apply Proposition 13 for a minimal element $\mathfrak{m}(i_0)$ and the rest $\bigvee_{i : \mathfrak{M} \setminus i_0} \mathfrak{m}(i)$ and repeat this for $\mathfrak{M} \setminus i_0$ to fracture types into modal types. Examples 32 and 33 suggest that $\mathcal{U}$ is the oplax limit of the diagram formed by $\mathcal{U}_{\mathfrak{m}(i)}$'s explained in Remark 28. Thus, our intended models of $\mathfrak{M}$ are oplax limits of ∞-logoses indexed over the $(\infty, 2)$-category presented by $\mathfrak{M}$. The formal account of this is sketched in Section 5 and fully described in the extended version [33].

4 Mode sketches and synthetic Tait computability

We give an alternative set of axioms for mode sketches and exhibit a connection between mode sketches and *synthetic Tait computability* of Sterling [27]. The core axiom of synthetic Tait computability is to postulate a proposition. The proposition induces the open and closed modalities, and then every type is fractured into an open type equipped with a closed type family and behaves like a *logical relation*. In this story, the open and closed modalities seem more essential than the postulated proposition, so we aim to formulate synthetic Tait computability purely in terms of modalities. We work in homotopy type theory.

4.1 Alternative mode sketch axioms

The ∞-logoses obtained by the Artin gluing can be characterized as ∞-logoses equipped with a subterminal object; see [11, A4.5.6] for the 1-categorical case. We generalize this from the Artin gluing to oplax limits indexed by mode sketches, internally to type theory: the type of functions $\mathfrak{M} \to \mathsf{LAM}$ satisfying Axioms A and C is equivalent to the type of morphisms from the lattice of cosieves on $\mathfrak{M}$ to the lattice Prop (Theorem 37).

▶ **Definition 34.** *A* cosieve *on a decidable poset I is an upward-closed decidable subset of it. Let $\mathsf{coSieve}(I)$ denote the poset of cosieves on I ordered by inclusion. Note that cosieves are closed under finite meets and joins, so $\mathsf{coSieve}(I)$ is a lattice.*

▶ **Notation 35.** For $i : \mathfrak{M}$, let $(i \downarrow \mathfrak{M})$ denote the cosieve $\{j : \mathfrak{M} \mid i \leq j\}$ and $\partial(i \downarrow \mathfrak{M})$ the cosieve $(i \downarrow \mathfrak{M}) \setminus \{i\}$.

▶ **Construction 36.** Let $P : \mathsf{coSieve}(\mathfrak{M}) \to \mathsf{Prop}$ be a function. We define a function $\mathfrak{a}_P : \mathfrak{M} \to \mathsf{LAM}$ by $\mathfrak{a}_P(i) \equiv \mathfrak{Op}(P(i \downarrow \mathfrak{M})) \wedge \mathfrak{Cl}(P(\partial(i \downarrow \mathfrak{M})))$.

▶ **Theorem 37.** *Construction 36 is restricted to an equivalence between the following types:*
1. *the type of lattice morphisms $P : \mathsf{coSieve}(\mathfrak{M}) \to \mathsf{Prop}$;*
2. *the type of functions $\mathfrak{m} : \mathfrak{M} \to \mathsf{LAM}$ satisfying Axioms A and C.*

Before giving a proof of Theorem 37, let us relate Theorem 37 to *synthetic Tait computability* [29, 27, 30]. The core axiom of synthetic Tait computability is to postulate some propositions. One can work with those propositions directly but also with the induced open and closed modalities. Theorem 37 says that synthetic Tait computability can, in fact, be formulated completely in terms of modalities. The simplest version of synthetic Tait computability postulates a single proposition. The corresponding mode sketch is $\{0 \to 1\}$ as follows.

▶ **Example 38.** Let $\mathfrak{M}$ be the mode sketch for functors (Example 30). Then $\mathsf{coSieve}(\mathfrak{M}) = \{\{\}, \{1\}, \{0, 1\}\}$ is the free lattice generated by the single element $\{1\}$. We thus have $\{\text{lattice morphisms } \mathsf{coSieve}(\mathfrak{M}) \to \mathsf{Prop}\} \simeq \mathsf{Prop}$.

The rest of this subsection is devoted to the proof of Theorem 37. Because of space constraints, technical details are omitted and found in the extended version [33]. Here we focus on how to give an inverse construction to Construction 36. The key observation is that canonical joins of $\mathfrak{m}(i)$'s exist and are well-behaved under Axiom A.

▶ **Proposition 39.** *If a function $\mathfrak{m} : \mathfrak{M} \to \mathsf{LAM}$ satisfies Axiom A, then the canonical join $\bigvee_S \mathfrak{m}$ exists for any decidable subset $S \subset \mathfrak{M}$.*

Proof. By induction on the size of S. If S is empty, then $\bigvee_{\emptyset} \mathfrak{m}$ is the bottom modality. Suppose that S is non-empty. Since $\mathfrak{M}$ is finite, there is an element i_0 minimal in S. Then $S \setminus \{i_0\}$ admits a canonical join by the induction hypothesis. Since i_0 is minimal, $\mathfrak{m}(i_0) \leq {}^{\perp}\mathfrak{m}(i)$ for any $i : S \setminus \{i_0\}$ by Axiom A, and thus $\mathfrak{m}(i_0) \leq {}^{\perp}(\bigvee_{S \setminus \{i_0\}} \mathfrak{m})$. Then we have the canonical join $\bigvee_S \mathfrak{m} \equiv \mathfrak{m}(i_0) \vee (\bigvee_{(S \setminus \{i_0\})} \mathfrak{m})$ by Proposition 13. ◀

▶ **Lemma 40.** *Let $\mathfrak{m}_0$, $\mathfrak{m}_1$, and $\mathfrak{m}_2$ be LAMs such that $\mathfrak{m}_i \leq {}^{\perp}\mathfrak{m}_j$ for any $i < j$. Then $\mathfrak{m}_0 \vee \mathfrak{m}_1 \leq {}^{\perp}\mathfrak{m}_2$.*

Proof. Let A be a $(\mathfrak{m}_0 \vee \mathfrak{m}_1)$-modal type. By Proposition 13, $\eta_{\mathfrak{m}_1}(A) : A \to \bigcirc_{\mathfrak{m}_1} A$ has $\mathfrak{m}_0$-modal fibers. Then, by assumption, $\bigcirc_{\mathfrak{m}_1} A$ and the fibers of $\eta_{\mathfrak{m}_1}(A)$ are made contractible by $\bigcirc_{\mathfrak{m}_2}$. Thus, $\bigcirc_{\mathfrak{m}_2} A$ is contractible. ◀

▶ **Proposition 41.** *If a function $\mathfrak{m} : \mathfrak{M} \to \mathsf{LAM}$ satisfies Axiom A, then $\bigvee_{\mathfrak{M} \setminus S} \mathfrak{m} \leq {}^{\perp}(\bigvee_S \mathfrak{m})$ for any cosieve $S \subset \mathfrak{M}$.*

Proof. Since S is upward-closed, $j \not\leq i$ for any $i : \mathfrak{M} \setminus S$ and $j : S$. Thus, by Axiom A, $\mathfrak{m}(i) \leq {}^{\perp}\mathfrak{m}(j)$ for any $i : \mathfrak{M} \setminus S$ and $j : S$. The claim follows from Lemma 40 and the construction of the canonical join in Proposition 39. ◀

Sketch of proof of Theorem 37. Let $\mathfrak{m} : \mathfrak{M} \to \mathsf{LAM}$ be a function satisfying Axioms A and C. We define a function $\varphi_{\mathfrak{m}} : \mathsf{coSieve}(\mathfrak{M}) \to \mathsf{Prop}$ by $\varphi_{\mathfrak{m}}(S) \equiv \bigcirc_{\bigvee_{\mathfrak{M} \setminus S} \mathfrak{m}} \mathbf{0}$ which exists by Proposition 39. By Corollary 20 and by Proposition 41, $\varphi_{\mathfrak{m}}(S)$ is the unique proposition such that $\mathfrak{Op}(\varphi_{\mathfrak{m}}(S)) = \bigvee_S \mathfrak{m}$. On the other hand, we have $\bigvee_S \mathfrak{a}_P = \mathfrak{Op}(P(S))$ by construction, from which one can derive that the constructions $P \mapsto \mathfrak{a}_P$ and $\mathfrak{m} \mapsto \varphi_{\mathfrak{m}}$ are mutually inverses. We again note that technical details are omitted and found in the extended version [33]. Certain amount of calculation is needed to prove that $\mathfrak{a}_P : \mathfrak{M} \to \mathsf{LAM}$ satisfies Axioms A and C and that $\varphi_{\mathfrak{m}}$ is a lattice morphism. ◀

4.2 Logical relations as types

We have seen in Section 4.1 that synthetic Tait computability is reformulated in terms of LAMs. The slogan of synthetic Tait computability is "logical relations as types" [29]. This is also formulated purely in terms of LAMs.

▶ **Fact 42** ([22, Theorem 3.11])**.** *For any LAM $\mathfrak{m}$, the universe of $\mathfrak{m}$-modal types $\mathcal{U}_{\mathfrak{m}} \equiv \{A : \mathcal{U} \mid \mathsf{In}_{\mathfrak{m}}(A)\}$ is $\mathfrak{m}$-modal.*

▶ **Proposition 43** (Fracture and gluing)**.** *Let $\mathfrak{m}$ and $\mathfrak{n}$ be LAMs such that $\mathfrak{m} \leq {}^{\perp}\mathfrak{n}$. Then we have an equivalence*

$$\mathcal{U}_{\mathfrak{m} \vee \mathfrak{n}} \simeq \textstyle\sum_{B : \mathcal{U}_{\mathfrak{n}}} B \to \mathcal{U}_{\mathfrak{m}}$$

whose right-to-left function sends a (B, A) to $\sum_{\mathtt{x} : B} A(\mathtt{x})$.

Proof. For any $B : \mathcal{U}_{\mathfrak{n}}$, we have

$$\begin{aligned} & \textstyle\sum_{A : \mathcal{U}_{\mathfrak{m}}} A \to \bigcirc_{\mathfrak{m}} B \\ \simeq \quad & \{\text{equivalence between fibrations and type families}\} \\ & \bigcirc_{\mathfrak{m}} B \to \mathcal{U}_{\mathfrak{m}} \\ \simeq \quad & \{\text{Fact 42}\} \\ & B \to \mathcal{U}_{\mathfrak{m}}. \end{aligned}$$

Then apply Proposition 13. ◀

Proposition 43 asserts that a type in $\mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}}$ is a $\mathfrak{n}$-modal type equipped with a $\mathfrak{m}$-modal unary (proof-relevant) relation on it, so *types (in $\mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}}$) are relations*. More generally, for a mode sketch $\mathfrak{M}$ and a function $\mathfrak{m} : \mathfrak{M} \to \mathsf{LAM}$ satisfying Axiom A, types in $\mathcal{U}_{\bigvee_{\mathfrak{M}} \mathfrak{m}}$ are fractured into a sort of generalized relations by iterated applications of Proposition 43. Intuitively, the ordering on $\mathfrak{M}$ is understood as "dependency": every type $A : \mathcal{U}_{\bigvee_{\mathfrak{M}} \mathfrak{m}}$ is fractured into a family of type families $\{A_{\mathfrak{m}(i)}\}_{i:\mathfrak{M}}$ such that $A_{\mathfrak{m}(i)}$ depends on $A_{\mathfrak{m}(j)}$ for all $j > i$. One may also regard the underlying finite poset of $\mathfrak{M}$ as a FOLDS signature [18].

▶ **Example 44.** When $\mathfrak{M}$ is the mode sketch $\{0 \leftarrow 01 \to 1\}$, we have an equivalence

$$\mathcal{U}_{\mathfrak{m}(01)\vee\mathfrak{m}(1)\vee\mathfrak{m}(0)} \simeq \textstyle\sum_{A_0:\mathcal{U}_{\mathfrak{m}(0)}} \sum_{A_1:\mathcal{U}_{\mathfrak{m}(1)}} A_0 \to A_1 \to \mathcal{U}_{\mathfrak{m}(01)}.$$

▶ **Example 45.** When $\mathfrak{M}$ is the mode sketch $\{0 \to 1 \to 2\}$, we have an equivalence

$$\mathcal{U}_{\mathfrak{m}(0)\vee\mathfrak{m}(1)\vee\mathfrak{m}(2)} \simeq \textstyle\sum_{A_2:\mathcal{U}_{\mathfrak{m}(2)}} \sum_{A_1:A_2\to\mathcal{U}_{\mathfrak{m}(1)}} \prod_{\mathtt{x}_2} A_1(\mathtt{x}_2) \to \mathcal{U}_{\mathfrak{m}(0)}.$$

The equivalence in Proposition 43 nicely interacts with type constructors, and we derive the *logical relation translation* (also called the parametricity translation) of dependent type theory [5, 24, 32, 12] as a *theorem* in type theory. Let $\mathfrak{m}$ and $\mathfrak{n}$ be LAMs such that $\mathfrak{m} \leq {}^{\perp}\mathfrak{n}$. Type constructors in $\mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}}$ behave in the same way as the definition of the logical relation translation of type constructors [12, Section 3] as follows.

- $\mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}} : \Uparrow\mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}}$ corresponds (via Proposition 43) to the pair $(\mathcal{U}_{\mathfrak{n}}, \lambda B.B \to \mathcal{U}_{\mathfrak{m}})$.
- $\mathbf{1} : \mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}}$ corresponds to the pair $(\mathbf{1}, \lambda\_.\mathbf{1})$.
- Suppose that $A : \mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}}$ corresponds to a pair $(A_{\mathfrak{n}}, A_{\mathfrak{m}})$. Then $(A \to \mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}}) : \Uparrow\mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}}$ corresponds to the pair

  $$(A_{\mathfrak{n}} \to \mathcal{U}_{\mathfrak{n}}, \lambda B. \textstyle\prod_{\mathtt{x}:A_{\mathfrak{n}}} A_{\mathfrak{m}}(\mathtt{x}) \to B(\mathtt{x}) \to \mathcal{U}_{\mathfrak{m}}).$$

  Indeed,

  $$\begin{aligned} & A \to \mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}} \\ \simeq \quad & \{\text{fracture and gluing}\} \\ & (\textstyle\sum_{\mathtt{x}:A_{\mathfrak{n}}} A_{\mathfrak{m}}(\mathtt{x})) \to (\sum_{B:\mathcal{U}_{\mathfrak{n}}} B \to \mathcal{U}_{\mathfrak{m}}) \\ \simeq \quad & \{\textstyle\prod \text{ distributes over } \sum\} \\ & \textstyle\sum_{B:\, \prod_{\mathtt{x}:A_{\mathfrak{n}}} A_{\mathfrak{m}}(\mathtt{x}) \to \mathcal{U}_{\mathfrak{n}}} \prod_{\mathtt{x}} \prod_{\mathtt{y}} B(\mathtt{x}, \mathtt{y}) \to \mathcal{U}_{\mathfrak{m}} \\ \simeq \quad & \{\mathcal{U}_{\mathfrak{n}} \simeq (A_{\mathfrak{m}}(\mathtt{x}) \to \mathcal{U}_{\mathfrak{n}}) \text{ since } \mathfrak{m} \leq {}^{\perp}\mathfrak{n}\} \\ & \textstyle\sum_{B:A_{\mathfrak{n}}\to\mathcal{U}_{\mathfrak{n}}} \prod_{\mathtt{x}} A_{\mathfrak{m}}(\mathtt{x}) \to B(\mathtt{x}) \to \mathcal{U}_{\mathfrak{m}}. \end{aligned}$$

- Suppose that $A : \mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}}$ corresponds to a pair $(A_{\mathfrak{n}}, A_{\mathfrak{m}})$ and that $B : A \to \mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}}$ corresponds to a pair $(B_{\mathfrak{n}}, B_{\mathfrak{m}})$. Then $\prod_{\mathtt{x}:A} B(\mathtt{x}) : \mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}}$ corresponds to the pair

  $$(\textstyle\prod_{\mathtt{x}_{\mathfrak{n}}:A_{\mathfrak{n}}} B_{\mathfrak{n}}(\mathtt{x}_{\mathfrak{n}}), \lambda f. \prod_{\mathtt{x}_{\mathfrak{n}}} \prod_{\mathtt{x}_{\mathfrak{m}}:A_{\mathfrak{m}}(\mathtt{x}_{\mathfrak{n}})} B_{\mathfrak{m}}(\mathtt{x}_{\mathfrak{n}}, \mathtt{x}_{\mathfrak{m}}, f(\mathtt{x}_{\mathfrak{n}})))$$

  by a similar calculation to the previous clause. $\sum_{\mathtt{x}:A} B(\mathtt{x}) : \mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}}$ corresponds to the pair

  $$(\textstyle\sum_{\mathtt{x}_{\mathfrak{n}}:A_{\mathfrak{n}}} B_{\mathfrak{n}}(\mathtt{x}_{\mathfrak{n}}), \lambda(a_{\mathfrak{n}}, b_{\mathfrak{n}}). \sum_{\mathtt{x}_{\mathfrak{m}}:A_{\mathfrak{m}}(a_{\mathfrak{n}})} B_{\mathfrak{m}}(a_{\mathfrak{n}}, \mathtt{x}_{\mathfrak{m}}, b_{\mathfrak{n}})).$$

- Suppose that $A : \mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}}$ corresponds to a pair $(A_{\mathfrak{n}}, A_{\mathfrak{m}})$, that $a : A$ corresponds to a pair $(a_{\mathfrak{n}}, a_{\mathfrak{m}})$, and that $a' : A$ corresponds to a pair $(a'_{\mathfrak{n}}, a'_{\mathfrak{m}})$. Then $a = a' : \mathcal{U}_{\mathfrak{m}\vee\mathfrak{n}}$ corresponds to the pair

  $$(a_{\mathfrak{n}} = a'_{\mathfrak{n}}, \lambda p.a_{\mathfrak{m}} =^{A_{\mathfrak{m}}}_{p} a'_{\mathfrak{m}}).$$

Thus, any type $A : \mathcal{U}_{\mathfrak{m} \vee \mathfrak{n}}$ constructed using these type constructors is fractured into a type $A_{\mathfrak{n}} : \mathcal{U}_{\mathfrak{n}}$ and a type family $A_{\mathfrak{m}} : A_{\mathfrak{n}} \to \mathcal{U}_{\mathfrak{m}}$, and $A_{\mathfrak{m}}$ is equivalent to the logical relation translation of $A_{\mathfrak{n}}$. In this sense, *types in* $\mathcal{U}_{\mathfrak{m} \vee \mathfrak{n}}$ *are logical relations.* The interaction of the equivalences in Examples 44 and 45 and type constructors is similarly calculated. We thus conclude that types in $\mathcal{U}_{\bigvee_{\mathfrak{M}} \mathfrak{m}}$ are generalized logical relations.

5 Semantics of mode sketches

We give an overview of the semantics of mode sketches in diagrams of ∞-logoses. Many details are omitted and found in the extended version [33].

We assume that we are given Grothendieck universes $\mathfrak{U} \in \Uparrow \mathfrak{U} \in \Uparrow^2 \mathfrak{U} \in \dots$. An ∞-*logos* (over $\mathfrak{U}$) is informally an $(\infty, 1)$-category of $\mathfrak{U}$-small sheaves over a “space”. Any ∞-logos $\mathcal{L}$ is embedded into its *universe enlargement* $\Uparrow^n \mathcal{L}$, the $(\infty, 1)$-category of $(\Uparrow^n \mathfrak{U})$-small sheaves. Homotopy type theory is interpreted in any ∞-logos $\mathcal{L}$: types in $\Uparrow^n \mathcal{U}$ are interpreted as objects in $\Uparrow^n \mathcal{L}$; terms are interpreted as morphisms. Note that, instead of choosing universes in $\mathcal{L}$, we enlarge $\mathcal{L}$ with respect to the fixed Grothendieck universes to interpret large types, so there is no ambiguity in the interpretation of universes. Coherence issues in this interpretation are solved by presenting an ∞-logos by a model category [26] and then by the local universe method [15]. The *internal language* of $\mathcal{L}$ is the type theory obtained from homotopy type theory by adjoining objects and morphisms in $\Uparrow^n \mathcal{L}$ as types and terms, respectively.

Let $\mathfrak{M}$ be a mode sketch. A *model of* $\mathfrak{M}$ is an ∞-logos $\mathcal{L}$ equipped with a function $\mathfrak{m} : \mathfrak{M} \to \mathsf{LAM}$ in its internal language satisfying Axioms A–C. We write $|\mathfrak{M}|$ for the $(\infty, 2)$-category presented by $\mathfrak{M}$ as explained in Remark 28. Let $\Uparrow \mathbf{Cat}^{(2)}$ denote the $(\infty, 2)$-category of $(\Uparrow \mathfrak{U})$-small $(\infty, 1)$-categories. Let $\mathbf{Logos}^{(2)}_{\mathrm{LexAcc}} \subset \Uparrow \mathbf{Cat}^{(2)}$ denote the locally full subcategory whose 0-cells are the ∞-logoses and whose 1-cells are the accessible functors preserving finite limits. For an $(\infty, 2)$-category $\mathcal{C}$, let $\mathcal{C}^{\mathrm{op}(1,2)}$ denote the $(\infty, 2)$-category obtained from $\mathcal{C}$ by reversing the directions of 1-cells and 2-cells.

▶ **Construction 46.** Let $\mathcal{L}$ be an ∞-logos. For a LAM $\mathfrak{m}$ in the internal language of $\mathcal{L}$, the *externalization* of $\mathcal{U}_{\mathfrak{m}}$ is the full subcategory of $\mathcal{L}$ spanned by the $\mathfrak{m}$-modal types. For a function $\mathfrak{m} : \mathfrak{M} \to \mathsf{LAM}$ from a mode sketch $\mathfrak{M}$ in the internal language of $\mathcal{L}$, the externalizations of $\mathcal{U}_{\mathfrak{m}(i)}$'s and the functions $\bigcirc^{\mathfrak{m}(j)}_{\mathfrak{m}(i)}$ and $\eta^{\mathfrak{m}(i);\mathfrak{m}(k)}_{\mathfrak{m}(j)}$ form a functor $|\mathfrak{M}|^{\mathrm{op}(1,2)} \to \Uparrow \mathbf{Cat}^{(2)}$ which we call the *externalization* of the diagram $\{\mathcal{U}_{\mathfrak{m}(i)}\}_{i:\mathfrak{M}}$. It turns out that this functor factors through $\mathbf{Logos}^{(2)}_{\mathrm{LexAcc}}$ by verifying that the LAMs in the internal language of $\mathcal{L}$ correspond to the lex, accessible *localizations* of $\mathcal{L}$ [33, Section 8].

▶ **Construction 47.** Let I be a small $(\infty, 2)$-category and $\mathcal{C} : I^{\mathrm{op}(1,2)} \to \Uparrow \mathbf{Cat}^{(2)}$ a functor. The *oplax limit of* $\mathcal{C}$ is the $(\infty, 1)$-category $\mathrm{opLaxLim}_{i \in I} \mathcal{C}_i$ described as follows. An object x in $\mathrm{opLaxLim}_{i \in I} \mathcal{C}_i$ consists of: an object $x_i \in \mathcal{C}_i$ for any object $i \in I$; a morphism $x_\alpha : x_i \to \mathcal{C}_\alpha(x_j)$ for any morphism $\alpha : i \to j$ in I; some coherence data. A morphism $u : x \to y$ in $\mathrm{opLaxLim}_{i \in I} \mathcal{C}_i$ consists of: a morphism $u_i : x_i \to y_i$ for any object $i \in I$; a homotopy u_α filling the square

$$\begin{tikzcd}
x_i \arrow[r, "u_i"] \arrow[d, "x_\alpha"'] & y_i \arrow[d, "y_\alpha"] \\
\mathcal{C}_\alpha(x_j) \arrow[r, "\mathcal{C}_\alpha(u_j)"'] & \mathcal{C}_\alpha(y_j)
\end{tikzcd}$$

for any morphism $\alpha : i \to j$ in I; some coherence data. See [33, Section 9] for more explicit construction.

▶ **Theorem 48.** *For any mode sketch $\mathfrak{M}$, we have an equivalence between the following spaces:*

- *the space of models of $\mathfrak{M}$;*
- *the space of functors $|\mathfrak{M}|^{\mathrm{op}(1,2)} \to \mathbf{Logos}^{(2)}_{\mathrm{LexAcc}}$.*

Moreover, when a model $(\mathcal{L}, \mathfrak{m})$ of $\mathfrak{M}$ corresponds to a functor $\mathcal{K} : |\mathfrak{M}|^{\mathrm{op}(1,2)} \to \mathbf{Logos}^{(2)}_{\mathrm{LexAcc}}$, the following hold.

1. $\mathcal{L} \simeq \mathrm{opLaxLim}_{i \in |\mathfrak{M}|} \mathcal{K}_i$
2. *$\mathcal{K}$ is the externalization of the diagram $\{\mathcal{U}_{\mathfrak{m}(i)}\}_{i:\mathfrak{M}}$ in the internal language of $\mathcal{L}$.*

Sketch of proof. Let $\mathcal{K} : |\mathfrak{M}|^{\mathrm{op}(1,2)} \to \mathbf{Logos}^{(2)}_{\mathrm{LexAcc}}$ be a functor. We define $\mathcal{L} = \mathrm{opLaxLim}_{i \in |\mathfrak{M}|} \mathcal{K}_i$. For a cosieve S on $\mathfrak{M}$, we define $\psi_{\mathcal{K}}(S) \in \mathcal{L}$ by

$$\psi_{\mathcal{K}}(S)_i = \begin{cases} \mathbf{1} & \text{if } i \in S \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

The other components are uniquely determined by the universal properties of initial and final objects. This determines a lattice morphism $\psi_{\mathcal{K}} : \mathsf{coSieve}(\mathfrak{M}) \to \mathsf{Prop}$ in the internal language of $\mathcal{L}$. By Theorem 37, this corresponds to a function $\mathfrak{m} : \mathfrak{M} \to \mathsf{LAM}$ satisfying Axioms A and C. One can show that the induced diagram $\{\mathcal{U}_{\mathfrak{m}(i)}\}_{i:\mathfrak{M}}$ is interpreted as the given diagram $\mathcal{K}$, from which it follows that $\mathfrak{m}$ also satisfies Axiom B. Thus, $\mathcal{L}$ is part of a model of $\mathfrak{M}$. This construction is an equivalence by externalizing the argument of exhibiting $\mathcal{U} \simeq \mathcal{U}_{\bigvee_{\mathfrak{M}} \mathfrak{m}}$ as the oplax limit of the diagram $\{\mathcal{U}_{\mathfrak{m}(i)}\}_{i:\mathfrak{M}}$ (Section 3.3). ◀

References

1 Mathieu Anel, Georg Biedermann, Eric Finster, and André Joyal. A generalized Blakers-Massey theorem. *J. Topol.*, 13(4):1521–1553, 2020. doi:10.1112/topo.12163.

2 Mathieu Anel and André Joyal. Topo-logie. In *New spaces in mathematics. Formal and conceptual reflections*, pages 155–257. Cambridge University Press, 2021. doi:10.1017/9781108854429.007.

3 Peter Arndt and Krzysztof Kapulkin. Homotopy-Theoretic Models of Type Theory. In Luke Ong, editor, *Typed Lambda Calculi and Applications: 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011. Proceedings*, pages 45–60, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. doi:10.1007/978-3-642-21691-6_7.

4 Steve Awodey and Michael A. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146(1):45–55, 2009. doi:10.1017/S0305004108001783.

5 Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Proofs for Free: Parametricity for Dependent Types. *Journal of Functional Programming*, 22(2):107–152, March 2012. doi:10.1017/S0956796812000056.

6 John W. Duskin. Simplicial matrices and the nerves of weak n-categories. I. Nerves of bicategories. *Theory Appl. Categ.*, 9:198–308, 2001/02. CT2000 Conference (Como). URL: http://www.tac.mta.ca/tac/volumes/9/n10/9-10abs.html.

7 Eric Finster. A Note on Left Exact Modalities in Type Theory. URL: https://ericfinster.github.io/files/lmhtt.pdf.

8 Daniel Gratzer. Normalization for Multimodal Type Theory. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 2:1–2:13. ACM, 2022. doi:10.1145/3531130.3532398.

9 Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal Dependent Type Theory. *Logical Methods in Computer Science*, Volume 17, Issue 3, July 2021. doi:10.46298/lmcs-17(3:11)2021.

10 Kuen-Bang Hou (Favonia), Eric Finster, Daniel R. Licata, and Peter LeFanu Lumsdaine. A Mechanization of the Blakers-Massey Connectivity Theorem in Homotopy Type Theory. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, pages 565–574, New York, NY, USA, 2016. ACM. doi:10.1145/2933575.2934545.

11 Peter T. Johnstone. *Sketches of an Elephant : A Topos Theory Compendium Volume 1*, volume 43 of *Oxford Logic Guides*. Oxford University Press, 2002.

12 Marc Lasson. Canonicity of Weak ω-groupoid Laws Using Parametricity Theory. *Electronic Notes in Theoretical Computer Science*, 308:229–244, 2014. doi:10.1016/j.entcs.2014.10.013.

13 F. William Lawvere. Axiomatic cohesion. *Theory Appl. Categ.*, 19:No. 3, 41–49, 2007. URL: http://www.tac.mta.ca/tac/volumes/19/3/19-03abs.html.

14 Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. Internal Universes in Models of Homotopy Type Theory. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018)*, volume 108 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSCD.2018.22.

15 Peter LeFanu Lumsdaine and Michael A. Warren. The Local Universes Model: An Overlooked Coherence Construction for Dependent Type Theories. *ACM Trans. Comput. Logic*, 16(3):23:1–23:31, July 2015. doi:10.1145/2754931.

16 Jacob Lurie. *Higher Topos Theory*, volume 170 of *Annals of Mathematics Studies*. Princeton University Press, 2009. URL: https://www.math.ias.edu/~lurie/papers/HTT.pdf.

17 Jacob Lurie. $(\infty, 2)$-Categories and the Goodwillie Calculus I, 2009. arXiv:0905.0462v2.

18 Michael Makkai. First Order Logic with Dependent Sorts, with Applications to Category Theory, 1995. URL: http://www.math.mcgill.ca/makkai/folds/foldsinpdf/FOLDS.pdf.

19 Per Martin-Löf. An Intuitionistic Theory of Types: Predicative Part. *Studies in Logic and the Foundations of Mathematics*, 80:73–118, 1975. doi:10.1016/S0049-237X(08)71945-1.

20 Hoang Kim Nguyen and Taichi Uemura. ∞-type theories, 2022. arXiv:2205.00798v1.

21 Egbert Rijke. Introduction to Homotopy Type Theory, 2022. arXiv:2212.11082v1.

22 Egbert Rijke, Michael Shulman, and Bas Spitters. Modalities in homotopy type theory. *Log. Methods Comput. Sci.*, 16(1):Paper No. 2, 79, 2020. doi:10.23638/LMCS-16(1:2)2020.

23 Urs Schreiber and Michael Shulman. Quantum Gauge Field Theory in Cohesive Homotopy Type Theory. In Ross Duncan and Prakash Panangaden, editors, *Proceedings 9th Workshop on Quantum Physics and Logic, QPL 2012, Brussels, Belgium, 10-12 October 2012*, volume 158 of *EPTCS*, pages 109–126, 2012. doi:10.4204/EPTCS.158.8.

24 Michael Shulman. Univalence for inverse diagrams and homotopy canonicity. *Mathematical Structures in Computer Science*, 25(05):1203–1277, 2015. doi:10.1017/s0960129514000565.

25 Michael Shulman. Brouwer's fixed-point theorem in real-cohesive homotopy type theory. *Mathematical Structures in Computer Science*, 28(6):856–941, 2018. doi:10.1017/S0960129517000147.

26 Michael Shulman. All $(\infty, 1)$-toposes have strict univalent universes, 2019. arXiv:1904.07004v2.

27 Jonathan Sterling. *First Steps in Synthetic Tait Computability*. PhD thesis, Carnegie Mellon University, 2021. URL: https://www.jonmsterling.com/pdfs/sterling:2021:thesis.pdf.

28 Jonathan Sterling and Carlo Angiuli. Normalization for Cubical Type Theory. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–15, 2021. doi:10.1109/LICS52264.2021.9470719.

29 Jonathan Sterling and Robert Harper. Logical Relations as Types: Proof-Relevant Parametricity for Program Modules. *J. ACM*, 68(6):41:1–41:47, 2021. doi:10.1145/3474834.

30 Jonathan Sterling and Robert Harper. Sheaf Semantics of Termination-Insensitive Noninterference. In Amy P. Felty, editor, *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*, volume 228 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:19, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSCD.2022.5.

31 The Univalent Foundations Program. Homotopy Type Theory: Univalent Foundations of Mathematics, 2013. URL: http://homotopytypetheory.org/book/.

32 Taichi Uemura. Fibred Fibration Categories. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, June 2017. doi:10.1109/LICS.2017.8005084.

33 Taichi Uemura. Homotopy type theory as internal languages of diagrams of ∞-logoses, 2022. arXiv:2212.02444v1.

34 Taichi Uemura. Normalization and coherence for ∞-type theories, 2022. arXiv:2212.11764v1.

35 Gavin Wraith. Artin glueing. *J. Pure Appl. Algebra*, 4:345–348, 1974. doi:10.1016/0022-4049(74)90014-0.

The Formal Theory of Monads, Univalently

Niels van der Weide
Institute for Computing and Information Sciences, Radboud University, Nijmegen, The Netherlands

Abstract

We develop the formal theory of monads, as established by Street, in univalent foundations. This allows us to formally reason about various kinds of monads on the right level of abstraction. In particular, we define the bicategory of monads internal to a bicategory, and prove that it is univalent. We also define Eilenberg-Moore objects, and we show that both Eilenberg-Moore categories and Kleisli categories give rise to Eilenberg-Moore objects. Finally, we relate monads and adjunctions in arbitrary bicategories. Our work is formalized in Coq using the `UniMath` library.

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

Keywords and phrases bicategory theory, univalent foundations, formalization, monads, Coq

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.6

Related Version *Preprint*: `https://arxiv.org/abs/2212.08515`

Supplementary Material *Software*: `https://doi.org/10.5281/zenodo.7848572` [53]
archived at `swh:1:dir:b2edfe385dd6437e5efc4fdc18c9bd48e006a144`

Funding This research was supported by the NWO project "The Power of Equality" OCENW.M20.380, which is financed by the Dutch Research Council (NWO).

Acknowledgements The author thanks Benedikt Ahrens, Deivid Vale, Herman Geuvers, and Kobe Wullaert for their comments on earlier versions of this paper. The author also thanks the Coq developers for providing the Coq proof assistant and their continuous support to keep `UniMath` compatible with Coq.

1 Introduction

Monads are ubiquitous in both mathematics and computer science, and many different kinds of monads have been considered in various settings. In functional programming, monads are used to capture computational effects [22]. Strong monads have been used to provide semantics of programming languages such as Moggi's computational λ-calculus [35, 36] and models of call-by-push-value [30]. Monads are also used in algebra to represent algebraic theories, and in fact, the class of algebraic theories is equivalent to a class of monads [20]. This result has been adapted to the enriched case as well in order to relate various notions of computation with enriched monads [16, 39, 40, 42]. Comonads, the dual notion of monads, found applications in the semantics of linear logic [11, 34].

A general setting in which all these different variations of monads can be studied, has been developed by Street [47]. This setting, known as *the formal theory of monads*, uses the fact that the notion of monad can be defined internal to an arbitrary bicategory [10], including 2-categories (which were used by Street). Each of the aforementioned kinds of monads is actually an instance of this more general notion. For example, monads in the bicategory of symmetric monoidal categories are symmetric monoidal monads, and strong monads are monads in the bicategory of so-called left actegories [13]. Comonads in a bicategory B are the same as monads in B^{co}, which is B with the 2-cells reversed. Even several kinds of

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 6; pp. 6:1–6:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Various notions of monads.

Bicategory	Notion of monad
Symmetric monoidal categories	Symmetric monoidal monad
Actegories [13]	Strong monad
Enriched categories [25]	Enriched monad
Bicategory of monads	Distributive law

distributive laws, including mixed distributive laws [12, 44] and iterated distributive laws [14], are instances of this notion of monad. An overview of the different kinds of monads internal to various bicategories can be found in Table 1. As such, the formal theory of monads provides a general setting to study various kinds of monads.

Foundations. In this paper, we work in **univalent foundations** [50]. Univalent foundations is an extension of intensional type theory with the univalence axiom. Roughly speaking, this axiom says that equivalent types are equal. More precisely, we have a map $\mathsf{idtoequiv}$ that sends identities $X = Y$ to equivalences from X to Y, and the univalence axiom states that $\mathsf{idtoequiv}$ is an equivalence itself. This axiom has numerous effects on the mathematics in this foundation. One consequence is function extensionality and another is that identity types must necessarily be proof relevant: since there could be multiple equivalence between two types, we could have different proofs of their equality.

In addition, the notion of category studied in this setting is that of **univalent categories** [2]. In every category C, we have a map $\mathsf{idtoiso}_{x,y} : x = y \to x \cong y$, and C is **univalent** if $\mathsf{idtoiso}_{x,y}$ is an equivalence for all $x, y : \mathsf{C}$. From the univalence axiom, one can deduce that the category of sets is univalent. The reason why this notion is interesting, is because in the set-theoretical semantics [23], univalent categories correspond to ordinary categories. Furthermore, every property expressible in type theory about univalent categories is closed under equivalence.

However, there are some challenges when working with univalent categories. For instance, the usual definition of the *Kleisli category* [31] does not give rise to a univalent category, so this category does not actually define a Kleisli category in univalent foundations. A solution to this problem has already been given: we need to use its Rezk completion [6]. However, the necessary theorems about the Kleisli category (e.g., every monad gives rise to an adjunction via the Kleisli category) have not been proven in that work.

In the present paper, we study and formalize the formal theory of monads by Street [47] in univalent foundations. More specifically, we formalize the key notions, which are the bicategory of monads and Eilenberg-Moore objects, and we illustrate them with numerous examples. We also prove the two main theorems that relate monads to adjunctions: every adjunction gives rise to a monad and in a bicategory with Eilenberg-Moore objects, every monad gives rise to an adjunction. In addition, we instantiate the formal theory of monads to deduce the main theorems about Kleisli categories. The contributions of this paper is the development of the formal theory of monads in univalent foundations and a proof that in univalent foundations, every monad gives rise to an adjunction via the Kleisli category.

The abstract setting provided by the formal theory of monads is beneficial for formalization, which is our main motivation for this work. The main theorems are only proven once in this setting, and afterwards they are instantiated to the relevant cases of interest without the need of reproving anything. In addition, we think that this work would be useful to formalize the categorical semantics of linear logic [34] or the enriched effect calculus [16].

Formalization. The results in this paper are formalized using the Coq proof assistant [48], and they are integrated in the `UniMath` library [53]. `UniMath` is under constant development, and the paper refer to the version with `git` hash 2f79746. The formalization consists of around 12,000 lines of code. More specifically, the tool `coqwc` gives the following count:

```
spec    proof comments
4470     7839      182 total
```

The main difficulty arises from the coherences that have to be proven in this development. Since we use bicategories rather than 2-categories, every coherence becomes more complicated. There also are points where univalence helps with obtaining simple and elegant proofs, such as Propositions 3.7 and 5.5. Displayed bicategories play a fundamental role in Section 3, where they are used to give a modular proof that the bicategory of monads is univalent.

Definitions, theorems, constructions, and examples in this paper are accompanied with a link that points to the relevant definition in the formalization. For example, `bicat` refers to the definition of a bicategory.

Related work. The formal theory of monads was originally developed by Street [47], and later extended by Lack and Street [28]. There are two differences between our work and Street's work. First of all, Street used strict 2-categories while we use bicategories. As has already been noticed by Lack [26], this difference is rather minor. The resulting definitions are similar: the only difference is that associators and unitors have to be put on the right places. More fundamental is the second difference: we work in univalent foundations and univalent (bi)categories whereas Street works in set-theoretic foundations. This affects the development in several ways. While both bicategories and strict 2-categories have been defined and studied in a univalent setting [1], a coherence theorem [32, 41] has not been proven in this setting. In addition, since we work in an intensional setting, working with a strict 2-category is not significantly more convenient than working with a bicategory. The reason for that, is that equality proofs of associativity and unitality are present in terms to guarantee that the whole expression is well-typed. As such, a coherence theorem would only have limited usability in our setting compared to a classical one. Another difference is that in our framework, the usual definition of the Kleisli category does not give rise to a univalent category, and we need to work with its Rezk completion instead. An overview of the main notions in bicategory theory can be found in various sources [10, 21, 27, 29].

Several formalizations have results about bicategory theory. The coherence theorem [29] is formalized in both Isabelle [37, 46] and Lean [33], but neither of those are based on univalent foundations. Some notions in bicategory theory have been formalized in Agda [38], namely in the 1Lab [49] and the Agda-categories library [19]. However, neither of these cover the formal theory of monads. We use `UniMath` [53] and its formalization of bicategories [1, 51]. Formalizations on category theory are more plentiful, and an overview can be found in [19]. Within the framework of univalent foundations, there is the HoTT library [9, 18], Agda-UniMath [45], and Cubical Agda [52]. Ahrens, Matthes, and Mörtberg formalized monads of categories in `UniMath` [53]. They also defined a notion of signature, that allows for binding, and they showed that every signature gives rise to a monad [4, 5].

Overview. We start this paper by recalling some preliminary notions in Section 2. Next we construct in Section 3 the bicategory of monads internal to bicategories and we prove that it is univalent. We illustrate the material of Section 3 with various examples in Section 4. In Section 5 we discuss Eilenberg-Moore objects. We follow that up in Section 6 by using Kleisli categories to construct Eilenberg-Moore objects in the opposite bicategory. In Sections 7 and 8 we prove some theorems in this setting. We prove in Section 7 that every adjunction gives rise to a monad and that every monad gives rise to an adjunction under mild assumptions. In Section 8 we define the notion of monadic adjunctions in an arbitrary bicategory and we characterize those using monadic adjunctions in categories. We conclude in Section 9.

2 Preliminaries

In this section, we briefly recall some of the basic notions needed in this paper. First of all, we use the notions of *propositions* and *sets* from univalent foundations. Types A for which we have $x = y$ for all $x, y : A$, are called **propositions**, and types A for which every $x = y$ is a proposition, are called **sets**. In addition, we assume that our foundation supports the **propositional truncation**: the truncation $\|A\|$ is A with all its elements identified. More concretely, we have a map $A \to \|A\|$ and for all $x, y : \|A\|$, we have $x = y$. Next we discuss some notions from bicategory theory [1, 10, 29], and we start with *bicategories*.

▶ **Definition 2.1** (`bicat`). A **bicategory** B consists of a type B of objects, for every $x, y : \mathsf{B}$ a type $x \to y$ of 1-cells, and for every $f, g : x \to y$, a *set* $f \Rightarrow g$. On this data, we have the following operations:

- For every $x : \mathsf{B}$, a 1-cell $\mathrm{id}_x : x \to x$;
- For all 1-cells $f : x \to y$ and $g : y \to z$, a 1-cell $f \cdot g : x \to z$;
- For every 1-cell $f : x \to y$, a 2-cell $\mathrm{id}_f : f \Rightarrow f$;
- For all 2-cells $\theta : f \Rightarrow g$ and $\tau : g \Rightarrow h$, a 2-cell $\theta \bullet \tau : f \Rightarrow h$;
- For every 1-cell $f : x \to y$, invertible 2-cells $\lambda_f : \mathrm{id}_x \cdot f \Rightarrow f$ and $\rho_f : f \cdot \mathrm{id}_y \Rightarrow f$;
- For all 1-cells $f : w \to x$, $g : x \to y$, and $h : y \to z$, an invertible 2-cell $\alpha_{f,g,h} : (f \cdot g) \cdot h \Rightarrow f \cdot (g \cdot h)$.

If the relevant 1-cells are clear from the context, we write λ, ρ, and α instead of λ_f, ρ_f, and $\alpha_{f,g,h}$ respectively. We can also whisker 2-cells with 1-cells in two ways. Given 1-cells and 2-cells as depicted in the diagram on the left below, we have a 2-cell $f \lhd \tau : f \cdot g_1 \Rightarrow f \cdot g_2$, and from 1-cells and 2-cells as depicted in the diagram on the right below, we get a 2-cell $\tau \rhd g : f_1 \cdot g \Rightarrow f_2 \cdot g$.

$$\begin{tikzcd} x \arrow[r, "f"'] & y \arrow[r, shift left=2, "g_1"] \arrow[r, shift right=2, "g_2"'] \arrow[r, phantom, "\Downarrow\tau"] & z \end{tikzcd}$$

$$\begin{tikzcd} x \arrow[r, shift left=2, "f_1"] \arrow[r, shift right=2, "f_2"'] \arrow[r, phantom, "\Downarrow\tau"] & y \arrow[r, "g"] & z \end{tikzcd}$$

The laws that need to be satisfied, can be found in the literature [1, Definition 2.1].

Note that we use diagrammatic order for composition instead of compositional order. We use the notation $\underline{\mathsf{B}(x, y)}$ for the category whose objects are 1-cells $f : x \to y$ and whose morphisms from $f : x \to y$ to $g : x \to y$ are 2-cells $\tau : f \Rightarrow g$. Given a 1-cell $f : x \to y$ and an object $w : \mathsf{B}$, we have a functor $(- \cdot f)_w : \underline{\mathsf{B}(w, x)} \to \underline{\mathsf{B}(w, y)}$, which sends a 1-cell $g : w \to x$ to $g \cdot f$ and a 2-cell $\tau : g_1 \Rightarrow g_2$ to $\tau \rhd f$.

The core example of a bicategory in this paper is $\mathsf{UnivCat}$. Its objects are *univalent* categories, the 1-cells are functors, and the 2-cells are natural transformations. We also have a bicategory Cat whose objects are (not necessarily univalent) categories, 1-cells are functors, and 2-cells are natural transformations.

In this paper, we also make use of *univalent* bicategories. To define this property, we use that between every two objects $x, y : \mathsf{B}$, we have a type $x \simeq y$ of *adjoint equivalences* between them. In addition, for all 1-cells $f, g : x \to y$, there is a type $f \cong g$ of *invertible 2-cells* between them. For the precise definition of these notions, we refer the reader to the literature [1, Definitions 2.4 and 2.5].

▶ **Definition 2.2.** Let B be a bicategory

- (`is_univalent_2_1`) Using path induction, we define a function $\mathsf{idtoiso}^{2,1}_{f,g} : f = g \to f \cong g$ for all 1-cells $f, g : x \to y$. A bicategory is **locally univalent** if $\mathsf{idtoiso}^{2,1}_{f,g}$ is an equivalence for all f and g.
- (`is_univalent_2_0`) Using path induction, we define a function $\mathsf{idtoiso}^{2,0}_{x,y} : x = y \to x \simeq y$ for all objects x and y. We say that B is **globally univalent** if $\mathsf{idtoiso}^{2,0}_{x,y}$ is an equivalence for all x and y.
- (`is_univalent_2`) A bicategory is **univalent** if it is both locally and globally univalent.

The bicategory $\mathsf{UnivCat}$ of univalent categories is both locally and globally univalent. However, Cat, whose objects objects are not required to be univalent, is neither.

If we have a bicategory B, then we define the bicategory B^{op} by 'reversing the 1-cells' in B. More precisely, objects are the same as objects in B, 1-cells from x to y in B^{op} are 1-cells $y \to x$ in B, while 2-cells from $f : y \to x$ to $g : y \to x$ are 2-cells $f \Rightarrow g$ in B. In addition, from a bicategory B, we obtain B^{co} by 'reversing the 2-cells'. Objects and 1-cells in B^{co} are the same as objects and 1-cells in B respectively, but a 2-cell from f to g in B^{co} is a 2-cell $g \Rightarrow f$ in B. Next we define *pseudofunctors*.

▶ **Definition 2.3** (`psfunctor`)**.** Let B_1 and B_2 be bicategories. A **pseudofunctor** $F : \mathsf{B}_1 \to \mathsf{B}_2$ consists of

- A function $F : \mathsf{B}_1 \to \mathsf{B}_2$;
- For every $x, y : \mathsf{B}_1$ a function that maps $f : x \to y$ to $F\,f : F\,x \to F\,y$;
- For all 1-cell $f, g : x \to y$ a function that maps $\theta : f \Rightarrow g$ to $F\,\theta : F\,f \Rightarrow F\,g$;
- For every $x : \mathsf{B}_1$, an invertible 2-cell $F_i(x) : \mathrm{id}_{F\,x} \Rightarrow F(\mathrm{id}_x)$;
- For all $f : x \to y$ and $g : y \to z$, an invertible 2-cell $F_c(f, g) : F(f) \cdot F(g) \Rightarrow F(f \cdot g)$.

The coherences that need to be satisfied, can be found in the literature [1, Definition 2.12].

In applications, we are interested in a wide variety of bicategories beside $\mathsf{UnivCat}$, and among those are $\mathsf{UnivCat}_{\mathsf{Terminal}}$ and $\mathsf{SymMonUnivCat}$. These examples have something in common: their objects are categories equipped with some extra structure, the 1-cells are structure preserving functors, while the 2-cells are structure preserving natural transformations. We capture this pattern using *displayed bicategories*. This notion is an adaptation of *displayed categories* to the bicategorical setting [3].

To get an idea of what displayed bicategories are, let us first briefly discuss displayed categories. A displayed category D over C represents structure and properties to be added to the objects and morphisms of C. For every object $x : \mathsf{C}$, we have a type of *displayed objects* D_x and for every morphism $f : x \to y$ and displayed objects $\overline{x} : \mathsf{D}_x$ and $\overline{y} : \mathsf{D}_y$, we have a set $\overline{x} \xrightarrow{f} \overline{y}$ of *displayed morphisms*. For example, if for C we take the category of sets, an example of a displayed category would be group structures. The displayed objects over a set X are group structures over X, while the displayed morphisms over $f : X \to Y$ between two group structures are proofs that f preserves the group operations. Every displayed category D gives rise to the *total category* $\int \mathsf{D}$ and a functor $\pi_{\mathsf{D}} : \int \mathsf{D} \to \mathsf{C}$. In the example we mentioned before, $\int \mathsf{D}$ would be the category of groups and π_{D} maps a group to its underlying set.

In the bicategorical setting, we use a similar approach, but a displayed bicategory should not only have displayed objects and 1-cells, but also displayed 2-cells. More precisely, we define displayed bicategories as follows.

▶ **Definition 2.4** (`disp_bicat`). A **displayed bicategory** D over a bicategory B consists of

- For every $x : \mathsf{B}$ a type D_x of **displayed objects**;
- For all 1-cells $f : x \to y$ and displayed objects $\overline{x} : D_x$ and $\overline{y} : D_y$, a type $\overline{x} \xrightarrow{f} \overline{y}$ of **displayed morphisms**;
- For all 2-cells $\theta : f \Rightarrow g$ and displayed morphisms $\overline{f} : \overline{x} \xrightarrow{f} \overline{y}$ and $\overline{g} : \overline{x} \xrightarrow{g} \overline{y}$ a set $\overline{f} \overset{\theta}{\Rightarrow} \overline{g}$ of **displayed 2-cells**.

In addition, there are displayed versions of every operation of bicategories. For example, for every $x : \mathsf{B}$ and $\overline{x} : D_x$, we have the displayed identity $\overline{\mathrm{id}_x} : \overline{x} \xrightarrow{\mathrm{id}_x} \overline{x}$, and from two displayed morphisms $\overline{f} : \overline{x} \xrightarrow{f} \overline{y}$ and $\overline{g} : \overline{y} \xrightarrow{g} \overline{z}$ we get a displayed morphism $\overline{f} \cdot \overline{g} : \overline{x} \xrightarrow{f \cdot g} \overline{z}$. A list of the operations and laws can be found in the literature [1, Definition 6.1].

Just like for displayed categories, every displayed bicategory D gives rise to bicategory $\int \mathsf{D}$ and a pseudofunctor $\int \mathsf{D} \to \mathsf{B}$.

▶ **Problem 2.5.** *Given a displayed bicategory* D *over* B*, to construct a bicategory* $\int \mathsf{D}$ *and a pseudofunctor* $\pi_{\mathsf{D}} : \int \mathsf{D} \to \mathsf{B}$.

▶ **Construction 2.6** (for Problem 2.5; `total_bicat`)**.** The bicategory $\int \mathsf{D}$ is defined as follows:

- Its objects are pairs $x : \mathsf{B}$ together with $\overline{x} : \mathsf{D}_x$;
- Its 1-cells from $(x, \overline{x})$ to $(y, \overline{y})$ are pairs $f : x \to y$ together with $\overline{f} : \overline{x} \xrightarrow{f} \overline{y}$;
- Its 2-cells from $(f, \overline{f})$ to $(g, \overline{g})$ are pairs $\tau : f \Rightarrow g$ together with $\overline{\tau} : \overline{f} \overset{\tau}{\Rightarrow} \overline{g}$.

The pseudofunctor π_{D} sends objects $(x, \overline{x})$ to x, 1-cells $(f, \overline{f})$ to f, and 2-cells $(\theta, \overline{\theta})$ to θ. ⌟

Univalent displayed bicategories are defined similarly to univalent bicategories, and the precise definition can be found elsewhere [1, Definition 7.3]. If we have a displayed bicategory D over B and if both B and D are univalent, then $\int \mathsf{D}$ is univalent as well. This gives a modular way to construct univalent bicategories.

In numerous different examples, we look at displayed bicategories whose displayed 2-cells are actually trivial, in a certain sense. More precisely, we look at two properties. One of them expresses that between all displayed 1-cells $\overline{f} : \overline{x} \xrightarrow{f} \overline{y}$ and $\overline{g} : \overline{x} \xrightarrow{g} \overline{y}$, there is at most one displayed 2-cell. A displayed bicategory satisfying hat property, is called a *local preorder*: it expresses that the type $\overline{f} \overset{\tau}{\Rightarrow} \overline{g}$ is a proposition. The other property is *locally groupoidal*, and it says that every displayed 2-cell over an invertible 2-cell is again invertible.

▶ **Definition 2.7** (`disp_cell_unit_bicat`). Suppose, we have a bicategory B and

- For each $x : \mathsf{B}$ a type D_x;
- For every 1-cell $f : x \to y$ and elements $\overline{x} : \mathsf{D}_x$ and $\overline{y} : \mathsf{D}_y$ a type $\mathsf{D}_{\overline{x},\overline{y},f}$;
- For every $x : \mathsf{B}$ and $\overline{x} : \mathsf{D}_x$ an inhabitant of $\mathsf{D}_{\overline{x},\overline{x},\mathrm{id}_x}$;
- For all $\overline{f} : \mathsf{D}_{\overline{x},\overline{y},f}$ and $\overline{g} : \mathsf{D}_{\overline{y},\overline{z},g}$ an inhabitant of $\mathsf{D}_{\overline{x},\overline{z},f \cdot g}$.

Then we get a displayed bicategory over B whose 2-cells are inhabitants of the unit type.

Note that every displayed bicategory constructed using Definition 2.7 is both a local preorder and locally groupoidal. To illustrate the notion of displayed category, let us look at some examples. These were already considered in previous work [7, 54].

▶ **Example 2.8** (`univ_cat_with_terminal_obj`). Using Definition 2.7, we define a displayed bicategory $\mathsf{dUnivCat}_{\mathsf{Terminal}}$ over $\mathsf{UnivCat}$.

- Objects over C are terminal objects T_{C} in C;
- Displayed 1-cells over $F : \mathsf{C}_1 \to \mathsf{C}_2$ are proofs that F preserves terminal objects.

We define $\mathsf{UnivCat}_{\mathsf{Terminal}}$ to be $\int \mathsf{dUnivCat}_{\mathsf{Terminal}}$.

▶ **Example 2.9** (`disp_bicat_univmon`)**.** We define a displayed bicategory $\mathsf{dMonUnivCat}$ over $\mathsf{UnivCat}$ as follows:

- Objects over C are monoidal structures over C;
- 1-cells over $F : \mathsf{C}_1 \to \mathsf{C}_2$ are structures that F is a lax monoidal functor;
- 2-cells over $\theta : F \Rightarrow G$ are proofs that θ is a monoidal transformation.

We denote its total bicategory by $\mathsf{MonUnivCat}$.

Note that similarly, one can define a displayed bicategory $\mathsf{dSymMonUnivCat}$ over $\mathsf{UnivCat}$ whose total bicategory $\mathsf{SymMonUnivCat}$ is the bicategory of symmetric monoidal categories. In the remainder of this paper, we use several operations on displayed bicategories.

▶ **Definition 2.10.** We have the following operations on displayed bicategories:

1. (`disp_dirprod_bicat`) Given displayed bicategories D_1 and D_2 over B, we construct a displayed bicategory $\mathsf{D}_1 \times \mathsf{D}_2$ over B whose objects, 1-cells and 2-cells are pairs of objects, 1-cells, and 2-cells in D_1 and D_2 respectively.
2. (`sigma_bicat`) Suppose that we have a displayed bicategory D_1 over B and a displayed bicategory D_2 over $\int \mathsf{D}_1$. We construct a displayed bicategory $\sum(\mathsf{D}_2)$ over B by setting the objects over x to be pairs of $(\overline{x}, \overline{\overline{x}})$ of objects $\overline{x} : D_{1x}$ and $\overline{\overline{x}} : \mathsf{D}_{2(x,\overline{x})}$.
3. (`disp_fullsubbicat`) Let B be a bicategory and let P be a predicate on the objects of B. We define a displayed bicategory $\mathsf{dFullSub}(P)$ over B by setting the displayed objects over x to be proofs of $P(x)$ and the 1-cells and 2-cells are inhabitants of the unit type.

The last notion we discuss, is the notion of a *section* of a displayed bicategory. Intuitively, a section assigns to every x a displayed object $\overline{x} : \mathsf{D}_x$ in a pseudofunctorial way. As such every section s of D induces a pseudofunctor $\mathsf{Section}(s) : \mathsf{B} \to \int D$ such that every $x : \mathsf{B}$ is *definitionally* equal to $\pi_{\mathsf{D}}(\mathsf{Section}(s)(x))$.

▶ **Definition 2.11** (`section_disp_bicat`)**.** A **section** s of D consists of

- For every object $x : \mathsf{B}$ a displayed object $s_x : D_x$;
- For every 1-cell $f : x \to y$ a displayed 1-cell $s_f : s_x \xrightarrow{f} s_y$;
- For every 2-cell $\theta : f \Rightarrow g$ a displayed 2-cell $s_\theta : s_f \overset{\theta}{\Rightarrow} s_g$;
- For every $x : \mathsf{B}$ an invertible 2-cell $s_i(x) : \overline{\mathrm{id}_{s_x}} \xRightarrow{\mathrm{id}_{\mathrm{id}_x}} s_{\mathrm{id}_x}$;
- For every $f : x \to y$ and $g : y \to z$ an invertible 2-cell $s_c(f,g) : s_f \cdot s_g \xRightarrow{\mathrm{id}_{f \cdot g}} s_{f \cdot g}$.

We also require several coherences reminiscent of the laws of pseudofunctors, and for a precise description of those, we refer the reader to the formalization.

▶ **Problem 2.12.** *Given a displayed bicategory* D *on* B *and a section* s *on* D*, to construct a pseudofunctor* $\mathsf{Section}(s) : \mathsf{B} \to \int D$.

▶ **Construction 2.13** (for Problem 2.12; `section_to_psfunctor`)**.** We define the pseudofunctor $\mathsf{Section}(s)$ as follows: it sends objects x to the pair (x, s_x), 1-cells f to (f, s_f), and 2-cells $\tau : f \Rightarrow g$ to (τ, s_τ). ⌟

3 The Bicategory of Monads

Two concepts play a key role in the formal theory of monads: the bicategory of monads $\mathsf{Mnd}(\mathsf{B})$ internal to B and Eilenberg-Moore objects. In this section, we study the first concept, and we use displayed bicategories to construct the bicategory $\mathsf{Mnd}(\mathsf{B})$ given a bicategory B. We also show that $\mathsf{Mnd}(\mathsf{B})$ must be univalent if B is.

The main idea behind the construction is to split up monads in several independent parts. We first define $\mathsf{dEndo}(\mathsf{B})$ whose displayed objects over x are 1-cells $e : x \to x$. After that, we define $\mathsf{dUnit}(\mathsf{B})$ and $\mathsf{dMult}(\mathsf{B})$ whose displayed objects are the unit and multiplication of the monad respectively. We finally take a full subcategory for the monad laws.

▶ **Definition 3.1** (`disp_end`)**.** Let B be a bicategory. Define a displayed bicategory $\mathsf{dEndo}(\mathsf{B})$ over B as follows:

- The displayed objects over x are 1-cells $e_x : x \to x$;
- The displayed 1-cells over $f : x \to y$ from $e_x : x \to x$ to $e_y : y \to y$ are 2-cells τ_f

$$\begin{tikzcd}
x \arrow[r, "e_x"] \arrow[d, "f"'] & x \arrow[d, "f"] \\
y \arrow[r, "e_y"'] \arrow[ur, phantom, "\Uparrow \tau_f"] & y
\end{tikzcd}$$

- The displayed 2-cells over $\tau : f \Rightarrow g$ from $\theta_f : f \cdot e_y \Rightarrow e_x \cdot f$ to $\theta_g : g \cdot e_y \Rightarrow e_x \cdot g$ are proofs that the following diagram commutes

$$\begin{tikzcd}
f \cdot e_y \arrow[r, Rightarrow, "\theta_f"] \arrow[d, Rightarrow, "\tau \rhd e_y"'] & e_x \cdot f \arrow[d, Rightarrow, "e_x \lhd \tau"] \\
g \cdot e_y \arrow[r, Rightarrow, "\theta_g"'] & e_x \cdot g
\end{tikzcd}$$

We define $\mathsf{Endo}(B)$ by $\int \mathsf{dEndo}(\mathsf{B})$.

Next we define two displayed bicategories over $\mathsf{Endo}(\mathsf{B})$. For both, we use Definition 2.7.

▶ **Definition 3.2** (`disp_add_unit`)**.** Given a bicategory B, we define the displayed bicategory $\mathsf{dUnit}(\mathsf{B})$ over $\mathsf{Endo}(\mathsf{B})$ as follows

- The displayed objects over (x, e_x) are 2-cells $\eta_x : \mathrm{id}_x \Rightarrow e_x$;
- The displayed 1-cells over (f, θ_f) where $f : x \to y$ and $\theta_f : f \cdot e_y \Rightarrow e_x \cdot f$ from $\eta_x : \mathrm{id}_x \Rightarrow e_x$ to $\eta_y : \mathrm{id}_y \Rightarrow e_y$ are proofs that the following diagram commutes

$$\begin{tikzcd}
f \arrow[rr, Rightarrow, "\lambda^{-1}"] \arrow[d, Rightarrow, "\rho^{-1}"'] & & \mathrm{id}_x \cdot f \arrow[d, Rightarrow, "\eta_x \rhd f"] \\
f \cdot \mathrm{id}_y \arrow[r, Rightarrow, "f \lhd \eta_y"'] & f \cdot e_y \arrow[r, Rightarrow, "\theta_f"'] & e_x \cdot f
\end{tikzcd}$$

▶ **Definition 3.3** (`disp_add_mu`)**.** Given a bicategory B, we define the displayed bicategory $\mathsf{dMult}(\mathsf{B})$ over $\mathsf{Endo}(\mathsf{B})$ as follows

- The displayed objects over (x, e_x) are 2-cells $\mu_x : e_x \cdot e_x \Rightarrow e_x$;
- The displayed 1-cells over (f, θ_f) where $f : x \to y$ and $\theta_f : f \cdot e_y \Rightarrow e_x \cdot f$ from $\mu_x : e_x \cdot e_x \Rightarrow e_x$ to $\mu_y : e_y \cdot e_y \Rightarrow e_y$ are proofs that the following diagram commutes

$$\begin{tikzcd}
f \cdot (e_y \cdot e_y) \arrow[rr, Rightarrow, "f \lhd \mu_y"] \arrow[d, Rightarrow, "\alpha"'] & & f \cdot e_y \arrow[rr, Rightarrow, "\theta_f"] & & e_x \cdot f \\
(f \cdot e_y) \cdot e_y \arrow[r, Rightarrow, "\theta_f \rhd e_y"'] & (e_x \cdot f) \cdot e_y \arrow[r, Rightarrow, "\alpha^{-1}"'] & e_x \cdot (f \cdot e_y) \arrow[r, Rightarrow, "e_x \lhd \theta_f"'] & e_x \cdot (e_x \cdot f) \arrow[r, Rightarrow, "\alpha"'] & (e_x \cdot e_x) \cdot f \arrow[u, Rightarrow, "\mu_x \rhd f"']
\end{tikzcd}$$

Next we define $\mathsf{dMndData}(\mathsf{B})$ to be $\sum(\mathsf{dUnit}(\mathsf{B}) \times \mathsf{dMult}(\mathsf{B}))$ and we denote its total bicategory by $\mathsf{MndData}(\mathsf{B})$. To obtain $\mathsf{Mnd}(\mathsf{B})$, we take a full subbicategory.

▶ **Definition 3.4** (disp_mnd)**.** For a bicategory B, we define the predicate isMnd over $\mathsf{MndData}(\mathsf{B})$: given $(x,(e_x,(\eta_x,\mu_x)))$, the following diagrams commute

$$\begin{tikzcd}
e_x \arrow[r, Rightarrow, "\rho^{-1}"] \arrow[drr, Rightarrow, "\mathrm{id}_x"'] & e_x \cdot \mathrm{id}_x \arrow[r, Rightarrow, "e_x \lhd \eta_x"] & e_x \cdot e_x \arrow[d, Rightarrow, "\mu_x"] & \mathrm{id}_x \cdot e_x \arrow[l, Rightarrow, "\eta_x \rhd e_x"'] & e_x \arrow[l, Rightarrow, "\lambda^{-1}"'] \arrow[dll, Rightarrow, "\mathrm{id}_{e_x}"] \\
 & & e_x & &
\end{tikzcd}$$

$$\begin{tikzcd}
e_x \cdot (e_x \cdot e_x) \arrow[rr, Rightarrow, "e_x \lhd \mu_x"] \arrow[d, Rightarrow, "\alpha"'] & & e_x \cdot e_x \arrow[d, Rightarrow, "\mu_x"] \\
(e_x \cdot e_x) \cdot e_x \arrow[r, Rightarrow, "\mu_x \rhd e_x"'] & e_x \cdot e_x \arrow[r, Rightarrow, "\mu_x"'] & e_x
\end{tikzcd}$$

We define $\mathsf{dMnd}(\mathsf{B})$ to be $\sum(\mathsf{dFullSub}(\mathsf{isMnd}))$, and we denote its total bicategory by $\mathsf{Mnd}(\mathsf{B})$.

Note that the predicate isMnd contains all monads laws. Alternatively, we could have defined a displayed bicategory for each monad law. We refrained from doing so, because that would not further simplify the proof that $\mathsf{Mnd}(\mathsf{B})$ is univalent.

Before we continue, let us discuss the objects, 1-cells, and 2-cells of $\mathsf{Mnd}(\mathsf{B})$, and fix the relevant notation for the remainder of this paper. This also explains what the displayed objects, 1-cells, and 2-cells in $\mathsf{dMnd}(\mathsf{B})$ are. The data of a monad m in B consists of

- an object $\mathsf{ob}_m : \mathsf{B}$;
- a 1-cell $\mathsf{mor}_m : \mathsf{ob}_m \to \mathsf{ob}_m$;
- a 2-cell $\eta_m : \mathrm{id}_{\mathsf{ob}_m} \Rightarrow m$;
- a 2-cell $\mu_m : m \cdot m \Rightarrow m$.

If no confusion arises, we write m instead of mor_m.

The data of a monad morphism $f : m_1 \to m_2$ between monads m_1 and m_2 consists of a 1-cell $\mathsf{mor}_f : \mathsf{ob}_{m_1} \to \mathsf{ob}_{m_2}$ and a 2-cell $\mathsf{cell}_f : f \cdot m_2 \Rightarrow m_1 \cdot f$. Lastly, the data of monad 2-cell $\gamma : f_1 \Rightarrow f_2$ between monad morphisms is a 2-cell $\mathsf{cell}_\gamma : f_1 \Rightarrow f_2$. We write f instead of mor_f and γ instead of cell_γ if no confusion arises.

Next we look at the univalence of $\mathsf{Mnd}(\mathsf{B})$, and to prove it, we use displayed univalence [1, Definition 7.3]. This way, it suffices to prove the displayed univalence of $\mathsf{dEndo}(\mathsf{B})$, $\mathsf{dUnit}(\mathsf{B})$, and $\mathsf{dMult}(\mathsf{B})$. We also need B to be univalent.

▶ **Proposition 3.5** (is_univalent_2_mnd)**.** *If* B *is univalent, then so is* $\mathsf{Mnd}(\mathsf{B})$*.*

We also characterize invertible 2-cells and adjoint equivalences in $\mathsf{Mnd}(\mathsf{B})$.

▶ **Proposition 3.6** (is_invertible_mnd_2cell)**.** *A 2-cell γ between monad morphisms is invertible, if the underlying 2-cell* $\mathsf{cell}_\gamma : f_1 \Rightarrow f_2$ *is invertible.*

▶ **Proposition 3.7** (to_equivalence_mnd)**.** *Let* B *be a univalent bicategory and let $f : m_1 \to m_2$ be a 1-cell in* $\mathsf{Mnd}(B)$*. If* mor_f *is an adjoint equivalence and* cell_f *is an invertible 2-cell, then f is an adjoint equivalence.*

Note that in Proposition 3.7, we assume that B is univalent. Because of univalence, it suffices to prove this proposition assuming that mor_f is the identity 1-cell, which simplifies the involved coherences. The same idea can be used to show that pointwise pseudonatural adjoint equivalences are adjoint equivalences in the bicategory of pseudofunctors.

4 Examples of Monads

Next we look at examples of monads, and we start by characterizing monads internal to several bicategories. Let us start by observing that monads in the bicategory $\mathsf{UnivCat}$ of categories correspond to monads as how they usually are defined in category theory. However, since this notion of monad is defined in *every* bicategory, we can also look at other bicategories, such as $\mathsf{SymMonUnivCat}$,and $\mathsf{UnivCat}_{\mathsf{Terminal}}$.

In a wide variety of applications, one is interested in monads in a bicategory of categories with some extra structure. For example, symmetric monoidal monads are monads internal to the bicategory of symmetric monoidal categories. Strong monads are monads in the bicategory of left actegories. The bicategories in these two examples can be constructed as a total bicategory of some displayed bicategory over $\mathsf{UnivCat}$. To characterize monads in total bicategories, we define *displayed monads.*

▶ **Definition 4.1** (`disp_mnd`)**.** Let B be a bicategory and let D be a displayed bicategory over B and suppose that D is a local preorder and locally groupoidal. A **displayed monad** $\overline{m}$ over a monad m in B consists of

- a displayed object $\overline{\mathsf{ob}_m} : \mathsf{D}_{\mathsf{ob}_m}$;
- a displayed 1-cell $\overline{m} : \overline{\mathsf{ob}_m} \xrightarrow{m} \overline{\mathsf{ob}_m}$;
- a displayed 1-cell $\overline{\eta_m} : \overline{\mathrm{id}_{\mathsf{ob}_m}} \overset{\eta_m}{\Longrightarrow} \overline{m}$;
- a displayed 1-cell $\overline{\mu_m} : \overline{m} \cdot \overline{m} \overset{\mu_m}{\Longrightarrow} \overline{m}$.

Note that we do not require any coherences in Definition 4.1, because the involved displayed bicategory is assumed to be a local preorder.

▶ **Problem 4.2.** *Given a monad m and a displayed monad $\overline{m}$ over m, to construct a monad $\int m$ in $\int D$.*

▶ **Construction 4.3** (for Problem 4.2; `make_mnd_total_bicat`)**.** We construct $\int m$ as follows

- We define the object $\mathsf{ob}_{\int m}$ to be $(\mathsf{ob}_m, \overline{\mathsf{ob}_m})$;
- We define the 1-cell $\int m$ to be $(m, \overline{m})$;
- We define the unit $\eta_{\int m}$ to be $(\eta_m, \overline{\eta_m})$;
- We define multiplication $\mu_{\int m}$ to be $(\mu_m, \overline{\mu_m})$. ⌟

▶ **Example 4.4** (`make_mnd_univ_cat_with_terminal_obj`)**.** Every displayed monad in $\mathsf{dUnivCat}_{\mathsf{Terminal}}$ over m consists of a terminal object in ob_m and a proof that m preserves terminal objects.

Analogously, we can characterize monads in $\mathsf{SymMonUnivCat}$. Next we look at monads in B^{op} and B^{co}.

▶ **Example 4.5.** We characterize monads in B^{op} and B^{co}.

- (`MonadsInOp1Bicat.v`) Monads in B^{op} are the same as monads in B. However, 1-cells in $\mathsf{Mnd}(\mathsf{B}^{\mathsf{op}})$ are different from 1-cells in $\mathsf{Mnd}(\mathsf{B})$. If we have a 1-cell $f : m_1 \to m_2$ in $\mathsf{Mnd}(\mathsf{B}^{\mathsf{op}})$, then the cell cell_f gives rise to a 2-cell $m_2 \cdot f \Rightarrow f \cdot m_1$. Hence, 1-cells in $\mathsf{Mnd}(\mathsf{B}^{\mathsf{op}})^{\mathsf{op}}$ are the same as oplax monad morphisms in B.
- (`MonadsInOp2Bicat.v`) Suppose, we have $m : \mathsf{Mnd}(\mathsf{B}^{\mathsf{co}})$. Then $\mathsf{ob}_m : \mathsf{B}$ and we also have a 1-cell $m : \mathsf{ob}_m \to \mathsf{ob}_m$ in B. However, since the direction of the 2-cells are reversed in B^{co}, the 2-cells η_m and μ_m give rise to a 2-cell $m \Rightarrow \mathrm{id}_{\mathsf{ob}_m}$ and $m \Rightarrow m \cdot m$ respectively. As such, monads in B^{co} are the same as comonads in B.

▶ **Example 4.6** (`mnd_mnd_to_distr_law`)**.** Objects in $m : \mathsf{Mnd}(\mathsf{Mnd}(\mathsf{B}))$ are distributive laws between monads. To see why, observe that the object ob_m is a monad in B. In addition, we can construct another monad m' in B as follows:

- The object $\mathsf{ob}_{m'}$ is $\mathsf{ob}_{\mathsf{ob}_m}$;
- The endomorphism is m, which is a 1-cell from $\mathsf{ob}_{\mathsf{ob}_m}$ to $\mathsf{ob}_{\mathsf{ob}_m}$;
- The unit and multiplication are the underlying 2-cells of η_m and μ_m respectively.

The 2-cell cell_m is the 2-cell of the distributive law, and the laws are the proofs that η_m and μ_m are 2-cells in $\mathsf{Mnd}(\mathsf{B})$.

Analogously, we can show that objects in $\mathsf{Mnd}(\mathsf{Mnd}(\mathsf{B}^{\mathsf{co}})^{\mathsf{co}})$ are *mixed distributive laws* in B [12, 44]. We can also look at *iterated distributive laws*, which are monads in $\mathsf{Mnd}^n(\mathsf{B})$ [14].

Next we give two general constructions of monads. First of all, we consider the identity monad: on every object $x : \mathsf{B}$, we construct a monad $\mathsf{idMnd}(x)$.

▶ **Problem 4.7.** *Given a bicategory* B*, to construct a section on* $\mathsf{dMnd}(\mathsf{B})$*.*

▶ **Construction 4.8** (for Problem 4.7; `mnd_section_disp_bicat`)**.** To construct the desired section, we define the identity monad $\mathsf{idMnd}(x)$ for every x.

- The object is x;
- The 1-cell is $\mathrm{id}_x : x \to x$;
- The unit is $\mathrm{id}_{\mathrm{id}_x} : \mathrm{id}_x \Rightarrow \mathrm{id}_x$;
- The multiplication is $\lambda : \mathrm{id}_x \cdot \mathrm{id}_x \Rightarrow \mathrm{id}_x$. ⌟

In the remainder, we only use the pseudofunctor arising from Construction 4.8 and Construction 2.13, and this pseudofunctor is denoted as $\mathsf{idMnd} : \mathsf{B} \to \mathsf{Mnd}(\mathsf{B})$. Second, we notice that every monad $m : \mathsf{Mnd}(\mathsf{B})$ gives rise to a monad of categories. This example is used in Section 5.

▶ **Problem 4.9.** *Given a monad* m *in a bicategory* B *and an object* x*, to construct a monad* $\mathsf{HomMnd}_x(m)$ *on* $\underline{\mathsf{B}(x, \mathsf{ob}_m)}$*.*

▶ **Construction 4.10** (for Problem 4.9; `mnd_to_cat_Monad`)**.** The monad $\mathsf{HomMnd}_x(m)$ is defined as follows:

- The endofunctor is $(- \cdot m)_x : \underline{\mathsf{B}(x, \mathsf{ob}_m)} \to \underline{\mathsf{B}(x, \mathsf{ob}_m)}$.
- For every $f : x \to \mathsf{ob}_m$, the unit is defined to be

$$f \xRightarrow{\rho^{-1}} f \cdot \mathrm{id}_{\mathsf{ob}_m} \xRightarrow{f \lhd \eta_m} f \cdot m$$

- For every $f : x \to \mathsf{ob}_m$, the multiplication is defined to be

$$(f \cdot m) \cdot m \xRightarrow{\alpha^{-1}} f \cdot (m \cdot m) \xRightarrow{f \lhd \mu_m} f \cdot m$$

⌟

Next we show that pseudofunctors preserve monads.

▶ **Problem 4.11.** *Given bicategories* B_1 *and* B_2*, a pseudofunctor* $F : \mathsf{B}_1 \to \mathsf{B}_2$*, and a monad* $m : \mathsf{Mnd}(\mathsf{B}_1)$*, to construct a monad* $F(m) : \mathsf{Mnd}(\mathsf{B}_2)$*.*

▶ **Construction 4.12** (for Problem 4.11; `psfunctor_on_mnd`)**.** The object of $F(m)$ is $F(\mathsf{ob}_m)$ while the 1-cell is $F(m) : F(\mathsf{ob}_m) \to F(\mathsf{ob}_m)$. The unit and multiplication are constructed using to the following pasting diagrams respectively

$$\begin{tikzcd}
F(\mathsf{ob}_m) \arrow[rr, "F(\mathrm{id}_{\mathsf{ob}_m})" description] \arrow[rr, bend left=60, "\mathrm{id}_{F(\mathsf{ob}_m)}"] \arrow[rr, bend right=60, "F(m)"'] \arrow[rr, phantom, bend left=30, "\Downarrow F_i(\mathsf{ob}_m)"] \arrow[rr, phantom, bend right=30, "\Downarrow F(\eta_m)"] & & F(\mathsf{ob}_m)
\end{tikzcd}
\qquad
\begin{tikzcd}
F(\mathsf{ob}_m) \arrow[rr, "F(m \cdot m)" description] \arrow[rr, bend left=60, "F(m) \cdot F(m)"] \arrow[rr, bend right=60, "F(m)"'] \arrow[rr, phantom, bend left=30, "\Downarrow F_c(m,m)"] \arrow[rr, phantom, bend right=30, "\Downarrow F(\mu_m)"] & & F(\mathsf{ob}_m)
\end{tikzcd}$$

For a proof of the monad laws, we refer the reader to the formalization. ⌟

In fact, if we have a pseudofunctor $F : \mathsf{B}_1 \to \mathsf{B}_2$, then we obtain a pseudofunctor $\mathsf{Mnd}(F) : \mathsf{Mnd}(\mathsf{B}_1) \to \mathsf{Mnd}(\mathsf{B}_2)$. Next we show that monads can be composed if we have a distributive law between them.

▶ **Example 4.13** (`compose_mnd`)**.** Suppose that we have a distributive law τ between monads m_1 and m_2 (Example 4.6). Then we define a monad $m_1 \cdot m_2$ as follows:

- The object is ob_{m_1} (which is definitionally equal to ob_{m_2});
- The 1-cell is $m_1 \cdot m_2$;
- The unit is constructed as the following composition of 2-cells

$$\begin{tikzcd}
\mathrm{id}_{\mathsf{ob}_{m_1}} \arrow[r, Rightarrow, "\lambda^{-1}"] & \mathrm{id}_{\mathsf{ob}_{m_1}} \cdot \mathrm{id}_{\mathsf{ob}_{m_1}} \arrow[r, Rightarrow, "\eta_{m_1} \triangleright \mathrm{id}_{\mathsf{ob}_{m_1}}"] & m_1 \cdot \mathrm{id}_{\mathsf{ob}_{m_1}} \arrow[r, Rightarrow, "m_1 \triangleleft \eta_{m_2}"] & m_1 \cdot m_2
\end{tikzcd}$$

- The multiplication is the following composition of 2-cells

$$\begin{tikzcd}
(m_1 \cdot m_2) \cdot (m_1 \cdot m_2) \arrow[r, Rightarrow, "\alpha^{-1}"] & m_1 \cdot (m_2 \cdot (m_1 \cdot m_2)) \arrow[r, Rightarrow, "m_1 \triangleleft \alpha"] & m_1 \cdot ((m_2 \cdot m_1) \cdot m_2) \arrow[dll, Rightarrow, "m_1 \triangleleft (\tau \triangleright m_2)" description] \\
m_1 \cdot ((m_1 \cdot m_2) \cdot m_2) \arrow[r, Rightarrow, "m_1 \triangleleft \alpha^{-1}" description] & m_1 \cdot (m_1 \cdot (m_2 \cdot m_2)) \arrow[r, Rightarrow, "\alpha" description] & (m_1 \cdot m_1) \cdot (m_2 \cdot m_2) \arrow[dll, Rightarrow, "\mu_{m_1} \triangleright (m_2 \cdot m_2)" description] \\
m_1 \cdot (m_2 \cdot m_2) \arrow[rr, Rightarrow, "m_1 \triangleleft \mu_{m_2}"'] & & m_1 \cdot m_2
\end{tikzcd}$$

5 Eilenberg-Moore Objects

The second important concept in the formal theory of monads is the notion of *Eilenberg-Moore objects*. An important property of monads in category theory is that every monad gives rise to an adjunction. One can do this in two ways: either via Eilenberg-Moore categories or via Kleisli categories. In this section, we study Eilenberg-Moore objects, which formulate Eilenberg-Moore categories in bicategorical terms.

Note that the terminology in this section is slightly differently compared to what was used by Street [47]. Whereas Street would say that a bicategory admits the construction of algebras, we follow [24, 28, 43] and we say that a bicategory has Eilenberg-Moore objects. Our notions are also formulated slightly differently, because we use *Eilenberg-Moore cones*.

▶ **Definition 5.1** (em_cone)**.** Let B be a bicategory and let m be a monad in B. An **Eilenberg-Moore cone** for m consists of an object e together with a 1-cell $\mathsf{idMnd}(e) \to m$.

More concretely, an Eilenberg-Moore cone e for a monad m consists of

- An object ob_e;
- A 1-cell $\mathsf{mor}_e : \mathsf{ob}_e \to \mathsf{ob}_m$;
- A 2-cell $\mathsf{cell}_e : \mathsf{mor}_e \cdot m \Rightarrow \mathrm{id}_x \cdot \mathsf{mor}_e$

such that the following diagrams commutes

$$\begin{tikzcd}
\mathsf{mor}_e \arrow[r, Rightarrow, "\rho^{-1}"] \arrow[drr, Rightarrow, "\lambda^{-1}"'] & \mathsf{mor}_e \cdot \mathrm{id}_{\mathsf{ob}_m} \arrow[r, Rightarrow, "\mathsf{mor}_e \lhd \eta_m"] & \mathsf{mor}_e \cdot m \arrow[d, Rightarrow, "\mathsf{cell}_e"] \\
& & \mathrm{id}_x \cdot \mathsf{mor}_e
\end{tikzcd}$$

$$\begin{tikzcd}
\mathsf{mor}_e \cdot (m \cdot m) \arrow[rrr, Rightarrow, "\mathsf{mor}_e \lhd \mu_m"] \arrow[d, Rightarrow, "\alpha"'] & & & \mathsf{mor}_e \cdot m \arrow[d, Rightarrow, "\mathsf{cell}_e"] \\
(\mathsf{mor}_e \cdot m) \cdot m \arrow[r, Rightarrow, "\mathsf{cell}_e \rhd m"'] & (\mathrm{id}_x \cdot \mathsf{mor}_e) \cdot m \arrow[r, Rightarrow, "\lambda \rhd m"'] & \mathsf{mor}_e \cdot m \arrow[r, Rightarrow, "\mathsf{cell}_e"'] & \mathrm{id}_x \cdot \mathsf{mor}_e
\end{tikzcd}$$

If no confusion arises, we write e instead of ob_e.

Next we look at the universal property of Eilenberg-Moore objects. Since Eilenberg-Moore objects are examples of limits in bicategories [43], there are multiple methods to express their universal property. A first possibility, which is used by Street, is to use biadjunctions [47], and a second option is to write out explicit mapping properties. Alternatively, one could express the universal property as an adjoint equivalence on the hom-categories. We use the last option. To write out the desired definition precisely, we first define a functor with domain $\underline{\mathsf{B}(x, e)}$ where e is an Eilenberg-Moore cone and x is any object.

▶ **Problem 5.2.** *Given an Eilenberg-Moore cone e for m and an object x, to construct a functor* $\mathsf{EMFunctor}_{x,e} : \underline{\mathsf{B}(x, e)} \to \underline{\mathsf{Mnd}(\mathsf{B})(\mathsf{idMnd}(x), m)}$.

▶ **Construction 5.3** (for Problem 5.2; em_hom_functor)**.** Suppose that we have a 1-cell $f : x \to e$. We construct the monad morphism $\mathsf{EMFunctor}_{x,e}(f)$ as follows:

- The underlying morphism is $f \cdot \mathsf{mor}_e$.
- For the 2-cell $(f \cdot \mathsf{mor}_e) \cdot m \Rightarrow \mathrm{id}_x \cdot (f \cdot \mathsf{mor}_e)$, we take

$$(f \cdot \mathsf{mor}_e) \cdot m \xrightarrow{\alpha^{-1}} f \cdot \mathsf{mor}_e \cdot m \xrightarrow{f \lhd \mathsf{cell}_e} f \cdot \mathrm{id}_e \cdot \mathsf{mor}_e \xrightarrow{f \lhd \lambda} f \cdot \mathsf{mor}_e \xrightarrow{\lambda^{-1}} \mathrm{id}_x \cdot (f \cdot \mathsf{mor}_e).$$

Given a 2-cell $\tau : f \Rightarrow g$, the underlying cell of $\mathsf{EMFunctor}_{x,e}(\tau) : \mathsf{EMFunctor}_{x,e}(f) \Rightarrow \mathsf{EMFunctor}_{x,e}(g)$ is $\tau \rhd \mathsf{mor}_e$. ⌟

▶ **Definition 5.4** (bicat_has_em)**.** Let B be a bicategory and let m be a monad in B. An Eilenberg-Moore cone e for m is called **universal** if for every object x the functor $\mathsf{EMFunctor}_{x,e}$ is an adjoint equivalence of categories. We say that a bicategory has **Eilenberg-Moore objects** if for every monad m there is a universal Eilenberg-Moore cone.

▶ **Proposition 5.5** (isaprop_is_universal_em_cone)**.** *Let* B *be a locally univalent bicategory and let e be an Eilenberg-Moore cone for a monad m in* B*. The type that e is universal, is a proposition.*

Another way to formulate the universality of an Eilenberg-Moore cone, is by using Eilenberg-Moore categories. If we have a monad m on a category C, we write $\mathsf{EM}(m)$ for the Eilenberg-Moore category of m. Recall that the objects of $\mathsf{EM}(m)$ are pairs (x, f) of an object $x : \mathsf{C}$ and a morphism $f : m(x) \to x$ such that $\eta_m(x) \cdot f = \mathrm{id}_x$ and $\mu_m(x) \cdot f = m(f) \cdot f$. Morphisms from (x, f) to (y, g) are morphisms $h : x \to y$ such that $f \cdot h = m(h) \cdot g$.

▶ **Problem 5.6.** *Given an Eilenberg-Moore cone e for a monad m and an object $x : \mathsf{B}$, to construct a functor $\mathsf{EMFunctor}'_{x,e} : \underline{\mathsf{B}(x, e)} \to \mathsf{EM}(\mathsf{HomMnd}_x(m))$.*

▶ **Construction 5.7** (for Problem 5.6; `is_em_universal_em_cone_functor`)**.** First, we say how $\mathsf{EMFunctor}'_{x,e}$ acts on objects. Suppose that we have $f : x \to e$. To define an object of $\mathsf{EM}(\mathsf{HomMnd}_x(m))$, we first need to give a 1-cell $h : x \to \mathsf{ob}_x$. We define h as $x \xrightarrow{f} e \xrightarrow{\mathsf{mor}_e} \mathsf{ob}_m$ We also need to define a 2-cell $\tau : h \cdot m \Rightarrow h$ for which we take

$$(f \cdot \mathsf{mor}_e) \cdot m \xRightarrow{\alpha} f \cdot (\mathsf{mor}_e \cdot m) \xRightarrow{f \triangleleft \mathsf{cell}_e} f \cdot (\mathrm{id}_x \cdot \mathsf{mor}_e) \xRightarrow{f \triangleleft \lambda} f \cdot \mathsf{mor}_e$$

Next we define the action on morphisms. Suppose that we have $f, g : x \to e$ and a 2-cell $\theta : f \Rightarrow g$. We need to construct a 2-cell $f \cdot \mathsf{mor}_e \Rightarrow g \cdot \mathsf{mor}_e$, for which we take $\theta \triangleright \mathsf{mor}_e$. ⌟

▶ **Proposition 5.8** (`is_universal_em_cone_weq_is_em_universal_em_cone`)**.** *Let e be an Eilenberg-Moore cone. Then e is universal if and only if for every x the functor $\mathsf{EMFunctor}'_{x,e}$ is an adjoint equivalence.*

Proof. We only give a brief sketch of the proof. The main idea is that one can construct an adjoint equivalence from $\mathsf{EM}(\mathsf{HomMnd}_x(m))$ to $\underline{\mathsf{Mnd}(\mathsf{B})(\mathsf{idMnd}(x), m)}$. With this equivalence available, the theorem follows from the 2-out-of-3 property. ◀

Alternatively, universality can be formulated using mapping properties. These properties are deduced from the fact that every adjoint equivalence is split essentially surjective and fully faithful. More precisely, an Eilenberg-Moore cone e is universal if and only if

- for every Eilenberg-Moore cone q there is a 1-cell $\mathsf{EM}_{\mathsf{mor}}(q) : q \to e$ and an invertible 2-cell $\mathsf{EM}_{\mathsf{com}}(q) : \mathsf{idMnd}(\mathsf{EM}_{\mathsf{mor}}(q)) \cdot \mathsf{mor}_e \Rightarrow \mathsf{mor}_q$ in $\mathsf{Mnd}(\mathsf{B})$;
- for all 1-cells $g_1, g_2 : q \to e$ and 2-cells $\tau : \mathsf{idMnd}(g_1) \cdot \mathsf{mor}_e \Rightarrow \mathsf{idMnd}(g_2) \cdot \mathsf{mor}_e$ in $\mathsf{Mnd}(\mathsf{B})$, there is a unique 2-cell $\mathsf{EM}_{\mathsf{cell}}(\tau)$ such that $\mathsf{idMnd}(\mathsf{EM}_{\mathsf{cell}}(\tau)) \triangleright \mathsf{mor}_e = \tau$.

Let us consider some examples of Eilenberg-Moore objects.

▶ **Example 5.9** (`has_em_bicat_of_univ_cats`)**.** The bicategory $\mathsf{UnivCat}$ has Eilenberg-Moore objects. As suggested by the name, the Eilenberg-Moore object on a monad m is indeed given by the Eilenberg-Moore category $\mathsf{EM}(m)$. Note that if C is a univalent and m is a monad on C, then $\mathsf{EM}(m)$ is univalent as well.

▶ **Example 5.10** (`has_em_univ_cat_with_terminal_obj`)**.** The bicategory $\mathsf{UnivCat}_{\mathsf{Terminal}}$ has Eilenberg-Moore objects as well. If we have a monad $m : \mathsf{Mnd}(\mathsf{UnivCat}_{\mathsf{Terminal}})$, then ob_m has a terminal object and m preserves terminal objects. Under these conditions, it follows that the Eilenberg-Moore category of m has a terminal object. From this, we can conclude that $\mathsf{UnivCat}_{\mathsf{Terminal}}$ indeed has Eilenberg-Moore objects.

One can also show that $\mathsf{SymMonUnivCat}^{\mathsf{co}}$ has Eilenberg-Moore objects. These are given by Eilenberg-Moore categories of comonads.

6 Duality and Kleisli Objects

The goal of this section is to construct Eilenberg-Moore objects in $\mathsf{UnivCat}^{\mathsf{op}}$. To do so, we start by characterizing such objects via *Kleisli objects.*

▶ **Definition 6.1** (`kleisli_cocone`). Let B be a bicategory and let m be a monad in B. A **Kleisli cocone** k for m in B consists of an object $\mathsf{ob}_k : \mathsf{B}$, a 1-cell $\mathsf{mor}_k : \mathsf{ob}_m \to \mathsf{ob}_k$, and a 2-cell $\mathsf{cell}_k : m \cdot \mathsf{mor}_k \Rightarrow \mathsf{mor}_k$ such that the following diagrams commute

$$\begin{tikzcd}
\mathrm{id}_{\mathsf{ob}_m} \cdot \mathsf{mor}_k \arrow[r, Rightarrow, "\eta_m \triangleright \mathsf{mor}_k"] \arrow[dr, Rightarrow, "\lambda"'] & m \cdot \mathsf{mor}_k \arrow[d, Rightarrow, "\mathsf{cell}_k"] \\
& \mathsf{mor}_k
\end{tikzcd}$$

$$\begin{tikzcd}
(m \cdot m) \cdot \mathsf{mor}_k \arrow[r, Rightarrow, "\alpha^{-1}"] \arrow[d, Rightarrow, "\mu_m \triangleright \mathsf{mor}_k"'] & m \cdot (m \cdot \mathsf{mor}_k) \arrow[r, Rightarrow, "m \triangleleft \mathsf{cell}_k"] & m \cdot \mathsf{mor}_k \arrow[d, Rightarrow, "\mathsf{cell}_k"] \\
m \cdot \mathsf{mor}_k \arrow[rr, Rightarrow, "\mathsf{cell}_k"'] & & \mathsf{mor}_k
\end{tikzcd}$$

▶ **Definition 6.2** (`has_kleisli_ump`). A Kleisli cocone k is said to be **universal** if

- For every Kleisli cocone q there there is a 1-cell $\mathsf{Kl}_{\mathsf{mor}}(q) : \mathsf{ob}_k \to \mathsf{ob}_q$ and an invertible 2-cell $\mathsf{Kl}_{\mathsf{com}}(q) : \mathsf{mor}_k \cdot \mathsf{Kl}_{\mathsf{mor}}(q) \Rightarrow \mathsf{mor}_q$ such that the following diagram commutes

$$\begin{tikzcd}
m \cdot (\mathsf{mor}_k \cdot \mathsf{Kl}_{\mathsf{mor}}(q)) \arrow[rr, Rightarrow, "m \triangleleft \mathsf{Kl}_{\mathsf{com}}(q)"] \arrow[d, Rightarrow, "\alpha"'] & & m \cdot \mathsf{mor}_q \arrow[d, Rightarrow, "\mathsf{cell}_q"] \\
(m \cdot \mathsf{mor}_k) \cdot \mathsf{Kl}_{\mathsf{mor}}(q) \arrow[r, Rightarrow, "\mathsf{cell}_k \triangleright \mathsf{Kl}_{\mathsf{mor}}(q)"'] & \mathsf{mor}_k \cdot \mathsf{Kl}_{\mathsf{mor}}(q) \arrow[r, Rightarrow, "\mathsf{Kl}_{\mathsf{com}}(q)"'] & \mathsf{mor}_q
\end{tikzcd}$$

- Suppose that we have an object $x : \mathsf{B}$, two 1-cells $g_1, g_2 : \mathsf{ob}_k \to x$, and a 2-cell $\tau : \mathsf{mor}_k \cdot g_1 \Rightarrow \mathsf{mor}_k \cdot g_2$ such that the following diagram commutes

$$\begin{tikzcd}
m \cdot (\mathsf{mor}_k \cdot g_1) \arrow[r, Rightarrow, "\alpha"] \arrow[d, Rightarrow, "m \triangleleft \tau"'] & (m \cdot \mathsf{mor}_k) \cdot g_1 \arrow[r, Rightarrow, "\mathsf{cell}_k \triangleright g_1"] & \mathsf{mor}_k \cdot g_1 \arrow[d, Rightarrow, "\tau"] \\
m \cdot (\mathsf{mor}_k \cdot g_2) \arrow[r, Rightarrow, "\alpha"'] & (m \cdot \mathsf{mor}_k) \cdot g_2 \arrow[r, Rightarrow, "\mathsf{cell}_k \triangleright g_2"'] & \mathsf{mor}_k \cdot g_2
\end{tikzcd}$$

  Then there is a unique 2-cell $\mathsf{Kl}_{\mathsf{cell}}(\tau) : g_1 \Rightarrow g_2$ such that $\mathsf{cell}_k \triangleleft \mathsf{Kl}_{\mathsf{cell}}(\tau) = \tau$.

A **Kleisli object** is a universal Kleisli cocone. We say that a bicategory has Kleisli objects if there is a Kleisli object for every monad m.

▶ **Proposition 6.3** (`op1_has_em`). *If* B *has Kleisli objects, then* B^{op} *has Eilenberg-Moore objects.*

As such, to find Eilenberg-Moore objects in $\mathsf{UnivCat}^{\mathsf{op}}$, we need to find Kleisli objects in $\mathsf{UnivCat}$. However, before we look at those, we look at Kleisli objects in Cat. These are constructed via the usual definition of *Kleisli categories.*

▶ **Problem 6.4.** *To construct Kleisli objects in* Cat.

▶ **Construction 6.5** (for Problem 6.4; `bicat_of_cats_has_kleisli`). Recall that given a monad m on a category C, the Kleisli category $\mathcal{K}(m)$ is usually defined to be the category whose objects are $x : \mathsf{C}$ and whose morphisms from $x : \mathsf{C}$ to $y : \mathsf{C}$ are morphisms $x \to m(y)$.

Note that we have a functor $F : \mathsf{C} \to \mathcal{K}(m)$: it sends objects x to x and morphisms $f : x \to y$ to $x \xrightarrow{f} y \xrightarrow{\eta_m(y)} m(y)$. We also have a natural transformation $m \cdot F \Rightarrow F$, which is the identity on ever object x. As such, we have a Kleisli cocone. This cocone is universal, and for a proof we refer the reader to the formalization. ⌟

Note that even if C is required to be univalent, the Kleisli category $\mathcal{K}(m)$ is *not* necessarily univalent. As such, to obtain Kleisli objects in $\mathsf{UnivCat}$, we need to use an alternative definition for the Kleisli category [6]. First, we define a functor $\mathsf{FreeAlg}_m : \mathsf{C} \to \mathsf{EM}(m)$ which sends objects $x : \mathsf{C}$ to the algebra $\mu_m(x) : m(m(x)) \to m(x)$ and morphisms $f : x \to y$ to $m(f) : m(x) \to m(y)$. Note that this 1-cell can actually be defined in arbitrary bicategories (see Section 7). By taking the full image of this functor, we obtain the category $\mathsf{Kleisli}(m)$.

- Objects of $\mathsf{Kleisli}(m)$ are pairs $y : \mathsf{C}$ together with a proof of $\|\sum(x : \mathsf{C}), m(x) \cong y\|$.
- Morphisms from $y_1 : \mathsf{Kleisli}(m)$ to $y_2 : \mathsf{Kleisli}(m)$ are morphisms $y_1 \to y_2$ in C.

If C is univalent, then $\mathsf{EM}_{\mathsf{UnivCat}}(m)$ is univalent, and thus $\mathsf{Kleisli}(m)$ is so as well.

▶ **Problem 6.6.** *To construct a fully faithful essentially surjective functor* $\mathsf{incl}_m : \mathcal{K}(m) \to \mathsf{Kleisli}(m)$.

▶ **Construction 6.7** (for Problem 6.6; `functor_to_kleisli_cat`)**.** The functor incl_m sends every object $x : \mathcal{K}(m)$ to $\mathsf{FreeAlg}_m(x)$, which is indeed in the image of $\mathsf{FreeAlg}_m$. Morphisms $f : x \to m(y)$ are sent to $m(x) \xrightarrow{m(f)} m(m(y)) \xrightarrow{\mu_m(y)} m(y)$. A proof that this functor is both essentially surjective and fully faithful can be found in the formalization. ⌟

In univalent foundations, not every functor that is both fully faithful and essentially surjective is automatically an adjoint equivalence as well. This statement only holds if the domain is univalent. However, we can still use the functor incl_m to deduce the universal property of $\mathsf{Kleisli}(m)$. For that, we use Theorem 8.4 in [2].

▶ **Proposition 6.8** (`precomp_adjoint_equivalence`)**.** *Let $F : \mathsf{C}_1 \to \mathsf{C}_2$ be a fully faithful and essentially surjective functor, and suppose that C_3 is a univalent category. Then the functor $(F \cdot -)_{\mathsf{C}_3} : \mathsf{C}_3^{\mathsf{C}_2} \to \mathsf{C}_3^{\mathsf{C}_1}$, given by precomposition with F, is an adjoint equivalence.*

▶ **Problem 6.9.** *To construct Kleisli objects in* $\mathsf{UnivCat}$.

▶ **Construction 6.10** (for Problem 6.9; `bicat_of_univ_cats_has_kleisli`)**.** We only show how to construct the required 1-cells. Suppose that we have a Kleisli cocone q in $\mathsf{UnivCat}$. Note that q also is a Kleisli cocone in Cat, and as such, we get a functor $\mathsf{Kl}_{\mathsf{mor}}(q) : \mathcal{K}(m) \to \mathsf{ob}_q$. By Proposition 6.8, we now get the desired functor $\mathsf{Kleisli}(m) \to \mathsf{ob}_q$. ⌟

7 Monads and Adjunctions

The cornerstone of the theory of monads is the relation between monads and adjunctions. More specifically, every adjunction gives rise to a monad and vice versa. This was generalized by Street to 2-categories that have Eilenberg-Moore objects [47]. In this section, we prove these theorems, and to do so, we start by recalling adjunctions in bicategories.

▶ **Definition 7.1** (`adjunction`)**.** An **adjunction** $l \,\frac{\varepsilon}{\eta}\!\big|\, r$ in a bicategory B consists of

- objects x and y;
- 1-cells $l : x \to y$ and $r : y \to x$;
- 2-cells $\eta : \mathrm{id}_x \Rightarrow l \cdot r$ and $\varepsilon : r \cdot l \Rightarrow \mathrm{id}_y$

such that the following 2-cells are identities

$$l \xRightarrow{\lambda^{-1}} \mathrm{id}_x \cdot l \xRightarrow{\eta \triangleright l} (l \cdot r) \cdot l \xRightarrow{\alpha^{-1}} l \cdot (r \cdot l) \xRightarrow{l \triangleleft \varepsilon} l \cdot \mathrm{id}_y \xRightarrow{\rho} l$$

$$r \xRightarrow{\rho^{-1}} r \cdot \mathrm{id}_x \xRightarrow{r \triangleleft \eta} r \cdot (l \cdot r) \xRightarrow{\alpha} (r \cdot l) \cdot r \xRightarrow{\varepsilon \triangleright r} \mathrm{id}_y \cdot r \xRightarrow{\lambda} r$$

Our notation for adjunctions is taken from [15]. The two coherences in Definition 7.1 are called the *triangle equalities.* As expected, adjunctions internal to UnivCat correspond to adjunctions of categories [31]. This is because the unitors and associators in UnivCat are pointwise the identity, so the triangle equalities in Definition 7.1 reduce to the usual ones.

▶ **Example 7.2.** We characterize adjunctions in B^{op} and B^{co} as follows.

- (`op1_left_adjoint_to_right_adjoint`) Every adjunction $l \mathrel{\frac{\varepsilon}{\eta}\Big|} r$ in B gives rise to an adjunction $r \mathrel{\frac{\varepsilon}{\eta}\Big|} l$ in B^{op} and vice versa.
- (`op2_left_adjoint_to_right_adjoint`) Every adjunction $l \mathrel{\frac{\varepsilon}{\eta}\Big|} r$ in B gives rise to an adjunction $r \mathrel{\frac{\eta}{\varepsilon}\Big|} l$ in B^{co} and vice versa.

Example 7.2 can be strengthened by using the terminology of *left adjoints* and *right adjoints.* Given a 1-cell $f : x \to y$, the type $\mathsf{LeftAdj}_{\mathsf{B}}(f)$ says that we have r,η, and ε such that we have an adjunction $l \mathrel{\frac{\varepsilon}{\eta}\Big|} r$. The type $\mathsf{RightAdj}_{\mathsf{B}}(f)$ is defined analogously. Now we can reformulate Example 7.2 as follows: we have equivalences $\mathsf{LeftAdj}_{\mathsf{B}^{\mathsf{op}}}(f) \simeq \mathsf{RightAdj}_{\mathsf{B}}(f)$ and $\mathsf{LeftAdj}_{\mathsf{B}^{\mathsf{co}}}(f) \simeq \mathsf{RightAdj}_{\mathsf{B}}(f)$ of types.

Next we look at *displayed adjunctions*, which we use to obtain adjunctions in total bicategories [1]. This notion is used to characterize adjunctions in bicategories such as $\mathsf{UnivCat}_{\mathsf{Terminal}}$ and SymMonUnivCat.

▶ **Definition 7.3** (`disp_adjunction`)**.** Let B be a bicategory and let D be a displayed bicategory over B. A **displayed adjunction** over an adjunction $l \mathrel{\frac{\varepsilon}{\eta}\Big|} r$ where $l : x \to y$ consists of

- Objects $\overline{x} : D_x$ and $\overline{y} : D_y$;
- Displayed morphisms $\overline{l} : \overline{x} \xrightarrow{l} \overline{y}$ and $\overline{r} : \overline{y} \xrightarrow{r} \overline{x}$;
- Displayed 2-cells $\overline{\eta} : \overline{\mathrm{id}_x} \overset{\eta}{\Rightarrow} \overline{l} \cdot \overline{r}$ and $\overline{\varepsilon} : \overline{r} \cdot \overline{l} \overset{\varepsilon}{\Rightarrow} \mathrm{id}_x$.

We also require some coherences and those can be found in the formalization. We denote this data by $\overline{l} \mathrel{\frac{\overline{\varepsilon}}{\overline{\eta}}\Big|} \overline{r}$.

▶ **Problem 7.4.** *Given a displayed adjunction* $\overline{l} \mathrel{\frac{\overline{\varepsilon}}{\overline{\eta}}\Big|} \overline{r}$ *in a displayed bicategory* D *over* $l \mathrel{\frac{\varepsilon}{\eta}\Big|} r$*, to construct an adjunction* $\int \overline{l} \mathrel{\frac{\overline{\varepsilon}}{\overline{\eta}}\Big|} \overline{r}$ *in* $\int D$*.*

▶ **Construction 7.5** (for Problem 7.4; `left_adjoint_data_total_weq`)**.** The left adjoint of $\int \overline{l} \mathrel{\frac{\overline{\varepsilon}}{\overline{\eta}}\Big|} \overline{r}$ is $(l, \overline{l})$, the right adjoint is $(r, \overline{r})$, the unit is $(\eta, \overline{\eta})$, and the counit is $(\varepsilon, \overline{\varepsilon})$. ⌟

▶ **Example 7.6** (`disp_adj_weq_preserves_terminal`)**.** Adjunctions in $\mathsf{UnivCat}_{\mathsf{Terminal}}$ are given by an adjunction $l \mathrel{\frac{\varepsilon}{\eta}\Big|} r$ in UnivCat such that l preserves terminal objects. Note that r automatically preserves terminal objects, because r is a right adjoint.

Analogously, we characterize adjunctions in SymMonUnivCat. Now we have developed enough to state and prove the core theorems of the formal theory of monads [47]. These theorems relate adjunctions and monads, and we first prove that every adjunction gives rise to a monad.

▶ **Problem 7.7.** *Given an adjunction* $l \mathrel{\frac{\varepsilon}{\eta}\Big|} r$*, to construct a monad* $\mathsf{AdjToMnd}(l \mathrel{\frac{\varepsilon}{\eta}\Big|} r)$*.*

▶ **Construction 7.8** (for Problem 7.7; `mnd_from_adjunction`)**.** Let an adjunction $l \frac{\varepsilon}{\eta}\!\dashv r$ be given where $l : x \to y$. We define the monad $\mathsf{AdjToMnd}(l \frac{\varepsilon}{\eta}\!\dashv r)$ as follows.

- Its object is x;
- The endomorphism is $l \cdot r : x \to x$;
- The unit is $\eta : \mathrm{id}_x \to l \cdot r$;
- For the multiplication, we use the following composition of 2-cells

$$(l \cdot r) \cdot (l \cdot r) \xRightarrow{\alpha} l \cdot (r \cdot (l \cdot r)) \xRightarrow{l \triangleleft \alpha^{-1}} l \cdot ((r \cdot l) \cdot r) \xRightarrow{l \triangleleft (\varepsilon \triangleright r)} l \cdot (\mathrm{id}_y \cdot r) \xRightarrow{l \triangleleft \lambda} l \cdot r$$

The proofs of the necessary equalities can be found in the formalization. ⌟

Since by Example 4.5 and 7.2 adjunctions and monads in B^{co} correspond to adjunctions and comonads in B respectively, we get that every adjunction in B induces a comonad by Construction 7.8. Next we look at the converse: obtaining adjunctions from monads. For this, we need to work in a bicategory with Eilenberg-Moore objects. We show that every monad m gives rise to an adjunction and that the monad coming from this adjunction is equivalent to m.

▶ **Problem 7.9.** *Given a bicategory* B *with Eilenberg-Moore objects and a monad* m *in* B*, to construct an adjunction* $\mathsf{MndToAdj}(m)$ *and an adjoint equivalence* $\mathsf{MndEquiv}(m)$ *between* $\mathsf{AdjToMnd}(\mathsf{MndToAdj}(m))$ *and* m*.*

▶ **Construction 7.10** (for Problem 7.9; `mnd_to_adjunction`)**.** The right adjoint is the 1-cell $\mathsf{mor}_e : \mathsf{ob}_m \to \mathsf{EM}_{\mathsf{B}}(m)$. For the left adjoint, we need to define a 1-cell $\mathsf{FreeAlg}_m : \mathsf{EM}_{\mathsf{B}}(m) \to \mathsf{ob}_m$, and we use the universal property of Eilenberg-Moore objects for that. We construct a cone q as follows:

- The object is ob_m;
- The morphism is m;
- The 2-cell is $m \cdot m \xRightarrow{\mu_m} m \xRightarrow{\lambda^{-1}} \mathrm{id}_{\mathsf{ob}_m} \cdot m$.

We define $\mathsf{FreeAlg}_m$ as $\mathsf{EM}_{\mathsf{mor}}(q)$. The unit of the desired adjunction is defined as follows:

$$\mathrm{id}_{\mathsf{ob}_m} \xRightarrow{\eta_m} m \xRightarrow{\mathsf{EM}_{\mathsf{com}}(q)} \mathsf{FreeAlg}_m \cdot \mathsf{mor}_e$$

For the counit we use the universal property of Eilenberg-Moore objects. The details can be found in the formalization.

To construct $\mathsf{MndEquiv}(m)$, we use Proposition 3.7. Hence, it suffices to construct a monad morphism $G : \mathsf{AdjToMnd}(\mathsf{MndToAdj}(m)) \to m$ whose underlying 1-cell and 2-cell are an adjoint equivalence and invertible respectively. We define G as follows:

- The underlying 1-cell is $\mathrm{id}_{\mathsf{ob}_m} : \mathsf{ob}_m \to \mathsf{ob}_m$.
- For the underlying 2-cell, we take

$$\mathrm{id}_{\mathsf{ob}_m} \cdot m \xRightarrow{\lambda} m \xRightarrow{\mathsf{EM}_{\mathsf{com}}(q)} \mathsf{FreeAlg}_m \cdot \mathsf{mor}_e \xRightarrow{\rho^{-1}} (\mathsf{FreeAlg}_m \cdot \mathsf{mor}_e) \cdot \mathrm{id}_{\mathsf{ob}_m}$$ ⌟

Note that we can instantiate Construction 7.10 to several concrete instances.

- Since $\mathsf{UnivCat}$ has Eilenberg-Moore objects by Example 5.9, we get the usual construction of adjunctions from monads via Eilenberg-Moore categories.
- Since $\mathsf{UnivCat}^{\mathsf{op}}$ has Eilenberg-Moore objects by Construction 6.10, every monad gives rise to an adjunction via Kleisli categories.

One can also show that $\mathsf{SymMonUnivCat}^{\mathsf{co}}$ has Eilenberg-Moore objects, and thus every comonad of symmetric monoidal categories gives rise to an adjunction.

8 Monadic Adjunctions

By Construction 7.10 we have an equivalence $\mathsf{AdjToMnd}(\mathsf{MndToAdj}(m)) \simeq m$ for every monad m. However, such a statement does not hold for adjunctions. *Monadic adjunctions* are the adjunctions for which we do have such an equivalence.

▶ **Problem 8.1.** *Given an adjunction* $l \overset{\varepsilon}{\underset{\eta}{\dashv}} r$ *where* $l : x \to y$ *in a bicategory* B *with Eilenberg-Moore objects, to construct a 1-cell* $\mathsf{Comparison}(l \overset{\varepsilon}{\underset{\eta}{\dashv}} r) : y \to \mathsf{EM}_{\mathsf{B}}(\mathsf{AdjToMnd}(l \overset{\varepsilon}{\underset{\eta}{\dashv}} r))$*.*

▶ **Construction 8.2** (for Problem 8.1; `comparison_mor`)**.** We use the universal property of Eilenberg-Moore objects, and we construct a cone q as follows

- The object is y;
- The 1-cell is $r : y \to x$;
- The 2-cell is $r \cdot (l \cdot r) \xRightarrow{\alpha} (r \cdot l) \cdot r \xRightarrow{\varepsilon \triangleright r} \mathrm{id}_y \cdot r$.

Note that the object and 1-cell of $\mathsf{AdjToMnd}(l \overset{\varepsilon}{\underset{\eta}{\dashv}} r)$ are x and $l \cdot r$ respectively. Now we define $\mathsf{Comparison}(l \overset{\varepsilon}{\underset{\eta}{\dashv}} r)$ to be $\mathsf{EM}_{\mathsf{mor}}(q)$. $\lrcorner$

▶ **Definition 8.3** (`is_monadic`)**.** An adjunction $l \overset{\varepsilon}{\underset{\eta}{\dashv}} r$ in a bicategory B with Eilenberg-Moore objects is called **monadic** if the 1-cell $\mathsf{Comparison}(l \overset{\varepsilon}{\underset{\eta}{\dashv}} r)$ is an adjoint equivalence.

Next we look at a representable version of this definition. More specifically, we define monadic 1-cells using monadic functors in $\mathsf{UnivCat}$. To do so, we first show that every adjunction gives rise to an adjunction on the hom-categories.

▶ **Problem 8.4.** *Given* $l \overset{\varepsilon}{\underset{\eta}{\dashv}} r$ *where* $l : x \to y$ *and an object* w*, to construct an adjunction* $\mathsf{HomAdj}_w(l \overset{\varepsilon}{\underset{\eta}{\dashv}} r)$ *between* $\underline{\mathsf{B}(w,x)}$ *and* $\underline{\mathsf{B}(w,y)}$*.*

▶ **Construction 8.5** (for Problem 8.4; `left_adjoint_to_adjunction_cat`)**.** The left adjoint is $(- \cdot l)_w$, while the right adjoint is $(- \cdot r)_w$. For the unit, we need to construct natural 2-cells $f \Rightarrow (f \cdot l) \cdot r$, and for which we take

$$f \xRightarrow{\rho^{-1}} f \cdot \mathrm{id}_y \xRightarrow{f \triangleleft \eta} f \cdot (l \cdot r) \xRightarrow{\alpha} (f \cdot l) \cdot r$$

For the counit, we construct natural 2-cells $(f \cdot r) \cdot l$, which are defined as follows

$$(f \cdot r) \cdot l \xRightarrow{\alpha^{-1}} f \cdot (r \cdot l) \xRightarrow{f \triangleleft \varepsilon} f \cdot \mathrm{id}_y \xRightarrow{\rho} f$$

$\lrcorner$

▶ **Definition 8.6** (`is_monadic_repr`)**.** An adjunction $l \overset{\varepsilon}{\underset{\eta}{\dashv}} r$ in a locally univalent bicategory is called **representably monadic** if for every $w \in \mathsf{B}$ the adjunction $\mathsf{HomAdj}_w(l \overset{\varepsilon}{\underset{\eta}{\dashv}} r)$ is a monadic 1-cell in $\mathsf{UnivCat}$.

Note that we require the bicategory in Definition 8.6 to be locally univalent, so that each hom-category lies in $\mathsf{UnivCat}$. Now we show that these two notions of monadicity are equivalent. We first prove the following lemma.

▶ **Lemma 8.7** (`left_adjoint_equivalence_weq_left_adjoint_equivalence_repr`)**.** *A 1-cell* $f : x \to y$ *is an adjoint equivalence if and only if for all* w *the functor* $(- \cdot f)_w$ *is an adjoint equivalence of categories.*

Proof. Suppose, we have a a adjoint equivalence $f : x \simeq y$ and let $w : \mathsf{B}$. By Construction 8.5, we obtain an adjunction $\mathsf{HomAdj}_w(l \overset{\varepsilon}{\underset{\eta}{\dashv}} r)$ between $\underline{\mathsf{B}(w,x)}$ and $\underline{\mathsf{B}(w,y)}$ whose right adjoint is $(- \cdot f)_w$. Since the unit and counit of f are invertible, the unit and counit of $\mathsf{HomAdj}_w(l \overset{\varepsilon}{\underset{\eta}{\dashv}} r)$ are invertible as well, and thus we get the desired adjoint equivalence.

Next suppose that for every w the functor $(- \cdot f)_w$ is an adjoint equivalence. For every w, we denote its right adjoint by $R_w : \underline{\mathsf{B}(w,y)} \to \underline{\mathsf{B}(w,x)}$, its unit by $\eta_w : R_w \cdot (- \cdot f)_w \Rightarrow \mathrm{id}$, and its counit by $\varepsilon_w : \mathrm{id} \Rightarrow (- \cdot f)_w \cdot R_w$. Now we show that f is an adjoint equivalence

- The right adjoint is $R_y(\mathrm{id}_y) : \underline{\mathsf{B}(y,x)}$.
- The unit is $\eta_w(\mathrm{id}_y) : R_y(\mathrm{id}_y) \cdot f \Rightarrow \mathrm{id}_y$.
- For the counit, we use the following composition

$$\begin{tikzcd}
f \cdot R_y(\mathrm{id}_y) \arrow[r, Rightarrow, "\eta_x(f \cdot R_y(\mathrm{id}_y))"] & R_x((f \cdot R_y(\mathrm{id}_y)) \cdot f) \arrow[r, Rightarrow, "R(\alpha^{-1})"] & R_x(f \cdot (R_y(\mathrm{id}_y) \cdot f)) \arrow[dll, Rightarrow, "R(f \triangleleft \eta_y(\mathrm{id}_y))"] & & \\
R_x(f \cdot \mathrm{id}_y) \arrow[r, Rightarrow, "R(\rho)"'] & R_x(f) \arrow[r, Rightarrow, "R(\lambda^{-1})"'] & R_x(\mathrm{id}_x \cdot f) \arrow[r, Rightarrow, "\eta_x^{-1}(\mathrm{id}_x)"'] & \mathrm{id}_x
\end{tikzcd}$$

Since both the unit and counit are invertible, f is indeed an adjoint equivalence. ◀

▶ **Theorem 8.8** (`is_monadic_repr_weq_is_monadic`)**.** *An adjunction is monadic if and only if it is representably monadic.*

Proof. Suppose that we have an adjunction $l \overset{\varepsilon}{\underset{\eta}{\dashv}} r$ and that we have $w : \mathsf{B}$. First, we note that we have a monad on $\underline{\mathsf{B}(w,y)}$, namely $m := \mathsf{HomMnd}_w(\mathsf{HomAdj}_w(l \overset{\varepsilon}{\underset{\eta}{\dashv}} r))$. We denote the comparison cell $\mathsf{Comparison}(\mathsf{HomAdj}_w(l \overset{\varepsilon}{\underset{\eta}{\dashv}} r))$ by F.

We also have a functor $(- \cdot \mathsf{Comparison}(l \overset{\varepsilon}{\underset{\eta}{\dashv}} r))_w : \underline{\mathsf{B}(w,y)} \to \underline{\mathsf{B}(w, \mathsf{EM}_{\mathsf{B}}(m'))}$, which we denote by G. We write m' for the monad $\mathsf{HomMnd}_w(m)$, and recall that by Proposition 5.8 we have an adjoint equivalence from $\underline{\mathsf{B}(w, \mathsf{EM}_{\mathsf{B}}(m'))}$ to $\mathsf{EM}(\mathsf{HomMnd}_w(m))$. Denote this equivalence by H. There also is a functor $K : \mathsf{EM}(m') \to \mathsf{EM}(m)$ and a natural isomorphism $\tau : F \cdot (H \cdot K) \cong G$: their precise definition can be found in the formalization.

As such, we have the following diagram for every $w : \mathsf{B}$:

$$\begin{tikzcd}
\underline{\mathsf{B}(w,y)} \arrow[r, "G"] \arrow[d, "F"'] & \mathsf{EM}(m) \\
\underline{\mathsf{B}(w, \mathsf{EM}_{\mathsf{B}}(m'))} \arrow[r, "H"'] \arrow[ur, phantom, "\Uparrow \tau"] & \mathsf{EM}(m') \arrow[u, "K"']
\end{tikzcd}$$

Since both H and K are adjoint equivalences, we deduce that F is an adjoint equivalence if and only if G is. As such, if $l \overset{\varepsilon}{\underset{\eta}{\dashv}} r$ is representably monadic, then F is an adjoint equivalence and thus G is an adjoint equivalence. From Lemma 8.7 we get that $l \overset{\varepsilon}{\underset{\eta}{\dashv}} r$ is monadic. For the converse, we use the same argument: if $l \overset{\varepsilon}{\underset{\eta}{\dashv}} r$ is monadic, then G is an adjoint equivalence. Hence, F is an adjoint equivalence as well, so $l \overset{\varepsilon}{\underset{\eta}{\dashv}} r$ is representably monadic. ◀

9 Conclusion

We developed Street's formal theory of monads in this paper. We saw that it provides a good setting to study monads in univalent foundations, because it allows us to prove the core theorems in arbitrary bicategories instead of only for categories. For that reason, it helps us with concrete problems, such as constructing an adjunction from a monad using the univalent version of the Kleisli category. This is because one only needs to prove a universal property instead of reproducing the whole construction of the adjunction.

There are numerous ways to continue this line of research. One result that is missing, is Theorem 12 from Street's paper [47]. In addition, the work in this setting provides a framework in which one can study numerous applications, such as models of linear logic [11, 34], Moggi-style semantics [36], call-by-push-value [30], and the enriched effect calculus [16]. Formalizing these applications would be a worthwhile extension. Finally, one could study extensions of the formal theory to a wider class of monads, such as graded monads [17] or relative monads [8].

References

1 Benedikt Ahrens, Dan Frumin, Marco Maggesi, Niccolò Veltri, and Niels van der Weide. Bicategories in univalent foundations. *Mathematical Structures in Computer Science*, pages 1–38, 2022. doi:10.1017/S0960129522000032.

2 Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. Univalent categories and the Rezk completion. *Mathematical Structures in Computer Science*, 25:1010–1039, 2015. doi:10.1017/S0960129514000486.

3 Benedikt Ahrens and Peter LeFanu Lumsdaine. Displayed Categories. *Logical Methods in Computer Science*, 15(1), 2019. doi:10.23638/LMCS-15(1:20)2019.

4 Benedikt Ahrens, Ralph Matthes, and Anders Mörtberg. From Signatures to Monads in UniMath. *J. Autom. Reason.*, 63(2):285–318, 2019. doi:10.1007/s10817-018-9474-4.

5 Benedikt Ahrens, Ralph Matthes, and Anders Mörtberg. Implementing a category-theoretic framework for typed abstract syntax. In Andrei Popescu and Steve Zdancewic, editors, *CPP '22: 11th ACM SIGPLAN International Conference on Certified Programs and Proofs, Philadelphia, PA, USA, January 17 - 18, 2022*, pages 307–323. ACM, 2022. doi:10.1145/3497775.3503678.

6 Benedikt Ahrens, Paige Randall North, Michael Shulman, and Dimitris Tsementzis. The univalence principle. *CoRR*, abs/2102.06275, 2021. arXiv:2102.06275.

7 Benedikt Ahrens, Paige Randall North, and Niels van der Weide. Semantics for two-dimensional type theory. *CoRR*, abs/2201.10662, 2022. arXiv:2201.10662.

8 Nathanael Arkor and Dylan McDermott. The formal theory of relative monads. *arXiv preprint arXiv:2302.14014*, 2023.

9 Andrej Bauer, Jason Gross, Peter LeFanu Lumsdaine, Michael Shulman, Matthieu Sozeau, and Bas Spitters. The HoTT library: a formalization of homotopy type theory in Coq. In Yves Bertot and Viktor Vafeiadis, editors, *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, pages 164–172. ACM, 2017. doi:10.1145/3018610.3018615.

10 Jean Bénabou. Introduction to bicategories. In *Reports of the Midwest Category Seminar*, pages 1–77, Berlin, Heidelberg, 1967. Springer Berlin Heidelberg. doi:10.1007/BFb0074299.

11 P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models (extended abstract). In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 1994. doi:10.1007/BFb0022251.

12 Gabriella Böhm, Stephen Lack, and Ross Street. On the 2-categories of weak distributive laws. *Communications in Algebra*, 39(12):4567–4583, 2011.

13 Matteo Capucci and Bruno Gavranović. Actegories for the Working Amthematician. *arXiv preprint arXiv:2203.16351*, 2022.

14 Eugenia Cheng. Iterated distributive laws. *Mathematical Proceedings of the Cambridge Philosophical Society*, 150(3):459–487, 2011.

15 Ivan Di Liberti and Fosco Loregian. On the unicity of formal category theories. *arXiv preprint arXiv:1901.01594*, 2019.

16 Jeff Egger, Rasmus Ejlers Møgelberg, and Alex Simpson. The enriched effect calculus: syntax and semantics. *J. Log. Comput.*, 24(3):615–654, 2014. doi:10.1093/logcom/exs025.

17 Soichiro Fujii, Shin-ya Katsumata, and Paul-André Melliès. Towards a Formal Theory of Graded Monads. In Bart Jacobs and Christof Löding, editors, *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9634 of *Lecture Notes in Computer Science*, pages 513–530. Springer, 2016. doi:10.1007/978-3-662-49630-5_30.

18 Jason Gross, Adam Chlipala, and David I. Spivak. Experience Implementing a Performant Category-Theory Library in Coq. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8558 of *Lecture Notes in Computer Science*, pages 275–291. Springer, 2014. doi:10.1007/978-3-319-08970-6_18.

19 Jason Z. S. Hu and Jacques Carette. Formalizing category theory in Agda. In Catalin Hritcu and Andrei Popescu, editors, *CPP '21: 10th ACM SIGPLAN International Conference on Certified Programs and Proofs, Virtual Event, Denmark, January 17-19, 2021*, pages 327–342. ACM, 2021. doi:10.1145/3437992.3439922.

20 Martin Hyland and John Power. The category theoretic understanding of universal algebra: Lawvere theories and monads. *Electronic Notes in Theoretical Computer Science*, 172:437–458, 2007.

21 Niles Johnson and Donald Yau. *2-dimensional categories.* Oxford University Press, USA, 2021.

22 Simon L. Peyton Jones and Philip Wadler. Imperative Functional Programming. In Mary S. Van Deusen and Bernard Lang, editors, *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Charleston, South Carolina, USA, January 1993*, pages 71–84. ACM Press, 1993. doi:10.1145/158511.158524.

23 Krzysztof Kapulkin and Peter LeFanu Lumsdaine. The Simplicial Model of Univalent Foundations (after Voevodsky). *Journal of the European Mathematical Society*, 23(6):2071–2126, 2021. doi:10.4171/JEMS/1050.

24 Gregory Maxwell Kelly. Elementary observations on 2-categorical limits. *Bulletin of the Australian Mathematical Society*, 39(2):301–317, 1989.

25 Max Kelly. *Basic concepts of enriched category theory*, volume 64. CUP Archive, 1982.

26 Stephen Lack. Composing props. *Theory and Applications of Categories*, 13(9):147–163, 2004.

27 Stephen Lack. A 2-categories companion. In *Towards higher categories*, pages 105–191. Springer, 2010.

28 Stephen Lack and Ross Street. The formal theory of monads II. *Journal of pure and applied algebra*, 175(1-3):243–265, 2002.

29 Tom Leinster. Basic Bicategories, 1998. arXiv:math/9810017.

30 Paul Blain Levy. *Call-by-push-value: A Functional/imperative Synthesis*, volume 2. Springer Science & Business Media, 2012.

31 Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 2013.

32 Saunders MacLane and Robert Paré. Coherence for bicategories and indexed categories. *Journal of Pure and Applied Algebra*, 37:59–80, 1985.

33 The mathlib Community. The Lean mathematical library. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 367–381. ACM, 2020. doi:10.1145/3372885.3373824.

34 Paul-André Mellies. Categorical semantics of linear logic. *Panoramas et syntheses*, 27:15–215, 2009.

35 Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*, pages 14–23. IEEE Computer Society, 1989. doi:10.1109/LICS.1989.39155.

36 Eugenio Moggi. Notions of computation and monads. *Information and computation*, 93(1):55–92, 1991.

37 Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: a proof assistant for higher-order logic*, volume 2283. Springer Science & Business Media, 2002.

38 Ulf Norell. Dependently typed programming in Agda. In *Proceedings of the 4th international workshop on Types in language design and implementation*, pages 1–2, 2009.

39 Gordon Plotkin and John Power. Notions of computation determine monads. In *International Conference on Foundations of Software Science and Computation Structures*, pages 342–356. Springer, 2002.

40 Gordon Plotkin and John Power. Algebraic operations and generic effects. *Applied categorical structures*, 11(1):69–94, 2003.

41 John Power. Coherence for bicategories with finite bilimits I. In *Categories in computer science and logic*, volume 92 of *Contemporary Mathematics*, pages 341–347, 1989.

42 John Power. Enriched Lawvere theories. *Theory and Applications of Categories*, 6(7):83–93, 1999.

43 John Power and Edmund Robinson. A Characterization of PIE Limits. *Mathematical Proceedings of the Cambridge Philosophical Society*, 110:33–47, 1991.

44 John Power and Hiroshi Watanabe. Combining a monad and a comonad. *Theoretical Computer Science*, 280(1-2):137–162, 2002.

45 Egbert Rijke, Elisabeth Bonnevier, Jonathan Prieto-Cubides, et al. Univalent mathematics in Agda. https://unimath.github.io/agda-unimath/. URL: https://github.com/UniMath/agda-unimath/.

46 Eugene W. Stark. Bicategories. *Arch. Formal Proofs*, 2020, 2020. URL: https://www.isa-afp.org/entries/Bicategory.html.

47 Ross Street. The formal theory of monads. *Journal of Pure and Applied Algebra*, 2(2):149–168, 1972.

48 The Coq Development Team. The Coq Proof Assistant, September 2022. doi:10.5281/zenodo.7313584.

49 The 1Lab Development Team. The 1Lab. URL: https://1lab.dev.

50 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

51 Niccolò Veltri and Niels van der Weide. Constructing Higher Inductive Types as Groupoid Quotients. *Logical Methods in Computer Science*, Volume 17, Issue 2, April 2021. URL: https://lmcs.episciences.org/7391.

52 Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical Agda: a dependently typed programming language with univalence and higher inductive types. *Proc. ACM Program. Lang.*, 3(ICFP):87:1–87:29, 2019. doi:10.1145/3341691.

53 Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. Unimath — a computer-checked library of univalent mathematics. available at http://unimath.org. doi:10.5281/zenodo.7848572.

54 Kobe Wullaert, Ralph Matthes, and Benedikt Ahrens. Univalent monoidal categories. *arXiv preprint arXiv:2212.03146*, 2022.

Automata-Based Verification of Relational Properties of Functions over Algebraic Data Structures

Théo Losekoot
Université de Rennes, IRISA, France

Thomas Genet
Université de Rennes, IRISA, France

Thomas Jensen
Inria, Université de Rennes, France

Abstract

This paper is concerned with automatically proving properties about the input-output relation of functional programs operating over algebraic data types. Recent results show how to approximate the image of a functional program using a regular tree language. Though expressive, those techniques cannot prove properties relating the input and the output of a function, e.g., proving that the output of a function reversing a list has the same length as the input list. In this paper, we built upon those results and define a procedure to compute or over-approximate such a relation. Instead of representing the image of a function by a regular set of terms, we represent (an approximation of) the input-output relation by a regular set of tuples of terms. Regular languages of tuples of terms are recognized using a tree automaton recognizing convolutions of terms, where a convolution transforms a tuple of terms into a term built on tuples of symbols. Both the program and the properties are transformed into predicates and Constrained Horn clauses (CHCs). Then, using an Implication Counter Example procedure (ICE), we infer a model of the clauses, associating to each predicate a regular relation. In this ICE procedure, checking if a given model satisfies the clauses is undecidable in general. We overcome undecidability by proposing an incomplete but sound inference procedure for such relational regular properties. Though the procedure is incomplete, its implementation performs well on 120 examples. It efficiently proves non-trivial relational properties or finds counter-examples.

2012 ACM Subject Classification Theory of computation → Program verification; Theory of computation → Formal languages and automata theory

Keywords and phrases Formal verification, Tree automata, Constrained Horn Clauses, Model inference, Relational properties, Algebraic datatypes

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.7

Supplementary Material *Software*: `https://gitlab.inria.fr/tlosekoo/auto-forestation`
archived at `swh:1:dir:e25762023e1b0ed4326ca177d1084f71244c5b8d`
Dataset: `http://people.irisa.fr/Thomas.Genet/AutoForestation/`

1 Introduction

This paper is concerned with automatically proving properties about the input-output relation of functional programs operating over algebraic datatypes. We explore an approach in which both programs and properties are represented as Constrained Horn Clauses [2], i.e., Horn clauses with additional constraints expressed in an underlying theory. Using such representation, proving a property of a program is reduced to finding a model of the combined set of Horn clauses that represent the program and the property. We illustrate this using an example where we define the type of natural numbers and natural numbers lists, and two recursive functions, *len* computing the length of a list and *less* checking if a natural number is strictly less than another. We aim at (automatically) proving the logical properties $\forall x\ l.\ less\ Z\ (len\ Cons(x,l))$ and $\forall x\ l.\ less\ (len\ l)\ (len\ Cons(x,l))$. Here are the program in Ocaml-like syntax, the logical formulas for properties and their equivalent CHC representation. Note that n-ary functions (like unary *len*) are translated into $n+1$-ary relations (like binary Len). Because of this extra argument, we add a functionality constraint (the third clause of Len) for ensuring that the relation represents exactly the function. Without this functionality constraint, we could e.g. have a model where $\mathrm{Len}(Nil, S(Z))$ is true. Arity of predicates, like the binary *less*, do not change: Less is binary. In this case, we cannot use functionality constraint because the result is not reified. Instead, we use bi-implication to exclude all elements which are not in the relation defined by the OCaml function, e.g., exclude $\mathrm{Less}(S(S(Z)), S(Z))$.

```
type nat = Z | S of nat
type natlist = Nil | Cons of nat*natlist
```

```
let rec len (l : natlist) =
match l with
| Nil -> Z
| Cons(h,t) -> S (len t)
```

$\mathrm{Len}(Nil,\ Z).$
$\mathrm{Len}(\underline{l},\ \underline{n}) \Rightarrow \mathrm{Len}(Cons(\underline{x},\underline{l}),\ S(\underline{n})).$
$\mathrm{Len}(\underline{l},\ \underline{n_1}) \wedge \mathrm{Len}(\underline{l},\ \underline{n_2}) \Rightarrow \underline{n_1} = \underline{n_2}.$

```
let rec less (n : nat) (m : nat) =
match (n, m) with
| Z, S(_) -> true
| _, Z -> false
| S(n1), S(m1) -> less n1 m1
```

$\mathrm{Less}(Z,\ S(\underline{m})).$
$\mathrm{Less}(\underline{n},\ Z) \Rightarrow \mathrm{False}.$
$\mathrm{Less}(\underline{n},\ \underline{m}) \iff \mathrm{Less}(S(\underline{n}),\ S(\underline{m})).$

$\forall x\ l.\ less\ Z\ (len\ (Cons(x,l)))$
$\forall x\ l.\ less\ (len\ l)\ (len\ Cons(x,l))$

$\mathrm{Len}(Cons(\underline{x},\underline{l}),\ \underline{n}) \Rightarrow \mathrm{Less}(Z,\ \underline{n}).$
$\mathrm{Len}(\underline{l},\ \underline{n}) \wedge \mathrm{Len}(Cons(\underline{x},\underline{l}),\ \underline{n'}) \Rightarrow \mathrm{Less}(\underline{n},\ \underline{n'}).$

Our goal is thus to automatically infer a model of this set of clauses, i.e., solve the satisfiability problem for Constrained Horn Clauses over the theory of inductive datatypes. Tree automata [6] are a well-know formalism to represent, approximate, and infer models on functional programs [17, 11] or even on CHCs [16]. In all those works, the inferred model is not relational, i.e., it only consists of a regular set of unrelated terms. For instance, in our example, the first property $\forall x\ l.\ less\ Z\ (len\ (Cons(x,l)))$ is not relational and can thus be proven using regular sets like [16, 11, 17] do. To perform the proof, the solvers only need to consider two regular languages: $\mathcal{L}_{lists}$ containing all lists of natural numbers and $\mathcal{L}_{Cons+}$ containing all *non-empty* lists of natural numbers. Then, the proof is carried out by showing that if $l \in \mathcal{L}_{lists}$ then, for any natural number x, the term $Cons(x,l)$ belongs to $\mathcal{L}_{Cons+}$. Finally, since any list $l' \in \mathcal{L}_{Cons+}$ have a length strictly greater than 0 then the property is true.

On the opposite, the second property, $\forall x\ l.\ less\ (len\ l)\ (len\ Cons(x,l))$, is relational and, thus, out of the scope of the aforementioned approaches. We still have that if $l \in \mathcal{L}_{lists}$ then $cons(x,l) \in \mathcal{L}_{Cons+}$ but for any $l \in \mathcal{L}_{lists}$ and any $l' \in \mathcal{L}_{Cons+}$ we cannot prove that $less\ (len\ l)\ (len\ l')$. To preserve the relation between the two occurrences of the list l, we use convoluted automata [6] which can represent *regular relations* between terms. We build upon the preliminary results obtained in [12] and propose a sound but incomplete procedure for inferring an automaton that represents a model of the program and the property. This procedure is defined as an Implication Counter Example (ICE) procedure [8].

Contributions

- Definition of a sound model-checking procedure for CHCs on convoluted tree automata. We propose two sound optimisations of this procedure so as to make it efficient in practice;
- Definition of an ICE procedure for inferring models of CHCs;
- Definition of a specific over-approximation technique enlarging the class of properties which can be proved using regular models on CHCs programs;
- Implementation of the ICE procedure;
- On more than 120 examples, we show that our implementation automatically proves and disproves non-trivial examples.

This paper is organised as follows: In Section 2, we give an overview demonstrating the verification technique presented in this paper. In Section 3, we introduce the notions and notations. In Section 4, we briefly present how to encode functional programs into Horn clauses. In Section 5, we present a transformation from the model-checking procedure for CHCs into a search for a proof in a *proof system* representing the model. In Section 6, we present our use of the proof system for an efficient search. In Section 7, the ICE-procedure for inferring a model is defined. In Section 8, we present our approximation method. In Section 9, we discuss implementation-specific details and experiments. In Section 10, we present related work. Finally, we conclude in Section 11.

2 An overview of the verification procedure on an example

We continue our example of Section 1. We first give more details about the proof of the non-relational property $\forall x\ l.\ less\ Z\ (len\ (Cons(x,l)))$. To represent the set $\mathcal{L}_{lists}$ containing all lists of natural numbers and the set $\mathcal{L}_{Cons+}$ containing all non-empty lists of natural numbers, we use tree automata. Tree automata recognize sets of terms into states using *transitions*. *E.g.*, a tree automaton with states $\{q_{nat}, q_{Nil}, q_{Cons+}\}$ and transitions $\{Z() \to q_{nat},\ S(q_{nat}) \to q_{nat},\ Nil() \to q_{Nil},\ Cons(q_{nat}, q_{Nil}) \to q_{Cons+},\ Cons(q_{nat}, q_{Cons+}) \to q_{Cons+}\}$ recognizes Nil into the state q_{Nil} and any non-empty list of naturals into the state q_{Cons+}. To recognize a term, transitions are used to rewrite the term into a state, e.g, $Nil \to q_{Nil}$, and $Cons(S(Z), Nil) \to^* Cons(S(q_{nat}), q_{Nil}) \to Cons(q_{nat}, q_{Nil}) \to q_{Cons+}$. Similarly $Cons(Z, Cons(S(S(Z)), Nil)) \to^* q_{Cons+}$. To prove the property $\forall x\ l.less\ Z\ (len\ (Cons(x,l)))$ using such an automaton, it is enough to show that if l belongs to $\mathcal{L}_{lists}$ (whose terms are recognized by q_{Nil} or q_{Cons+}), then $Cons(x,l)$ belongs to $\mathcal{L}_{Cons+}$ (whose terms are recognized by q_{Cons+}). Using another automaton for Less, it is possible to show that $(len\ l')$, with l' recognized by q_{Cons+}, belongs to the language $\mathcal{L}_{pos}$ of strictly positive natural numbers, whereas $(len\ Nil)$ belongs to the language $\{Z\}$.

Now, we present a complete overview of our verification procedure for proving the second property $\forall x\ l.\ less\ (len\ l)\ (len\ Cons(x,l))$ which is relational and, thus, out of the scope of solvers like [16, 11, 17]. As shown before, the functions and the property are all translated into a set of CHCs. In the following, we denote by $\mathcal{C}$ this set. Given $\mathcal{C}$, we start the *model inference* phase whose objective is to infer a model of this set, named $\mathcal{M}$ in the following. For each relation R defined by the program, $\mathcal{M}$ contains an automaton $\mathcal{A}_R$ recognizing a language for the relation R. The model inference procedure can either

(i) succeed, i.e. find a model $\mathcal{M}$ satisfying $\mathcal{C}$, and the properties are proved, or

(ii) fail, i.e. find a contradiction, and the properties are disproved, or

(iii) never terminates.

This model inference is implemented as an Implication Counter-Example (ICE) procedure [8] between two entities: a learner and a teacher. The learner's goal is to infer a correct model using only feedback from the teacher. The teacher's goal is to verify if the clauses from $\mathcal{C}$ satisfy $\mathcal{M}$ (the model proposed by the learner) and to give feedback in the form of logical implications which are counter-examples.

Initially, $\mathcal{M}$ associates to each relation symbol an empty relation recognized by an empty automaton, denoted by $\mathcal{A}_\emptyset$. The relation recognized by $\mathcal{A}_\emptyset$, denoted by $\mathcal{R}(\mathcal{A}_\emptyset)$, is the empty relation. On our example, the initial value for $\mathcal{M}$ is thus $\mathcal{M} = \{\mathrm{Len} \mapsto \mathcal{A}_\emptyset, \mathrm{Less} \mapsto \mathcal{A}_\emptyset\}$.

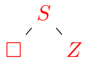
First iteration of the learner-teacher algorithm

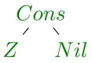
The learner proposes the model $\mathcal{M} = \{\mathrm{Len} \mapsto \mathcal{A}_\emptyset, \mathrm{Less} \mapsto \mathcal{A}_\emptyset\}$. The teacher checks if $\mathcal{M}$ satisfies each clause of $\mathcal{C}$, i.e., for each $\varphi \in \mathcal{C}$ it checks if $\mathcal{M} \models \varphi$. This is not true for the clause $\mathrm{Len}(Nil,\ Z)$ which imposes that the pair (Nil, Z) is part of the relation associated with Len. This is not the case here. Thus, the learner provides the ground clause $\mathrm{Len}(Nil,\ Z)$ as a counter-example.

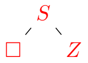
Second iteration of the learner-teacher algorithm

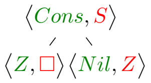
Starting from $\mathcal{M} = \{\mathrm{Len} \mapsto \mathcal{A}_\emptyset, \mathrm{Less} \mapsto \mathcal{A}_\emptyset\}$ and the counter-example $\mathrm{Len}(Nil,\ Z)$, the learner improves $\mathcal{M}$ in order to add the pair (Nil, Z) into the relation associated with Len, i.e., refines the automaton so as to recognize the pair (Nil, Z). For recognizing a relation, we need to extend the tree automaton formalism to recognize regular sets of tuples of terms. A solution proposed in [6] is to use a tree automaton recognizing convolutions of terms. A convolution transforms a tuple of terms into a term built on tuples of symbols. It does so by introducing new *convoluted* symbols which represent tuples of symbols. For example, to recognize the pair (Nil, Z) we define a new symbol $\langle Nil, Z\rangle$ and a tree automaton $\mathcal{A}_1$ with the state q_0 and the unique transition $\langle Nil, Z\rangle() \to q_0$. With such an automaton, the relation recognized by automaton $\mathcal{A}_1$ is $\mathcal{R}(\mathcal{A}_1) = \{(Nil, Z)\}$. Finally, we now have $\mathcal{M} = \{\mathrm{Len} \mapsto \mathcal{A}_1, \mathrm{Less} \mapsto \mathcal{A}_\emptyset\}$. Again, this model is given to the teacher which checks if $\mathcal{M} \models \mathcal{C}$. The teacher finds out that $\mathcal{M} \not\models \mathrm{Len}(\underline{l},\ \underline{n}) \Rightarrow \mathrm{Len}(Cons(\underline{x}, \underline{l}),\ S(\underline{n}))$. Indeed, since $(Nil, Z) \in \mathcal{L}(\mathcal{A}_1)$ we should have $(Cons(i, Nil), S(Z)) \in \mathcal{L}(\mathcal{A}_1)$ for all natural numbers i. The teacher provides a ground instance of this clause as a counter-example, e.g., $\mathrm{Len}(Nil,\ Z) \Rightarrow \mathrm{Len}(Cons(Z, Nil),\ S(Z))$.

Third iteration of the learner-teacher algorithm: Learner part

Starting from $\mathcal{M} = \{\mathrm{Len} \mapsto \mathcal{A}_1, \mathrm{Less} \mapsto \mathcal{A}_\emptyset\}$ and the counter-example obtained from the previous iteration $\mathrm{Len}(Nil,\ Z) \Rightarrow \mathrm{Len}(Cons(Z, Nil),\ S(Z))$, the learner should refine $\mathcal{A}_1$ into $\mathcal{A}_2$ so that it also recognizes the pair $(Cons(Z, Nil), S(Z))$. This time, to build the convolution we have to overlay the terms $Cons(Z, Nil)$ and $S(Z)$. However, because of the different arities of $Cons$ and S, the trees representing those two terms do not perfectly overlap. The convolution adds a padding symbol $\Box$ to complement trees in order to have a perfect overlap. Back to our example, with a convolution (known as right-convolution) the tree for $S(Z)$ becomes 

and the convolution of 

and 

is 

.

Thus, a refined automaton $\mathcal{A}_2$ recognizing both (Nil, Z) and $(Cons(Z, Nil), S(Z))$ has states $\{q_0, q_1, q_2\}$ and transitions $\{\langle Nil, Z\rangle() \to q_0,\ \langle Z, \Box\rangle() \to q_1,\ \langle Cons, S\rangle(q_1, q_0) \to q_2\}$. If we declare states q_0 and q_2 as final (meaning that we ignore the languages recognized by non final states) then $\mathcal{R}(\mathcal{A}_2) = \{(Nil, Z), (Cons(Z, Nil), S(Z))\}$.

A last phase of the ICE learning process is to reduce the number of states of the automaton and, doing so, possibly enlarge the recognized language. Note that this phase was skipped on automaton $\mathcal{A}_1$ because it has only one state. Reducing the number of states consists in finding state merging which are coherent w.r.t. the ground clauses sent by the teacher and coherent w.r.t. types of recognized languages. For instance, on $\mathcal{A}_2$, merging q_0 with q_2 is possible because both recognize pairs of lists and natural numbers. On the opposite, merging q_0 with q_1 is incorrect because q_0 recognize *pairs* of lists and q_1 only recognizes *a unique* natural number (omitting padding). After renaming q_2 to q_0, transitions of the automaton $\mathcal{A}_2$ become $\{\langle Nil, Z\rangle() \rightarrow q_0,\ \langle Z, \square\rangle() \rightarrow q_1,\ \langle Cons, S\rangle(q_1, q_0) \rightarrow q_0\}$. Note that this automaton now recognizes $\{(Nil, Z),\ (Cons(Z, Nil), S(Z)),\ (Cons(Z, Cons(Z, Nil)), S(S(Z))),\ \dots\}$, i.e., all pairs (l, n) where l is a list of Z whose length is n.

Conclusion of the learner-teacher algorithm

During following iterations, the learner-teacher proceed similarly to infer an automaton for Less and to finish inferring that of Len. Finally, during the 6-th iteration, the learner ends up on the following model $\mathcal{M} = \{\mathrm{Len} \mapsto \mathcal{A}_{\mathrm{Len}}, \mathrm{Less} \mapsto \mathcal{A}_{\mathrm{Less}}\}$ where $\mathcal{A}_{\mathrm{Len}}$ has final states $\{q_0\}$ and the transitions $\{\langle \square, S\rangle(q_1) \rightarrow q_1,\ \langle \square, Z\rangle() \rightarrow q_1,\ \langle Nil, Z\rangle() \rightarrow q_0,\ \langle Cons, S\rangle(q_1, q_0) \rightarrow q_0\}$. This automaton is close to automaton $\mathcal{A}_2$ except that it recognizes any natural number in place of Z in the list, i.e., it recognizes all pairs (l, n) where l is a list of natural numbers whose length is n. The automaton $\mathcal{A}_{\mathrm{Less}}$ has the final states $\{q_3\}$ and the transitions $\{\langle \square, Z\rangle() \rightarrow q_4,\ \langle \square, S\rangle(q_4) \rightarrow q_4,\ \langle Z, S\rangle(q_4) \rightarrow q_3,\ \langle S, S\rangle(q_3) \rightarrow q_3\}$. This model is given to the teacher which then checks that it satisfies all the clauses of $\mathcal{C}$. This terminates the verification and proves that $\forall x\ l.\ less\ (len\ l)\ (len\ Cons(x, l))$.

3 Prerequisites

3.1 Typed alphabet and term

▶ **Definition 1** (Typed alphabet). *A typed alphabet (Σ, τ, Γ) is a set of symbols Σ, a set of types Γ, and a typing function τ which assigns to each symbol f a type $\tau(f) = \tau_1 \times \ldots \times \tau_n \rightarrow \tau_0$ with $\forall i \in [\![0, n]\!], \tau_i \in \Gamma$ and $n \in \mathbb{N}$ varying for each symbol f. When $n = 0$, the symbol is a constant and does not take input. For $f \in \Sigma$ and $\tau(f) = \tau_1 \times \ldots \times \tau_n \rightarrow \tau_0$, we say that f is of arity n, written $|f| = n$, and that τ_0 is the* output type *of f, written $\tau_{out}(f) = \tau_0$. When clear from context, we identify the tuple (Σ, τ, Γ) with Σ.*

▶ **Definition 2** (Term). *A (typed) term t over an alphabet Σ is the data of a symbol $f \in \Sigma$, called the* root symbol *of t and written $Root(t)$, together with a list $t_1, \ldots, t_{|f|}$ of $|f|$ terms, called* children *of t, such that their type is compatible, i.e. $\tau(f) = \tau_{out}(Root(t_1)) \times \ldots \times \tau_{out}(Root(t_{|f|})) \rightarrow \tau_{out}(f)$. A term t is also written $f(t_1, \ldots, t_{|f|})$. We overload τ with $\tau(t) = \tau_{out}(Root(t))$. The set of terms over an alphabet Σ is written $\mathcal{T}(\Sigma)$.*

▶ **Definition 3** (Substitution). *A substitution σ is a finite map between variables and terms (which may contain variables). The application of a substitution σ to a variable x, written $\sigma(x)$, is defined as t if there exists a binding $(x, t) \in \sigma$ and x otherwise. The application of a substitution is generalized to terms by $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$. Even more generally, a substitution can be applied to any structure containing variables. The composition of substitution, which first applies σ_1 and then σ_2, is written $\sigma_1; \sigma_2$. The domain of a substitution is the set of variables for which a binding is defined and is written $dom(\sigma)$.*

A function $Vars$ is used without definition, if unambiguous, to fetch the set of variables contained in a structure. It can be called, for example, on a term or on a tuple of structures containing variables.

3.2 Tree automaton

▶ **Definition 4** (Tree automaton). *A (bottom-up) tree automaton $\mathcal{A} = (Q, Q_f, \Delta)$ over an alphabet Σ is given by a finite set of states Q, a set of final states $Q_f \subseteq Q$, and a set of transitions (or rules) Δ such that transitions are of the form $f(q_1, \ldots, q_{|f|}) \to q_0$, where $f \in \Sigma$ and $\forall i \in [\![0, |f|]\!], q_i \in Q$.*

▶ **Definition 5** (Language recognized by an automaton). *The set of terms recognized (or accepted) in a state q of an automaton $\mathcal{A}$ is inductively defined as $\mathcal{L}(\mathcal{A}, q) = \{f(t_1, \ldots, t_n) \mid f(q_1, \ldots, q_n) \to q \in \Delta \ \wedge \ \bigwedge_{i \in [\![1,n]\!]} t_i \in \mathcal{L}(\mathcal{A}, q_i)\}$. The language recognized by an automaton is $\mathcal{L}(\mathcal{A}) = \bigcup_{q_f \in Q_f} \mathcal{L}(\mathcal{A}, q_f)$.*

▶ **Definition 6** (Typed tree automaton). *A typed tree automaton is a tree automaton whose states are typed by types of the alphabet. We write $\tau(q)$ for the type of the state q. Transitions have to be compatible with the types of the symbols, i.e., for any rule $f(q_1, \ldots, q_n) \to q_0 \in \Delta$, $\tau(f) = \tau(q_1) \times \ldots \times \tau(q_n) \to \tau(q_0)$. All final states must be of the same type. The type of the automaton, written $\tau(\mathcal{A})$, is the type of its final states.*

We write $\overline{\mathcal{A}}$ for the complement of the automaton $\mathcal{A}$ w.r.t its type, i.e., $\mathcal{L}(\overline{\mathcal{A}}) = \{t \mid \tau(t) = \tau(\mathcal{A}) \wedge t \notin \mathcal{L}(\mathcal{A})\}$. We also use Q, Q_f, and Δ as accessors, that is, as functions to respectively extract states, final states, and transitions from an automaton. We usually write t or $f(t_1, \ldots, t_n)$ for terms, q for a state, and $\mathcal{A}$ for an automaton. Tuple of elements $(e_1, \ldots, e_n)$ are also written $\vec{e}$ and $\vec{e}[i]$ means e_i.

3.3 Automata recognizing a relation

There exist multiple formalism for representing a relation on terms with an automaton. They differ in their expressive power, closure properties, and decision procedure complexity. The most well known are *tuple automata*, *ground tree transducers*, and *automata on convoluted terms*, all described in [6]. We will pursue an approach based on automata on convoluted terms, or simply convoluted automata.

Convoluted automata are defined w.r.t an operation called *convolution* which transforms an n-tuple of terms into a unique term whose symbols are n-tuple of symbols. Intuitively, an automaton defined on this alphabet of tuple reads n terms at the same time, thereby recognizing a relation. The standard convolution operator amounts to overlaying the (syntax tree of the) terms, starting from the root, and adding a padding symbol $\square \notin \Sigma$ (of type $\tau_\square$) as there is an arity mismatch between symbols. To this end, we extend any alphabet Σ to $\Sigma_\square = \Sigma \cup \{\square\}$. We call this standard convolution the *left convolution*, in order to distinguish it from other convolutions, e.g. the right convolution, that has been used in section 2 and in the rest of the paper. We first define left-convolution of a tuple of tuple, and then use it to define convolution of terms.

▶ **Definition 7** (Left-convolution).

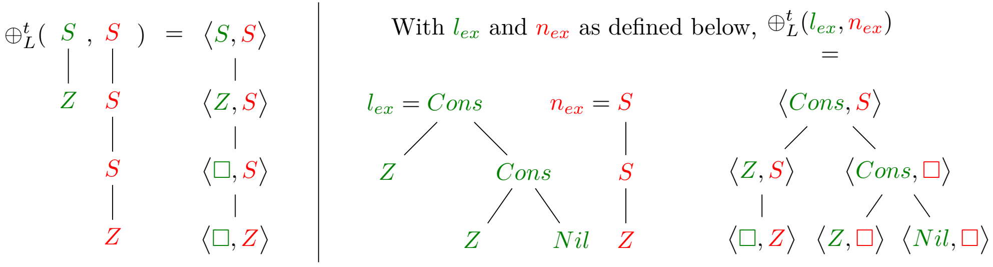
$$\oplus_L\left((e_1^1, \ldots, e_1^{k_1}), \ldots, (e_n^1, \ldots, e_n^{k_n})\right) = \left((\overline{e_1^1}, \ldots, \overline{e_n^1}), \ldots, (\overline{e_1^k}, \ldots, \overline{e_n^k})\right)$$

$$\textit{with } k = \max_{i \in [\![1,n]\!]}(k_i) \quad \textit{and} \quad \forall i \in [\![1,n]\!], \forall j \in [\![1,k]\!],\ \overline{e_i^j} = e_i^j \textit{ if } j \leq k_i \textit{ and } \square \textit{ otherwise}$$

▶ **Definition 8** (Left-convolution of terms). *The n-ary* left-convolution*, written $\oplus^t_L$, takes n terms $(t_1, \ldots, t_n)$ on an alphabet $\Sigma_\Box$ and returns a term $\oplus^t_L(t_1, \ldots, t_n)$ on a convoluted alphabet $\Sigma_{\oplus_L} = {\Sigma_\Box}^n$ whose elements are written $\langle f_1, \ldots, f_n \rangle$ or $\vec{f}$. The left-convolution of n terms is recursively defined as:*

$$\oplus^t_L(f_1(\vec{t_1}), \ldots, f_n(\vec{t_n})) = \langle f_1, \ldots, f_n \rangle(\oplus^t_L(\vec{t'_1}), \ldots, \oplus^t_L(\vec{t'_k})) \textit{ with } (\vec{t'_1}, \ldots, \vec{t'_k}) = \oplus_L(\vec{t_1}, \ldots, \vec{t_n})$$

▶ **Example 9** (Left convoluted terms). Let $\Sigma_{ex} = \{Z, S, Nil, Cons\}$, with $\tau(Z) = nat$, $\tau(S) = nat \to nat$, $\tau(Nil) = natlist$, $\tau(Cons) = nat \times natlist \to natlist$, be a typed alphabet for natural numbers and lists of natural numbers. Following are two examples of left convolution of terms.

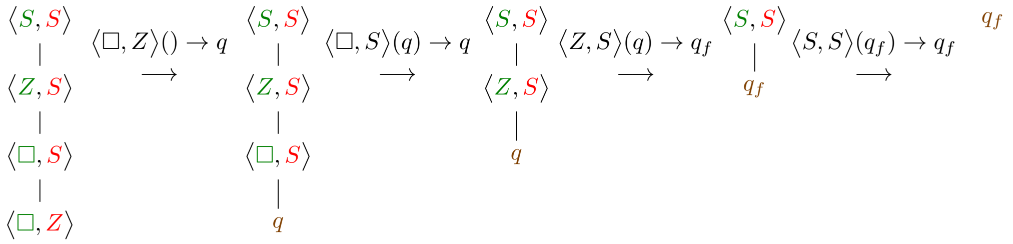


Note that, due to type constraints, $\mathcal{T}(\Sigma_\Box) = \mathcal{T}(\Sigma) \cup \{\Box\}$. The left-convolution $\oplus^t_L$ of n terms is an isomorphism between $\mathcal{T}(\Sigma_\Box)^n$ and $\mathcal{T}(\Sigma_{\oplus_L})$. Automata recognizing convoluted terms thus recognize relations on $\mathcal{T}(\Sigma_\Box)^n$.

▶ **Definition 10** (Regular relation). *A relation recognized by a tree automaton is said to be* regular*. The relation recognized by automaton $\mathcal{A}$ is $\mathcal{R}(\mathcal{A}) = {\oplus_L}^{-1}(\mathcal{L}(\mathcal{A})) = \{\vec{t} \mid \oplus_L(\vec{t}) \in \mathcal{L}(\mathcal{A})\}$. Similarly, the relation recognized by state q of $\mathcal{A}$ is $\mathcal{R}(\mathcal{A}, q) = {\oplus_L}^{-1}(\mathcal{L}(\mathcal{A}, q))$.*

We impose that the type of any final state q_f is $\tau_\Box$-free, that is, $\tau(q_f) = (\tau_1, \ldots, \tau_n)$ with $\forall i \in [\![i, n]\!], \tau_i \neq \tau_\Box$. This ensures that an automaton defines a relation between terms of $\mathcal{T}(\Sigma)$, i.e. terms without padding.

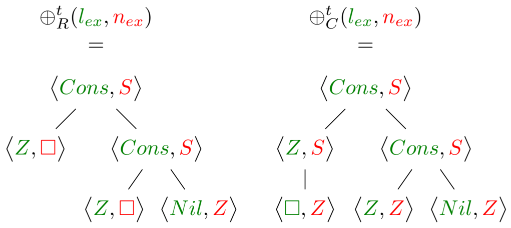
▶ **Example 11** (Convoluted automata). Let $\mathcal{A}_<$ be the automaton with states $\{q, q_f\}$, of which q_f is final, and transitions $\{\langle \Box, Z\rangle() \to q,\ \ \langle \Box, S\rangle(q) \to q,\ \ \langle Z, S\rangle(q) \to q_f,\ \ \langle S, S\rangle(q_f) \to q_f\}$. $\mathcal{R}(\mathcal{A}_<)$ is the $<$ relation on Peano numbers and $\tau(\mathcal{A}_<) = nat \times nat$. For example, the convolution of $S(Z)$ and $S(S(S(Z)))$ is recognized by this automaton, as shown below.


$\langle S, S\rangle$
$\langle Z, S\rangle$
$\langle \Box, S\rangle$
$\langle \Box, Z\rangle$
$\langle \Box, Z\rangle() \to q$
$\langle \Box, S\rangle(q) \to q$
$\langle Z, S\rangle(q) \to q_f$
$\langle S, S\rangle(q_f) \to q_f$
q
q_f

Convolutions and their expressivity

Which relations are representable by convoluted tree automaton highly depends on the precise datatypes definition. For example, when using the left-convolution, the Len relation can only be represented if the Cons constructor had its arguments swapped. This is because left-convoluting a list l and a natural number n will relate n with the left-most branch of l. Instead of modifying constructors, we can define other convolutions. The *right convolution*, written $\oplus_R$, is defined similarly to $\oplus_L$ but adds padding to the left of terms instead of to the right. This right convolution is effective for proving properties relating lists and unary natural numbers. Finally, we define the *complete convolution*, written $\oplus_C$, which is more expressive than both the left and the right convolution. This complete convolution relates every combination of tuple's element, which results in overlaying every same-depth constructor when convoluting terms. The complete convolution has the advantage of not depending on the constructor argument's order and being able to duplicate terms, but the drawback of generating big convoluted terms. Both convolution are extended to terms in the same way $\oplus_L$ was.

▶ **Example 12.** On the left is depicted the *right* convolution of l_{ex} and n_{ex} (of example 11), and on the right their *complete* convolution. Note how n_{ex}'s constructors have been duplicated in the complete convolution.



Since definitions of this paper hold for any convolution, we write $\bigcirc$ for any of $\oplus_L$, $\oplus_R$, or $\oplus_C$.

4 Functional programs and their logical representation

Regular models of functional programs

We consider first-order monomorphic functional programs. Such programs define a set of functions of the form $f : \tau_1 \to \ldots \to \tau_n$ and of the form $f : \tau_1 \to \ldots \to \tau_n \to bool$, with each τ_i being an algebraic datatype. Each of these can be viewed as a relation on $\tau_1 \times \ldots \times \tau_n$. Formally, these relations constitute a (relational) first-order structure on L, with L being the signature (the set of relation symbols together with their type). In our setting, the structures are typed, i.e. a relation R of type $\tau(R) = \tau_1 \times \ldots \times \tau_n$ only relates terms $t_1, \ldots, t_n$ satisfying $\forall i \in [\![1, n]\!], \tau(t_i) = \tau_i$.

▶ **Definition 13** (Regular model). *A* regular model *is a function $\mathcal{M}$ mapping each relation symbol $R \in L$ to an automaton $\mathcal{A}_R$. $\mathcal{M}$ denotes $\mathcal{S}_\mathcal{M}$, the L-structure where every $R \in L$ is interpreted as $\mathcal{R}(\mathcal{A}_R)$. We naturally extend first-order semantic judgement to write $\mathcal{M} \models \varphi$ for $\mathcal{S}_\mathcal{M} \models \varphi$.*

Regular models are close in essence to *automatic structures*. Automatic structures [14, 15, 10] are a kind of recursive structures [13], which are part of the study of finite representation of structures. Automatic structures have been studied for their decidable first-order theory. We shall use *tree automata* to represent first-order structures that model functional programs. This allows us to use specific and efficient methods for property checking.

We use Constrained Horn Clauses (CHCs) [2] as representation of our programs. CHCs are first-order Horn clauses with additional constraints from a theory T (see example in the Introduction). A CHC on a signature L is a closed formula of the form $\forall \vec{x}, \psi(\vec{x}) \wedge R_1(\vec{x}_1) \wedge \ldots \wedge R_n(\vec{x}_n) \Rightarrow R_0(\vec{x}_0)$, where $\forall i \in [\![0, n]\!], R_i \in L$. The formula $\psi(\vec{x})$ adds theory-related constraints. The semantic judgement $\mathcal{S} \models \varphi$ is standard first-order logic (modulo theory T). We usually leave out the universal quantifiers in front of CHCs: every variable in a formula is implicitly universally quantified. In our setting, we use the theory of inductive datatypes [1] over an alphabet Σ, which means that the value of variables are within $\mathcal{T}(\Sigma)$ and constraints are of the form $x = f(\vec{y})$, where $f \in \Sigma$, x is a variable and $\vec{y}$ is a tuple of variables. For simplicity, we sometimes write $R(t)$ for $x = t \wedge R(x)$. A *ground* CHC is one that has no variables or, in our context, where every variable's value is completely determined by datatypes constraints (for example, $x = Nil \Rightarrow R(x)$ is considered ground).

Our encoding of functional programs into clauses prevents us from using Horn clauses in the translation of the if-then-else construct. For example, the simple translation of **let** f x = **if** p x **then** e **else** e' yields the two clauses $\{\mathrm{P}(x) \Rightarrow \mathrm{F}(x,\ e),\ \ \neg\mathrm{P}(x) \Rightarrow \mathrm{F}(x,\ e')\}$. We therefore use non-Horn constrained clauses for modeling such functions. In the following, we handle a negated literal in the body as a positive head, in disjunction with the other heads. Other work [20] models similar programs with Horn clauses by reifying the truth of a predicate in the terms as its last argument, allowing to negate it in the body of a clause. Both ways of treating negation seems viable for our purpose but we have only experimented with the first one.

5 Model-checking of regular structures

In this section, we present the procedure for checking the truth of a given CHC φ in a model $\mathcal{M}$, i.e., check if $\mathcal{M} \models \varphi$. This model-checking fulfills the *teacher* role of the ICE model inference procedure (See sections 2 and 7). This procedure is devised as a counter-example search. A counter-example is a ground instantiation of each variable of φ, written as a ground substitution σ, that disproves $\mathcal{M} \models \varphi$. This procedure either returns *None* if $\mathcal{M} \models \varphi$, and otherwise *Some*$(\sigma)$, with σ a counter-example. However, this problem is undecidable in general, as showed in [18]. Therefore the procedure given here is correct but incomplete, that is, it may diverge.

The model checking problem can be seen as a type checking procedure where typing rules correspond to rules of automata.

▶ **Definition 14** (Type checking instance). *A* typing obligation $\omega = [\langle x_1, \ldots, x_n\rangle : (\mathcal{A}, q)]$ *is the data of a tuple* $\langle x_1, \ldots, x_n\rangle$*, with each* x_i *being a variable or* $\square$*, and of a* target type $(\mathcal{A}, q)$*. A* typing problem (E, Ω) *is a set of typing obligations* Ω *together with a set of constraints* E*, each of the form* $x = f(\vec{y})$ *with* f *a symbol of* Σ*. A solution for a typing problem is a substitution* $\sigma : \mathcal{X} \to \mathcal{T}(\Sigma)$ *that satisfies every typing obligation and constraint:*

$$\sigma \models (E, \Omega) \quad \doteq \quad \sigma \models \Omega \quad \wedge \quad \sigma \models E \qquad\qquad \textit{with}$$

$$\sigma \models \Omega \ \doteq\ \big(\forall [\vec{x} : (\mathcal{A}, q)] \in \Omega,\ \sigma(\vec{x}) \in \mathcal{R}(\mathcal{A}, q)\big) \quad \textit{and}$$

$$\sigma \models E \ \doteq\ \big(\forall (x = f(\vec{y})) \in E,\ \sigma(x) = f(\sigma(\vec{y}))\big)$$

▶ **Definition 15** (Coherence of a constraint set). *A set of constraints* E *is said to be* coherent *if it admits a syntactic unifier. The most general unifier (MGU) of a coherent set* E *is written* σ_E*.*

Note that, given a typing problem (E, Ω) with a coherent E, any σ such that $\sigma \models (E, \Omega)$ is equivalent to a σ' such that $\sigma_E; \sigma' \models \Omega$ (by characterisation of the MGU).

▶ **Definition 16** (Model checking as type checking).
Let some CHC formula $\varphi = \psi(\vec{x}) \wedge R_1(\vec{x}_1) \wedge \ldots \wedge R_n(\vec{x}_n) \Rightarrow R_0(\vec{x}_0)$ and model $\mathcal{M}$.
The set of typing problems associated to φ and $\mathcal{M}$ is $tp(\varphi, \mathcal{M}) = \{(\psi(\vec{x}), \Omega) \mid \Omega \in \Omega_s\}$ with

$$\Omega_s = \Big\{\{[\vec{x}_1 : (\mathcal{A}_1, q_1)], \ldots, [\vec{x}_n : (\mathcal{A}_n, q_n)], [\vec{x}_0 : (\mathcal{A}_0, q_0)]\} \mid \\ \mathcal{A}_1 = \mathcal{M}(R_1) \wedge \ldots \wedge \mathcal{A}_n = \mathcal{M}(R_n) \wedge \mathcal{A}_0 = \overline{\mathcal{M}(R_0)} \wedge \forall i \in [\![0, n]\!], q_i \in Q_f(\mathcal{A}_i)\Big\}$$

The set of solutions σ to $tp(\mathcal{M}, \varphi)$ is the same as the set of counter-examples to $\mathcal{M} \models \varphi$. Intuitively, for such a counter-example to exist, it should validate the atoms $R_1(\vec{x}_1), \ldots, R_n(\vec{x}_n)$ (i.e. be recognized by $\mathcal{M}(R_1) \ldots, \mathcal{M}(R_n)$) and invalidate the atom $R_0(\vec{x}_0)$ (i.e. be recognized by $\overline{\mathcal{M}(R_0)}$).

▶ **Theorem 17** (Model checking as type checking).
For each model $\mathcal{M}$ and CHC property φ, $\mathcal{M} \not\models \varphi \iff \exists \sigma, \exists (E, \Omega) \in tp(\mathcal{M}, \varphi), \ \sigma \models (E, \Omega)$.

▶ **Example 18** (Model checking a property). Let φ be Len($\underline{l}$, $\underline{n}$) $\Rightarrow$ Even($\underline{n}$), a formula stating that all lists are of even length. Let $\mathcal{M} = \{\text{Len} \mapsto \mathcal{A}_{\text{Len}}, \ \ \text{Even} \mapsto \mathcal{A}_{\text{Even}}\}$ where $\mathcal{A}_{\text{Len}}$ and $\mathcal{A}_{\text{Even}}$ respectively define the length relation on integer lists and the even predicate of integers. $\mathcal{A}_{\text{Len}}$ has states $\{q_f, q\}$, final states $\{q_f\}$, and rules $\{(A) : \ \big\langle Z, \Box\big\rangle() \to q, \quad (B) : \ \big\langle S, \Box\big\rangle(q) \to q, \quad (C) : \ \big\langle Cons, S\big\rangle(q, q_f) \to q_f, \quad (D) : \ \big\langle Nil, Z\big\rangle() \to q_f\}$. $\mathcal{A}_{\text{Even}}$ has states $\{q_e, q_o\}$, final states $\{q_e\}$, and rules $\{(1) : \ \big\langle Z\big\rangle() \to q_e, \quad (2) : \ \big\langle S\big\rangle(q_o) \to q_e, \quad (3) : \ \big\langle S\big\rangle(q_e) \to q_o\}$.
To check whether $\mathcal{M} \not\models \varphi$, we first translate $(\mathcal{M}, \varphi)$ into a typing problem instance. Note that Even appears in the head of the property φ, therefore we will need to complement $\mathcal{A}_{\text{Even}}$. We write its complement $\mathcal{A}_{\text{Odd}}$, which is the same automaton but with final states $\{q_o\}$.

$$tp(\mathcal{M}, \varphi) = \big\{(E_0, \Omega_0)\big\} \text{ with } E_0 = \emptyset \text{ and } \Omega_0 = \big\{[\big\langle l, n\big\rangle : (\mathcal{A}_{\text{Len}}, q_f)], [\big\langle n\big\rangle : (\mathcal{A}_{\text{Odd}}, q_o)]\big\}$$

In this case, $tp(\mathcal{M}, \varphi)$ only contains one element (as each automaton only has one final state), therefore $\mathcal{M} \not\models \varphi \iff \exists \sigma, \ \sigma \models (\emptyset, \Omega_0)$.

5.1 Proof system

A proof obligation is the assertion that some typing problem (E, Ω) admits a solution, which is written as $\vdash (E, \Omega)$. We first define the *unfolding* of typing obligations and then the proof system. Any solution for a typing obligation $\omega = [\big\langle x_1, \ldots, x_n\big\rangle : (\mathcal{A}, q)]$ can be found by following transitions of the automaton $\mathcal{A}$. A transition $\big\langle f_1, \ldots, f_n\big\rangle(q_1, \ldots, q_k) \to q$ of $\mathcal{A}$ (note that q is the same between the typing obligation and the rule's goal state) can act as a typing rule whose application generates k new typing obligations (one for each sub-state q_j of the rule) and n new algebraic datatype constraints, the i^{th} stating that variable x_i is of the form $f_i(\vec{x}_i)$ with $\vec{x}_i$ some fresh variables. We formally define this step as *unfolding* a typing obligation.

▶ **Definition 19** (Unfolding a typing obligation).
$unfold([\big\langle x_1, \ldots, x_n\big\rangle : (\mathcal{A}, q)]) = \big\{\big(E_r, \Omega_r\big) \mid r \in \Delta(\mathcal{A}) \wedge r = \big\langle f_1, \ldots, f_n\big\rangle(q_1, \ldots, q_k) \to q\}$ with $E_r = \{x_i = f_i(\vec{x}_i) \mid i \in [\![1, n]\!]\}$ and $\Omega_r = \{[\bigcirc(\vec{x}_1, \ldots, \vec{x}_n)[j] : (\mathcal{A}, q_j)] \mid j \in [\![1, k]\!]\}$ where $\forall i \in [\![1, n]\!], \ \vec{x}_i$ are fresh variables.

▶ **Example 20** (Unfolding)**.** Continuing with Example 18, we set $\omega_1 = [\langle l, n\rangle : (\mathcal{A}_{\text{Len}}, q_f)]$ and $\omega_0 = [\langle n\rangle : (\mathcal{A}_{\text{Odd}}, q_o)]$. Now, ω_0 can be unfolded by rules $\{(3)\}$ and ω_1 by $\{(C), (D)\}$.

$$unfold(\omega_0) = \{(E_{(3)}, \Omega_{(3)})\} \text{ with } E_{(3)} = \{n = S(m)\} \text{ and } \Omega_{(3)} = [\langle m\rangle : (\mathcal{A}_{\text{Odd}}, q_e)].$$
$$unfold(\omega_1) = \{(E_{(D)}, \Omega_{(D)}),\ (E_{(C)}, \Omega_{(C)})\} \text{ with}$$
$$E_{(D)} = \{l = Nil,\ n = Z\},\quad \Omega_{(D)} = \emptyset,$$
$$E_{(C)} = \{l = Cons(l_1, l_2),\ n = S(n_1)\},$$
$$\Omega_{(C)} = \{[\langle l_1, \Box\rangle : (\mathcal{A}_{\text{Len}}, q_n)],\quad [\langle l_2, n_1\rangle : (\mathcal{A}_{\text{Len}}, q_f)]\}.$$

We define the unfolding of a set of typing obligations as the (combination of) unfolding of *each* typing obligation at the same time, that is the application of one rule of the automaton to each typing obligation.

▶ **Definition 21** (Unfolding a typing problem)**.**

$$unfolds(\Omega) = \{(\bigcup_{\omega\in\Omega} E_\omega, \bigcup_{\omega\in\Omega} \Omega_\omega,) \mid \forall\omega \in \Omega, (E_\omega, \Omega_\omega) \in unfold(\omega)\}$$

▶ **Example 22.** $unfolds(\{\omega_0, \omega_1\}) = \{(E_{(3)} \cup E_{(D)},\ \Omega_{(3)} \cup \Omega_{(D)}),\ (E_{(3)} \cup E_{(C)},\ \Omega_{(3)} \cup \Omega_{(C)})\}$

Finally, the proof system on typing problems consists of two deduction rules. The rule Conclude concludes a proof when no typing obligation are left and when the algebraic datatype constraints are consistent. The rule Step applies unfolding of typing problems using rules of the tree automaton.

▶ **Definition 23** (Proof system)**.** *Our proof system contains two rules.*

$$\textsc{Conclude}\ \frac{}{\vdash (E, \emptyset)}\ \ \textit{if } Coherent(E) \qquad \textsc{Step}\ \frac{\vdash (E \cup E', \Omega')}{\vdash (E, \Omega)}\ \ \textit{if } Coherent(E \cup E') \textit{ and } (E', \Omega') \in unfolds(\Omega)$$

▶ **Example 24.** Continuing example 20, we build a proof tree of $\vdash (E_0,\ \Omega_0)$. Rule Conclude cannot be immediately applied, so let us consider Step, and thus $unfolds(\Omega_0)$.

Its element $(E_{(3)} \cup E_{(D)},\ \Omega_{(3)} \cup \Omega_{(D)})$ can be discarded because $E_{(3)} \cup E_{(D)}$ is contradictory, as both constraints $n = Z$ and $n = S(m)$ are present. Its other element, $(E_{(3)} \cup E_{(C)},\ \Omega_{(3)} \cup \Omega_{(C)})$, is coherent, so we can apply the Step rule. We write it (E_1, Ω_1) where $E_1 = \{l = Cons(l_1, l_2),\ n = S(n_1), n = S(m)\}$ and Ω_1 is the set of typing obligations $\Omega_1 = \{[\langle l_1, \Box\rangle : (\mathcal{A}_{\text{Len}}, q_n)],\quad [\langle l_2, n_1\rangle : (\mathcal{A}_{\text{Len}}, q_f)],\quad [\langle m\rangle : (\mathcal{A}_{\text{Odd}}, q_e)]\}$. We now have the new typing problem $(E_0 \cup E_1, \Omega_1)$. Rule Conclude still cannot be applied. Then, $unfolds(\Omega_1)$ has 8 elements, only 4 of which are coherent. Its four coherent element can be seen as two times the almost-same two elements, the only difference being which rule has been applied to $[\langle l_1, \Box\rangle : (\mathcal{A}_{\text{Len}}, q_n)]$. For this example, we only show the two elements that used rule (A), (E_2, Ω_2) and (E_2', Ω_2') with

$$E_2 = \{l_1 = Z,\ l_2 = Nil,\ n_1 = Z,\ m = Z\},\quad \Omega_2 = \emptyset,$$
$$E_2' = \{l_1 = Z,\ l_2 = Cons(l_{21}, l_{22}),\ n_1 = S(n_{11}),\ m = S(m_1)\},$$
$$\Omega_2' = \{[\langle l_{21}, \Box\rangle : (\mathcal{A}_{\text{Len}}, q_n)], [\langle l_{22}, n_{11}\rangle : (\mathcal{A}_{\text{Len}}, q_f)], [\langle m_1\rangle : (\mathcal{A}_{\text{Odd}}, q_o)]\}$$

Constraints $E_1 \cup E_2$ are coherent and Ω_2 is empty, so rule Conclude can be applied and a solution can be built from $E_0 \cup E_1 \cup E_2$, that is $\{n \mapsto S(Z),\ l \mapsto Cons(Z, Nil)\}$. The final proof tree is depicted below. For now, every proof tree is a single line. This will no longer be true with the introduction of the rule Split in section 6.

$$\text{Step}\;\dfrac{\text{Step}\;\dfrac{\text{Conclude}\;\dfrac{}{\vdash (E_1 \cup E_2, \emptyset)}}{\vdash (E_1, \Omega_1)}}{\vdash (\emptyset, \Omega_0)}$$

▶ **Definition 25** (Heights). *We define a useful metric for proofs, the* height*:*

- *The height of a term $t = f(t_1, \dots, t_n)$ is inductively defined as $h(t) = 1 + \max_{i \in \llbracket 1,n \rrbracket}(h(t_i))$.*
- *The height of a ground formula $\underline{\varphi}$, written $h(\underline{\varphi})$, is defined as the height of the highest term occurring in it.*
- *The height of a substitution σ together with a typing obligation $\omega = [\langle x_1, \dots, x_n \rangle : (\mathcal{A}, q)]$ is defined as $h(\sigma, \omega) = \max_{i \in \llbracket 1,n \rrbracket}(h(\sigma(x_i)))$.*
- *The height of a substitution with a set of typing obligations is $h(\sigma, \Omega) = \max_{\omega \in \Omega}(h(\sigma, \omega))$.*
- *The height of a proof tree T, written $h(T)$, is defined as the maximal number of occurrences of the Step rule on a branch.*

▶ **Theorem 26** (Proof system is correct and complete). *We have $\forall (E, \Omega), \big(\exists \sigma, \sigma \models (E, \Omega)\big) \iff \vdash (E, \Omega)$. More precisely, for any (E, Ω) and $n \in \mathbb{N}$,*

(A) *For any proof tree T of $\vdash (E, \Omega)$ with $h(T) = n$, there exists a substitution σ such that $\sigma \models (E, \Omega)$ and $h(\sigma, \Omega) = n$.*

(B) *For any substitution σ such that $\sigma \models (E, \Omega)$ and $h(\sigma, \Omega) = n$, there exists a proof tree T of $\vdash (E, \Omega)$ such that $h(T) = n$.*

The proof can be found in Appendix A.

▶ **Corollary 27** (Smallest counter-example). *By theorem 26, a breadth-first exploration of proof trees for a given typing problem (E, Ω) admitting a solution yields a solution of* minimal height*, that is, a substitution σ that has the minimal value $h(\sigma, \Omega)$.*

6 Proof search procedure

The search of a proof or the certainty of the absence of proof is implemented as a breadth-first exploration of the above-defined proof trees. This problem is undecidable in general [18], thus this procedure either finds a solution to the typing problem (i.e. a counter-example to $\mathcal{M} \models \varphi$) or tries every possibility and finds no counter-example (meaning that $\mathcal{M} \models \varphi$), or diverges. We present two sound optimizations which significantly improve the proving and disproving power of the proof search procedure. Using those optimizations makes this procedure usable and efficient in practice (see experiments in Section 9).

The first optimisation consists in *splitting independent typing obligations* when they do not depend on each other.

▶ **Definition 28** (Independence). *Let (E, Ω) be a typing problem with E coherent. $\Omega_a \subseteq \Omega$ and $\Omega_b \subseteq \Omega$ are said* independent w.r.t. *E, written $\Omega_a \parallel^E \Omega_b$, when*

$$\forall \sigma_a, \sigma_b, [\sigma_E; \sigma_a \models \Omega_a \wedge \sigma_E; \sigma_b \models \Omega_b] \Rightarrow [\forall x \in Vars(\sigma_E(\Omega_a)) \cap Vars(\sigma_E(\Omega_b)), \sigma_a(x) = \sigma_b(x)]$$

Therefore, any two solutions σ'_a of (E, Ω_a) and σ'_b of (E, Ω_b) with $\Omega_a \parallel^E \Omega$ can first be factorized by σ_E by letting σ_a and σ_b such that $\sigma'_a = \sigma_E; \sigma_a$ and $\sigma'_b = \sigma_E; \sigma_b$ and then joined into $\sigma_{ab} = \sigma_a \cup \sigma_b$, and we have $\sigma_E; \sigma_{ab} \models (E, \Omega_a \cup \Omega_b)$. Finding a most precise partitioning of (E, Ω) into independent sub-problems is hard, as it may require to examine the shape of automata. We define below a safe and easy-to-compute approximation of these independence classes that splits typing obligations whose variables cannot be related even using the equalities of E.

▶ **Definition 29** (Splitting). *Let E be a set of constraints. Let $V_E([\vec{x} : (\mathcal{A}, q)]) \doteq Vars(\sigma_E(\vec{x}))$. The set $V_E([\vec{x} : (\mathcal{A}, q)])$ is the set of variables remaining in a typing obligation after application of the most general unifier σ_E of E. Note how $(\mathcal{A}, q)$ has not been used. We define $D_E \subseteq \Omega \times \Omega$ as $D_E(\omega_1, \omega_2) \doteq (V_E(\omega_1) \cap V_E(\omega_2) \neq \emptyset)$. Since D_E is symmetric, its reflexive and transitive closure D_E^* is an equivalence relation. We define the function* $\text{Split}(E, \Omega)$ *to return the equivalence classes of D_E^* defined on Ω.*

▶ **Lemma 30.** $\forall \Omega_1, \Omega_2 \in \text{Split}(E, \Omega),\ \Omega_1 \parallel^E \Omega_2$.

Proof. For any $\Omega_1, \Omega_2 \in \text{Split}(E, \Omega)$, $Vars(\sigma_E(\Omega_1)) \cap Vars(\sigma_E(\Omega_2)) = \emptyset$. Therefore $\Omega_1 \parallel^E \Omega_2$. ◀

This separation into independent problems makes the search less combinatorial and give rise to a new rule for our typing system:

$$\textsc{Split}\ \frac{\vdash (E, \Omega_1) \qquad \dots \qquad \vdash (E, \Omega_n)}{\vdash (E, \Omega)} \qquad \text{with } \{\Omega_1, \dots, \Omega_n\} = \text{Split}(E, \Omega)$$

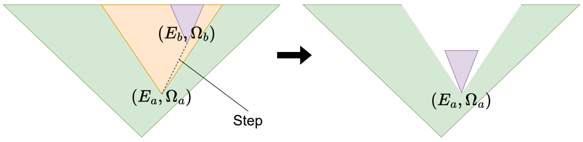
▶ **Example 31** (Splitting (E_1, Ω_1)). In example 24, we had $E_1 = \{l = Cons(l_1, l_2),\ n = S(n_1), n = S(m)\}$ and $\Omega_1 = \{\omega_1, \omega_2, \omega_3\}$ with $\omega_1 = [\langle l_1, \square \rangle : (\mathcal{A}_{\text{Len}}, q_n)]$, with $\omega_2 = [\langle l_2, n_1 \rangle : (\mathcal{A}_{\text{Len}}, q_f)]$, and $\omega_3 = [\langle m \rangle : (\mathcal{A}_{\text{Odd}}, q_e)]$. We have $\sigma_{E_1} = \{l \mapsto Cons(l_1, l_2),\ n \mapsto S(n'),\ n_1 \mapsto n',\ m \mapsto n'\}$, $V_{E_1}(\omega_1) = \{l_1\}$, $V_{E_1}(\omega_2) = \{l_2, n'\}$, and $V_{E_1}(\omega_3) = \{n'\}$. Therefore $\text{Split}(E_1, \Omega_1) = \{\{\omega_1\}, \{\omega_2, \omega_3\}\}$.
Solving ω_1 have no impact on the solving of ω_2 and ω_3 because the values that l_1 can take do not influence the values that l_2, n_1, or m_2 can take. On the other hand, because of E_1, m and n_1 must take the same value, and therefore typing obligations ω_2 and ω_3 cannot be separated. Note that applying this Split rule before the second Step (of example 24) would have separated (E_1, Ω_1) into two independent problems.

The second optimisation consists in *pruning the search tree*. The search space is, for almost all typing problems, infinite. Without pruning, it would be impossible to cover the whole search space, and therefore negative instances would (almost) all never terminate. Pruning the search tree allows, in some cases, to finitely ensure that no typing proof exists.

▶ **Definition 32** (Pruning). *Let T be a proof tree. A node $\vdash (E_b, \Omega_b)$ that appears in the sub-tree of T whose root is some other node $\vdash (E_a, \Omega_a)$ is* prunable *when both*
(i) *At least one* Step *rule is used on the path between $\vdash (E_a, \Omega_a)$ and $\vdash (E_b, \Omega_b)$;*
(ii) $\exists \sigma, \sigma(\sigma_{E_a}(\Omega_a)) \subseteq \sigma_{E_b}(\Omega_b)$.

▶ **Theorem 33** (Safety of pruning). *For any proof tree that contains a prunable node, there exist a strictly smaller (w.r.t the total number of times the* Step *rule is used) proof tree with the same root.*

The idea of pruning a proof T is to replace the orange proof sub-tree of $\vdash (E_a, \Omega_a)$ with the purple proof tree of $\vdash (E_b, \Omega_b)$ (with minor modifications).



Proof. Let T be a prunable tree, that is such that there exists nodes $\vdash (E_a, \Omega_a)$ and $\vdash (E_b, \Omega_b)$ with respective proof trees T_a and T_b, with T_b a sub-tree of T_a with a Step rule between $\vdash (E_a, \Omega_a)$ and $\vdash (E_b, \Omega_b)$, and σ a substitution such that $\sigma(\sigma_{E_a}(\Omega_a)) \subseteq \sigma_{E_b}(\Omega_b)$.
By theorem 26(A) there exists a substitution σ_b with $\sigma_b \models (E_b, \Omega_b)$ and $h(\sigma_b, \Omega_b) = h(T_b)$. Because σ_{E_b} is the most general unifier of E_b and $\sigma_b \models E_b$, there exists σ' such that $\sigma_b = \sigma_{E_b}; \sigma'$. Therefore the substitution $\sigma_a = \sigma_{E_a}; \sigma; \sigma'$ is such that $\sigma_a(\Omega_a) \subseteq \sigma_b(\Omega_b)$. Because $\sigma_b \models \Omega_b$, we also have $\sigma_a \models \Omega_a$. Because σ_a first applies σ_{E_a}, we have $\sigma_a \models E_a$. Therefore $\sigma_a \models (E_a, \Omega_a)$. Finally, again because $\sigma_a(\Omega_a) \subseteq \sigma_b(\Omega_b)$, we have $h(\sigma_a, \Omega_a) \leq h(\sigma_b, \Omega_b)$. By applying theorem 26(B) there exists a proof T'_a of $\vdash (E_a, \Omega_a)$ with $h(T'_a) = h(\sigma_a, \Omega_a) \leq h(\sigma_b, \Omega_b) = h(T_b)$.

Therefore, the proof tree T whose sub-tree T_a has been replaced by T'_a is valid and smaller. Besides, we know that the sub-tree T'_a is *strictly* smaller than T_a because T_a contains at least one application of the Step rule between its root and T_b. Therefore, this transformation strictly decreases the size of the proof tree. ◀

▶ **Corollary 34.** *By induction, if there exists a proof tree T of some initial typing problem, then there exists one without any prunable node along the proof tree, and therefore abandoning the search of prunable branches is safe.*

▶ **Example 35** (Pruning of the search tree). During the second Step application of example 24, the typing problem (E'_2, Ω'_2) is also in $unfolds(\Omega_1)$. This was no problem, as the algorithm found a solution and stopped. Now, if (for example) automaton $\mathcal{A}_{\text{Len}}$ did not have rule (D), then there would be no solution to the initial typing problem (E_0, Ω_0). The search would never stop, as, after a bit of unification and renaming, (E_0, Ω_0) can be included in $(E_1 \cup E'_2, \Omega'_2)$. Without pruning, the typing algorithm could therefore loop forever instead of returning *None*. Fortunately, $(E_1 \cup E'_2, \Omega'_2)$ can be pruned by taking $\sigma = \{l \mapsto l_{22},\ n \mapsto n_{11}\}$, as $\sigma(\sigma_0(\Omega_0)) \subseteq \sigma_2(\Omega'_2)$ (with σ_0 and σ_2 being most general unifiers of E_0 and $E_0 \cup E_1 \cup E'_2$, respectively).

7 Regular structure inference

This section presents a procedure for inferring a regular model of a set of CHCs. The input set of CHCs we later use the procedure for is $\mathcal{C} = \Gamma \cup \Gamma'$, with Γ defining a program and Γ' the desired properties. The procedure follows the Implication Counter-Example (ICE) framework [8]. In this framework, the task of inferring a correct model is divided between two entities (or procedures), a *learner* and a *teacher*, working iteratively. There are three possible outcomes for this procedure: either the learner finds a correct model (that the teacher validates), the learner finds a contradiction, or the procedure loops forever with more and more refined models.

The teacher's procedure takes as input a model $\mathcal{M}$ and a CHC system $\mathcal{C}$, and returns an optional ground Horn clause. It returns *None* if $\mathcal{M} \models \mathcal{C}$, and $\mathit{Some}(\sigma(\varphi))$ if $\mathcal{M} \not\models \varphi$ with counter-example σ for some $\varphi \in \mathcal{C}$. With the model checking procedure already defined, a teacher's implementation is only a matter of selecting an order in which to check the formulas. For example, taking as input the problem of example 18, the output would be $\mathrm{Len}(Cons(Z, Nil),\ S(Z)) \Rightarrow \mathrm{Even}(S(Z))$.

The learner's procedure is responsible for inferring a model from examples or finding a contradiction. It takes as input a finite set $\underline{\mathcal{C}}$ of ground CHCs and returns *None* if $\underline{\mathcal{C}}$ is contradictory and $\mathit{Some}(\mathcal{M})$ otherwise, with $\mathcal{M}$ being a smallest model (in the number of states) satisfying $\underline{\mathcal{C}}$. This procedure is divided into two steps, which are the main subject of this section, the *working model generation* and the *working model generalisation*.

▶ **Definition 36** (Working model generation). *The working model $\mathcal{W}$ of a given finite set of ground CHCs $\underline{\mathcal{C}}$ is the smallest model (up to state renaming) recognizing exactly the terms mentioned in $\underline{\mathcal{C}}$ in a different state for each. That is, for any atom $R(\vec{t})$ of any $\varphi \in \underline{\mathcal{C}}$, there exists a state q in $\mathcal{W}(R)$ such that $\mathcal{R}(\mathcal{W}(R), q) = \{\vec{t}\}$.*

This working model construction is carried out by classical automaton algorithms [6]. The model $\mathcal{W}$ can then be generalised by merging states and deciding which equivalence classes are to be considered as final states. Merging states leads to additional terms being recognized and makes regularity appear. We search for a merging that minimises the number of states of $\mathcal{W}$ while ensuring that the resulting model satisfies $\underline{\mathcal{C}}$.

▶ **Definition 37** (State merging problem). *The minimisation problem we define is on the first-order (functional) signature $S = \{c_q \mid \mathcal{A} \in dom(\mathcal{W}) \wedge q \in Q(\mathcal{A})\} \cup \{Final\}$ containing only constants, one for each state of every automaton in $\mathcal{W}$, and one unary predicate $Final$. The constraints are $\mathcal{C}_{ok} \cup \mathcal{C}_f$. The set $\mathcal{C}_{ok}$ represents essential constraints: (i) merged states must belong to the same automaton ; (ii) merged states must be of the same type ; (iii) any final state must be of its automaton's type. The set $\mathcal{C}_f$ forces states to be or not to be final, which also have an impact on which states to merge. It is defined from $\underline{\mathcal{C}}$ by transforming every clause $\varphi = R_1(\vec{t_1}) \wedge \ldots \wedge R_n(\vec{t_n}) \Rightarrow R_0(\vec{t_0})$ into $\varphi^q = Final(c_{q_1}) \wedge \ldots \wedge Final(c_{q_n}) \Rightarrow Final(c_{q_0})$, with each q_i being the state of $\mathcal{W}(R_i)$ that recognizes exactly $\vec{t_i}$. Recall that we use non-Horn clauses, so the head of φ could be empty or contain multiple predicates.*

A minimal solution $[\![\cdot]\!]$ to the state merging problem can be computed by a finite model finder. We write $[\![Final]\!]$ for the set of final states of the solution and $[\![c_q]\!]$ for the equivalence class of constant c_q.

▶ **Definition 38** (Generalisation of working model). *Given a solution $[\![\cdot]\!]$ to the state merging problem, we generalise the working model $\mathcal{W}$ by $\mathcal{M}$ with $\mathcal{M}(R) = (Q, Q_f, \Delta)$ with $Q = \{[\![c_q]\!] \mid q \in Q(\mathcal{W}(R))\}$, $Q_f = Q \cap [\![Final]\!]$ and $\Delta = \{\vec{f}([\![c_{q_1}]\!], \ldots, [\![c_{q_n}]\!]) \to [\![c_{q_0}]\!] \mid \vec{f}(q_1, \ldots, q_n) \to q_0 \in \Delta(\mathcal{W}(R))\}$.*

▶ **Example 39** (Learner: Model generation). We observe the ICE procedure after learner and teacher already had two exchanges to learn the Len relation defined in Section 2. The learner has accumulated the constraints $\{\text{Len}(Nil, Z), \quad \text{Len}(Nil, Z) \Rightarrow \text{Len}(Cons(Z, Nil), S(Z))\}$. The generated working model is $\mathcal{W} = \{\text{Len} \mapsto \mathcal{A}\}$ with $\mathcal{A} = (Q, Q_f, \Delta)$, $Q = \{q_{l_0}, q_{l_1}, q_n\}$, $Q_f = \emptyset$, and $\Delta = \{\langle Nil, Z\rangle() \to q_{l_0}\ ;\ \langle Cons, S\rangle(q_n, q_{l_0}) \to q_{l_1}\ ;\ \langle Z, \square\rangle() \to q_n\}$. We have $\mathcal{R}(\mathcal{A}, q_{l_0}) = \{(Nil, Z)\}$, $\mathcal{R}(\mathcal{A}, q_n) = \{(Z, \square)\}$, and $\mathcal{R}(\mathcal{A}, q_{l_1}) = \{(Cons(Z, Nil), S(Z))\}$. Note that state q_n recognizes the term $\langle Z, \square\rangle$ which does not appear in $\underline{\mathcal{C}}$ but is necessary to recognize $(Cons(Z, Nil), S(Z))$.

The minimisation problem is therefore on the signature with unary predicate $Final$ and constant symbols $c_{q_{l_0}}$, $c_{q_{l_1}}$, and c_{q_n}. The constraints $\mathcal{C}_{ok}$ are stating that q_n cannot be merged with q_{l_0} nor q_{l_1} because they are not of the same type, and that only q_{l_0} and q_{l_1} can be final, as they are the only states of the automaton's type, $natlist \times nat$. The constraints $\mathcal{C}_f$, generated from $\underline{\mathcal{C}}$, are $\{Final(c_{q_{l_0}}),\ Final(c_{q_{l_0}}) \Rightarrow Final(c_{q_{l_1}})\}$. The smallest model is a two-elements set $\{q_l, q_z\}$, with $[\![Final]\!] = \{q_l\}$, $[\![q_{l_0}]\!] = [\![q_{l_1}]\!] = q_l$, and $[\![q_n]\!] = q_z$.

The generalized model is $\mathcal{M} = \{\text{Len} \mapsto \mathcal{A}'\}$ with automaton $\mathcal{A}'$ having states $\{q_l, q_z\}$, final states $\{q_l\}$, and transitions $\{\langle Nil, Z\rangle() \to q_l,\ \langle Cons, S\rangle(q_z, q_l) \to q_l,\ \langle Z, \square\rangle() \to q_z\}$. This automaton recognizes an almost-correct relation: the set of pairs (l, n) of a list of zeros together with its size. The only missing rule is $\langle S, \square\rangle(q_z) \to q_z$, which will be added by the learner in the ICE step that follows.

8 Approximation

As we suppose programs to be deterministic and terminating, the CHC representation of a functional program has only one possible model. For many programs, this model is not regular and cannot be represented using convoluted tree automata. As a result, trying to verify a property using an exact model of the relation will fail on such programs. We circumvent this problem by approximating relations.

Our verification goals are CHCs of the form $\psi(\vec{x}) \wedge R_1(\vec{x}_1) \wedge \ldots \wedge R_n(\vec{x}_n) \Rightarrow R_0(\vec{x}_0)$. Given a relation R we denote by R^+ (resp. R^-) an over-approximation (resp. under-approximation) of R which can also be R itself. A safe way to prove the above implication using approximations is to over-approximate $R_1, \ldots, R_n$ and under-approximate R_0. If $\psi(\vec{x}) \wedge R_1^+(\vec{x}_1) \wedge \ldots \wedge R_n^+(\vec{x}_n) \Rightarrow R_0^-(\vec{x}_0)$ is true then so is the original CHC. Applying such a reasoning on the CHCs of the verification goal, we can infer which relations can be over or under-approximated. For instance, the functional program computing the sum of two natural numbers is represented by the relation $\mathrm{Plus}(n, m, u)$ associating any two natural numbers n and m with their sum u. This relation is not regular when using unary encoding of numbers. The argument for seeing this is very similar to that of $\{a^n \cdot b^n \mid n \in \mathbb{N}\}$ not being a regular string language. For the string automaton, it would require an unbounded counter for as in order to later exactly match their number with bs. For a convoluted tree automaton to recognize $\mathrm{Plus}(n, m, u)$, the counting is of the depth at which n and m root symbol stop being both S, which later needs to match the number of Ss left on u. However, to prove a property of the form $\mathrm{Plus}(n, m, u) \Rightarrow n \leq u$, we only need a regular over-approximation of the relation Plus, say Plus^+, and an under-approximation of $\leq$, say $\leq^-$, such that $\mathrm{Plus}^+(n, m, u) \Rightarrow n \leq^- u$.

In practice, we focus on over-approximation and do not under-approximate. We thus prove the stronger goal $\mathrm{Plus}^+(n, m, u) \Rightarrow n \leq u$. Here are the clauses defining the Plus relation:

$$\mathrm{Plus}(n, Z, n). \quad \mathrm{Plus}(n, m, u) \Rightarrow \mathrm{Plus}(n, S(m), S(u)). \quad \mathrm{Plus}(v, w, x) \wedge \mathrm{Plus}(v, w, y) \Rightarrow x = y.$$

These clauses form a system where the first clause invalidates under-approximations, the second clause can invalidate both over and under approximations, whereas the third only invalidates over-approximations. We can therefore obtain a safe approximation Plus^+ from Plus by simply removing the third clause. In our example, this suffices to prove $\mathrm{Plus}^+(n, m, u) \Rightarrow n \leq u$ because the approximation Plus^+ we built relates any n, m with all u greater than or equal to n (See the solver result for `isaplanner_prop21.smt2` in `http://people.irisa.fr/Thomas.Genet/AutoForestation/results_right/benchmarks.html`).

Finally, some relations cannot be approximated. If a relation appears on both sides of the verification goal then it cannot be approximated. *E.g.*, to prove $Z < m \wedge \mathrm{Plus}(n, m, u) \Rightarrow n < u$, we can safely use Plus^+. Since $<$ occurs (positively) on the left and right-hand side of the implication, we could use $<^+$ on the left-hand side and $<^-$ on the right-hand side. We could use different approximations for relations appearing at different positions in the formula. However, in our analyser, we choose to use a common approximation for any relation. In our example, we use the intersection between $<^+$ and $<^-$, which is exactly $<$.

9 Implementation and Experiments

We implemented the verification algorithm in Ocaml. It can be found on `https://gitlab.inria.fr/tlosekoo/auto-forestation`. This provides an implementation of terms, tree automata, model checking, model-inference procedure, as well as left, right, and complete convolution.

The *teacher* closely follows the depth-first search of the proof system described in section 5. There is a lot of redundancy in the proof search, so we used canonization and memoisation of typing problems. Memoisation avoids re-computing the unfolding of a typing problem if the search already did. However, memoisation alone is not very useful, as even equivalent typing problems are often different because of variable names. This is the reason for using canonization, which ensures that equivalent typing problems have the same internal representation. The *learner* delegates the merging of states to *Clingo* [9], a finite-model finder.

The solver presented in this paper builds regular relations, as opposed to [16, 11, 17] which only build regular sets of terms. Since regular sets are a particular case of regular relations, our solver should be able to handle the examples covered by those techniques, plus some relational problems. As a result, for the experiments, we choose some examples coming from benchmarks of Timbuk [11], add relational examples taken from the Isaplanner benchmark [7, 4] and built relational problems inspired by TIP [5, 4]. As shown in Section 2, a typical property which can be automatically proved by those non-relational solvers [16, 11, 17] is of the form $\forall x\ l.\ less\ Z\ (len\ (Cons(x, l)))$ where l is any list of natural numbers.

Non-relational solvers can also handle a restricted form of relations: the finite union of languages $\mathcal{L}_1 \times \ldots \times \mathcal{L}_n$ where $\forall i \in [\![1, n]\!]$, $\mathcal{L}_i$ is a regular language. This allows to prove properties with a limited form of relation. For instance, using a non-relational regular solver, it is possible to prove the property $\forall l_1\ l_2.\ less\ Z\ (len\ l_1) \Rightarrow\ less\ Z\ (len\ (append\ l_1\ l_2))$ where *append* is the function concatenating lists and l_1 and l_2 are lists of a. For the tuple of variables (l_1, l_2) to cover all the possible cases, it is enough to consider the two languages $\mathcal{L}_{nil} \times \mathcal{L}_{lists}$ and $\mathcal{L}_{Cons+} \times \mathcal{L}_{lists}$ where $\mathcal{L}_{nil} = \{Nil\}$ and $\mathcal{L}_{Cons+} = \mathcal{L}_{lists} \setminus \mathcal{L}_{nil}$. With the first language, the property is true because the left-hand side of the implication is false. With the second language $\mathcal{L}_{Cons+} \times \mathcal{L}_{list}$, both the left and right-hand side of the implication are true.

One of the simplest problem which cannot be proved using a non-relational "regular" solver is $\forall x\ y.\ Cons(x, y) \neq y$. Proving such a property cannot be done using a finite union of products of regular languages. However, this property can automatically be proven using our relational solver. Additionally to the above examples, we highlight some relational properties which are automatically proven using our solver.

- $\forall (l : ablist).\ (len\ l) = (len\ (reverse\ l))$ `length_reverse_eq.smt2`
- $\forall (l_1 : ablist)\ (l_2 : ablist).\ (prefix\ l_1\ (append\ l_1\ l_2))$ `prefix_append.smt2`
- $\forall (l : ablist).\ (len\ l) = (len\ (sort\ l))$ `sort_length_eq.smt2`
- $\forall (i : nat)(t_1 : natbintree)(t_2 : natbintree).\ t_1 \neq (node\ i\ t_1\ t_2)$ `tree_add_not_eq.smt2`

On the following properties our solver is able to find a counter-example.

- $\forall (n : nat).\ n < (double\ n)$ `nat_double_is_le.smt2`
- $\forall (x : ab)\ (l : ablist).\ (delete\_one\ x\ l) = (delete\_all\ x\ l)$ `list_delete_all_count.smt2`
  $\Rightarrow (count\ x\ l) = 1$

On the following properties, our solver does not terminate due to trying to represent a non-regular relation (ICE loops).

- $\forall (x : ab)\ (l : ablist).\ (delete\_one\ x\ l) = (delete\_all\ x\ l)$ `list_delete_all_count.smt2`
  $\Rightarrow (count\ x\ l) \leq 1$
- $\forall (n : nat)\ (m : nat).\ n + m = m + n$ `plus_commutative.smt2`

All of our experimental results for all convolution types are available at `http://people.irisa.fr/Thomas.Genet/AutoForestation/`. Because the properties of our database were mostly either on same-type relations or on lists and natural numbers, the right-convolution was the most efficient of convolution type. Left-convolution is not adapted for most of the list-based examples and complete-convolution revealed to be too costly in practice though it should help to prove properties on functions manipulating trees. On a total of 120 examples, our solver (using right-convolution) proves 66, disproves 23, and timeouts on 31 after 60s. Our solver succeeds on 20 out of the 79 first-order Isaplanner examples in less than 60s (and 18 in less than 5s). Our solver reveals to be more efficient on examples where a single level of structure have to be compared, i.e., natural numbers, lists of arbitrary elements, etc. It is generally unsuccessful on examples mixing several layers of structure, e.g., lists of natural numbers, or on examples where a precise counting is required to prove the property. Finally, on examples where using a non-relational model suffices to prove the property, our solving technique is flexible enough to find such a model, with an efficiency comparable with non-relational solvers. For instance, on 11 examples coming from the Timbuk benchmarks, we proved 6 of them (with execution times around 2 seconds), disproved 3, and have a timeout on the 2 last.

10 Related work

Other approaches for automatically proving algebraic and relational properties also rely on a CHC representation. The approach of [20] and [19] aims to solve the satisfiability problem of Horn clauses over any underlying theory supported by an SMT solver. This approach first reduces this problem to validity checking of first-order formulas with inductively-defined predicates. It is then based on syntactic proof, together with calls to the underlying theory solver. They design an inductive proof system tailored to Horn constraint solving. Using the theory of inductive datatypes, their method can reason about, and automatically prove, relational and algebraic properties.

Another approach, which is closer to ours, is that of [18]. This approach aims to check properties on recursive data-structure by using *symbolic automatic relations*, which are (almost) the languages defined by *symbolic synchronous automata* (ss-NFA), the combination of symbolic automata and automatic relations. They devise a sound but (necessarily) incomplete procedure for checking if a given formula admits an assignment of its free variables that makes it true in a given ss-NFA. This procedure corresponds to the *teacher* procedure, but for ss-NFAs. They plan to implement an ICE-based CHC solver, but have left the model discovery (*learner* section) to future work.

By manually writing ss-NFAs, authors of [18] are able to benchmark their verification procedure. Our approach and theirs seems to be complementary as they succeed on different sets of examples. This can be observed on the IsaPlanner benchmark where our technique fails on most of examples that [18] handles (i.e. 4, 5, 15, 16, 29, 30, 39, 42, 50, 62, 67, 71, 86) and succeeds on examples on which they do not report any success (i.e. 17, 18, 21, 22, 23, 24, 25, 26, 31, 32, 33, 34, 45, 46, 65, 69).

In [3], the authors present an expressive formalism for representing relations between trees called *synchronized context-free programs*. This formalism is more expressive than convoluted tree automata presented here. In particular, it can represent languages of the form $\{(g^n(a), g^n(b)) \mid n \in \mathbb{N}\}$ (like convoluted tree automata) and also languages of the form $\{f(g^n(a), g^n(b)) \mid n \in \mathbb{N}\}$ and $\{g^n(h(g^n(a))) \mid n \in \mathbb{N}\}$ (out of the scope of convoluted tree automata). This formalism is used to precisely approximate the set of outputs of a term rewriting system. However, [3] does not show how to automatically infer such a representation from the term rewriting system.

11 Conclusion and future work

This paper demonstrates that it is possible to use tree automata as a basis for analysing the input-output behaviour of a first-order functional program. This shows that existing automata-based techniques for approximating the set of reachable states of a function can be extended to also compute relations between input and output of a function. Such relational analysis is key to scaling static analyses to larger programs, because it enables a modular, function-by-function analysis technique. The extension to relational analysis is based on the notion of tree automata convolution. We argue that the standard left-convolution can be complemented by other convolution techniques in order to verify more properties of programs. Another technical contribution of the paper is the proof tree pruning used for verifying models of constrained Horn clauses. An efficient implementation of this proof search has been an essential part of the counter-example guided learner-teacher algorithm for inferring models from the CHC representation of the program to be analysed. This is confirmed by the benchmark used to evaluate our implementation of the verifier.

We believe our ICE procedure to be *relatively refutationally complete* and *relatively complete on regular structures*. *Relative* means that we suppose the termination of the model-checking procedure to be able to study the ICE cycle. *Refutationally complete* means that if the set of clauses $\mathcal{C}$ given to the ICE procedure is contradictory, then the procedure eventually finds a contradiction and stops. *Complete on regular structures* means that if the set of clauses $\mathcal{C}$ given to the ICE procedure admits a regular model, then the procedure eventually finds a model of $\mathcal{C}$. This has to be investigated further.

Fixing the convolution to be the either left or right convolution is however insufficient for proving non-trivial properties that would need a different overlay of terms, for example the *height* function on trees. Complete convolution can theoretically overcome this restriction but, as confirmed by our benchmarks, the size explosion of convoluted term makes it unusable in practice. We believe the convolution can and should be non-static, that is, being inferred together with the model.

Moreover, unlike the convolutions presented in this paper, we think that convolution could be lossy. For instance, if a subterm in a relation is not useful to prove a property, we think that we can forget about it in the convolution. Later on, if a new ground counter-example comes to the learner showing that the subterm was, in fact, necessary to prove the property then the convolution needs to be extended for that purpose.

References

1 Clark Barrett, Igor Shikanian, and Cesare Tinelli. An abstract decision procedure for a theory of inductive data types. *Journal on Satisfiability, Boolean Modeling and Computation*, 3(1-2):21–46, 2007.

2 Nikolaj Bjørner, Arie Gurfinkel, Ken McMillan, and Andrey Rybalchenko. Horn clause solvers for program verification. In *Fields of Logic and Computation II*, pages 24–51. Springer, 2015.

3 Yohan Boichut, Jacques Chabin, and Pierre Réty. Towards more precise rewriting approximations. *J. Comput. Syst. Sci.*, 104:131–148, 2019.

4 Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. Tip and isaplanner benchmarks, 2015. URL: `https://tip-org.github.io/`.

5 Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. Tip: tons of inductive problems. In *International Conference on Intelligent Computer Mathematics*, pages 333–337. Springer, 2015.

6 Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications.* 2008. URL: `https://hal.inria.fr/hal-03367725`.

7 Lucas Dixon and Jacques Fleuriot. Isaplanner: A prototype proof planner in isabelle. In *CADE'03*, volume 2741, pages 279–283. Springer, 2003.

8 Pranav Garg, Christof Löding, P Madhusudan, and Daniel Neider. Ice: A robust framework for learning invariants. In *International Conference on Computer Aided Verification*, pages 69–87. Springer, 2014.

9 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

10 Erich Grädel. Automatic structures: twenty years later. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 21–34, 2020.

11 Timothée Haudebourg, Thomas Genet, and Thomas Jensen. Regular Language Type Inference with Term Rewriting. *Proceedings of the ACM on Programming Languages*, 4(ICFP):1–29, 2020.

12 Timothée Haudebourg. *Automatic Verification of Higher-Order Functional Programs using Regular Tree Languages.* PhD thesis, Univ. Rennes1, 2020.

13 Tirza Hirst and David Harel. More about recursive structures: Descriptive complexity and zero-one laws. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, pages 334–347. IEEE, 1996.

14 R Hodgson Bernard. *Théories décidables par automate fini (Decidable theories via finite automata).* PhD thesis, Ph.D. thesis Département de Mathématiques et de Statistique, Université de . . . , 1976.

15 Bakhadyr Khoussainov and Anil Nerode. Automatic Presentations of Structures. In *International Workshop on Logic and Computational Complexity*, pages 367–392. Springer, 1994.

16 Yurii Kostyukov, Dmitry Mordvinov, and Grigory Fedyukovich. Beyond the elementary representations of program invariants over algebraic data types. In Stephen N. Freund and Eran Yahav, editors, *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, pages 451–465. ACM, 2021.

17 Yuma Matsumoto, Naoki Kobayashi, and Hiroshi Unno. Automata-based abstraction for automated verification of higher-order tree-processing programs. In *Asian Symposium on Programming Languages and Systems*, pages 295–312. Springer, 2015.

18 Takumi Shimoda, Naoki Kobayashi, Ken Sakayori, and Ryosuke Sato. Symbolic automatic relations and their applications to SMT and CHC solving. In *International Static Analysis Symposium*, pages 405–428. Springer, 2021.

19 Takeshi Tsukada and Hiroshi Unno. Software model-checking as cyclic-proof search. *Proceedings of the ACM on Programming Languages*, 6(POPL):1–29, 2022.

20 Hiroshi Unno, Sho Torii, and Hiroki Sakamoto. Automating induction for solving horn clauses. In *International Conference on Computer Aided Verification*, pages 571–591. Springer, 2017.

A Appendix

Proof of theorem 26

Proof of A. Let suppose that T proves $\vdash (E, \Omega)$ and $h(T) = n$. Let us proceed by induction on the last rule used in T.

- case CONCLUDE:
  By hypothesis, we have that T is of the form $\dfrac{}{\vdash (E, \emptyset)}$ with $Coherent(E)$, and therefore $n = 0$. Take $\sigma = \sigma_E$ a most general unifier of E, which is well-defined, as E is coherent. We have: (i) $\sigma \models E$ is immediate, as σ unifies E; (ii) $\sigma \models \Omega$ is trivial, as $\Omega = \emptyset$; (iii) $h(\Omega, \sigma) = 0 = n$, as Ω is empty.

- case STEP:
  By hypothesis, we have that T is of the form $\textsc{Step}\,\dfrac{T'}{\vdash (E, \Omega)}$ with T' of the form $\dfrac{\dots}{\vdash (E \cup E', \Omega')}$ and $(E', \Omega') \in unfolds(\Omega)$. By induction, we have that there exists σ' with $\sigma' \models (E \cup E', \Omega')$ and $h(\Omega', \sigma') = h(T')$. We also know that $h(T') = n - 1$. Take $\sigma = \sigma'$. Then:
  - $\boldsymbol{\sigma \models E}$: Immediate by $\sigma' \models E \cup E'$ and monotonicity of first-order logic.
  - $\boldsymbol{\sigma \models \Omega}$: Let $\omega = [\vec{x} : (\mathcal{A}, q)] \in \Omega$. We must prove that $\sigma(\vec{x}) \in \mathcal{R}(\mathcal{A}, q)$. For this, it is sufficient (and necessary) to show that there exists a rule $r = \vec{f}(\vec{q}) \to q$ of $\mathcal{A}$ such that
    * $\forall i \in [\![1, |\vec{f}|]\!], \sigma(x_i) = f_i(\vec{y_i})$ for some variables $\vec{y_i}$;
    * $\forall j \in [\![1, |\vec{q}|]\!], \sigma \models [\bigcirc(\vec{y}_1, \dots, \vec{y}_{|\vec{f}|})[j] : (\mathcal{A}, q_j)]$.

    Since $(E', \Omega') \in unfolds(\Omega)$, we know that there exists such a rule r with $(E_r, \Omega_r) \in unfold(\omega)$. The first property is immediate from $\sigma \models E'$ and $E_r \subseteq E'$ while the second is immediate from $\sigma \models \Omega'$ and $\Omega_r \subseteq \Omega'$.
  - $\boldsymbol{h(\Omega, \sigma) = n}$: Because $(E', \Omega') \in unfolds(\Omega)$, every variable y in Ω' is such that there exists a variable x in Ω with $\sigma(x) = f(\dots, \sigma(y), \dots)$ for some function f, that is, $h(\sigma, \Omega') < h(\sigma, \Omega)$. Moreover, every variable x in Ω with $h(\sigma(x)) > 1$ yields a least one variable y in Ω' with $h(\sigma(y)) = h(\sigma(x)) - 1$.
    Therefore, $h(\sigma, \Omega) = h(\sigma, \Omega') + 1 = h(T') + 1 = n$. ◀

Proof of B.

Let us build a proof tree by induction on $h(\Omega, \sigma)$.

In any case, let suppose that there exists σ such that $\sigma \models (E, \Omega)$ and $h(\Omega, \sigma) = n$. We then construct a proof tree T of $\vdash (E, \Omega)$ such that $h(T) = n$.

- case $h(\Omega, \sigma) = 0$: This is only possible when $\Omega = \emptyset$. Take $T = \textsc{conclude}\,\dfrac{}{\vdash (E, \Omega)}$. This proof tree T is correct, as $\Omega = \emptyset$ and E is coherent (because $\sigma \models E$). Also $h(T) = 0$.
- case $h(\Omega, \sigma) > 0$:
  Because $\sigma \models \Omega$, we have, for each $\omega = [\langle x_1, \dots, x_n \rangle : (\mathcal{A}, q)] \in \Omega$, that there exists an associated rule $r_\omega = \langle f_1, \dots, f_n \rangle(q_1, \dots, q_k) \to q$ such that
  - $\forall i \in [\![1, n]\!]$, $\sigma(x_i) = f_i(\vec{t_i})$ for some terms $\vec{t_i}$;
  - $\forall j \in [\![1, k]\!], \bigcirc(\vec{t}_1, \dots, \vec{t}_n)[j] \in \mathcal{R}(\mathcal{A}, q_j)$.

  Therefore we can build three functions, F^c, F^t, F^s, which assign to each such typing obligation and rule the following:
  - $F^c(\omega) = \{x_1 = f_1(\vec{x}_1), \dots, x_n = f_n(\vec{x}_n)\}$, with $\forall i \in [\![1, n]\!], \vec{x}_i$ are fresh variables.
  - $F^t(\omega) = \{[\bigcirc(\vec{x}_1, \dots, \vec{x}_n)[j] : (\mathcal{A}, q_j)] \mid j \in [\![1, k]\!]\}$
  - $F^s(\omega) = \{(x_i^j, t_i^j) \mid x_i = f_i(x_i^1, \dots, x_i^m) \in F^c(\omega) \wedge j \in [\![1, m]\!] \wedge \sigma(x_i) = f(t_i^1, \dots, t_i^m)\}$

Let $E' = \bigcup_{\omega \in \Omega} F^c(\omega)$ and $\Omega' = \bigcup_{\omega \in \Omega} F^t(\omega)$. Note that $(E', \Omega') \in unfolds(\Omega)$.
Let $\sigma' = \sigma \cup \bigcup_{\omega \in \Omega} F^s(\omega)$. We have:

- σ' is well-defined: Any binding of σ' which is not in σ is of the form $x_i^j = \sigma(t_i^j)$ for some fresh variable x_i^j. Therefore, as σ is well-defined, so is σ'.
- $\sigma' \models E \cup E'$: We have $\sigma \subseteq \sigma'$, therefore $\sigma' \models E$. Any constraint of E' is of the form $x_i = f_i(\vec{x}_i)$ with x_i a variable appearing in a node $\omega \in \Omega$, for which we therefore have $\sigma'(x_i) = f_i(\sigma'(\vec{x}_i)) = \sigma'(f_i(\vec{x}_i))$ by definition of $F^s(\omega)$.
- $\sigma' \models \Omega'$: For any typing obligation $\omega' \in \Omega'$, we have $\omega' \in F^t(\omega)$ for some $\omega \in \Omega$, so $\omega' = [\langle x_1, \ldots, x_n \rangle : (\mathcal{A}, q_j)]$ for some $x_1, \ldots, x_n$ such that $\langle \sigma'(x_1), \ldots, \sigma'(x_n) \rangle \in \mathcal{R}(\mathcal{A}, q_j)$, by definition of $F^t(\omega)$ and $F^s(\omega)$.
- $h(\Omega', \sigma') = h(\Omega, \sigma) - 1$: For this case, let $\omega = argmax_{\omega \in \Omega}(h(\sigma, \omega))$ and $\omega' = argmax_{\omega' \in \Omega'}(h(\sigma', \omega'))$. By definition of $F^t(\omega)$ and $F^s(\omega)$, we have both $h(\sigma', \Omega') \geq h(\sigma, \omega) - 1$ and $h(\sigma', \Omega') \leq h(\sigma, \omega) - 1$.

By induction on $\sigma' \models (E \cup E', \Omega')$, we have that there exists a proof tree T' of $\vdash (E \cup E', \Omega')$ such that $h(T') = h(\sigma', \Omega')$.

Therefore, take $T = \textsc{Step}\,\dfrac{T'}{\vdash (E, \Omega)}$

We have that T is a valid proof tree and that $h(T) = h(T') + 1 = h(\Omega, \sigma)$. ◀

The Sum-Product Algorithm For Quantitative Multiplicative Linear Logic

Thomas Ehrhard
Université de Paris Cité, CNRS, IRIF, F-75013, Paris, France

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

Michele Pagani
Université de Paris Cité, IRIF, F-75013, Paris, France

Abstract

We consider an extension of multiplicative linear logic which encompasses bayesian networks and expresses samples sharing and marginalisation with the polarised rules of contraction and weakening. We introduce the necessary formalism to import exact inference algorithms from bayesian networks, giving the sum-product algorithm as an example of calculating the weighted relational semantics of a multiplicative proof-net improving runtime performance by storing intermediate results.

2012 ACM Subject Classification Theory of computation → Linear logic; Theory of computation → Denotational semantics; Mathematics of computing → Variable elimination

Keywords and phrases Linear Logic, Proof-Nets, Denotational Semantics, Probabilistic Programming

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.8

Funding This work was partially supported by ANR PRC project PPS ANR-19-CE48-0014.

1 Introduction

Linear logic [18] provides a linear algebra flavour to logic, associating linear algebra operations with logical connectives, e.g. tensor $\otimes$ is seen as a form of conjunction, direct sum $\oplus$ as a disjunction and duality as an involutive negation $(\cdot)^{\perp}$. This perspective has given many insights. In denotational semantics, we have *quantitative semantics*, e.g. [25, 21, 10, 11, 6, 24]: a family of models denoting λ-terms and functional programs with some notion of analytic maps or power series that can be locally approximated by multilinear functions, these latter denoting linear logic proofs. In proof-theory, we have *proof-nets*: a representation of proofs and programs expressing the interdependences of these algebraic operations in a graph-theoretical fashion.

Quantitative semantics turns out to be particularly suitable for probabilistic programming, giving fully abstract semantics [14, 15, 17], denoting probabilistic programs with very regular functions (absolutely monotone) even on "continuous" datatypes (e.g. real numbers) [16, 5, 13], giving a compositional analysis of various operational behaviours, such as runtime or liveness [24], providing suitable notions of program metrics [12]. Due to this expressivity, calculating the quantitative denotations for a Turing complete programming language is obviously non-computable, but we can fix on relevant fragments supporting an effective procedure. Effectiveness is a relevant feature for a denotational model, as it can provide automatic tools for verifying program correctness, as well as the other mentioned operational properties.

Let us focus our attention to one of the simplest fragments of linear logic: the multiplicative fragment (MLL), which has the $\otimes$ conjunction, its unit 1 and their respective duals, the par $⅋$ (a disjunction different from $\oplus$) and $\perp = 1^{\perp}$. From a programming perspective, this fragment contains (although it is not restricted to) an exponential-free fragment of the linear λ-calculus

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 8; pp. 8:1–8:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

with tuples, e.g. [1]: the linear functional type $F \multimap G$ is in fact represented by $F^\perp \parr G$. Although very simple, this fragment is already surprisingly expressive on probabilistic data. First, positive types (i.e. combinations of $\mathbf{1}$, $\oplus$ and $\otimes$) express linear combinations of the values of a finite data-type. For example, the quantitative denotation of $\mathbf{1} \oplus \mathbf{1}$ contains linear combinations of booleans and can be used to model boolean random variables[1]. Moreover, it is known since the inception of polarised linear logic that positive formulas are endowed with a polarised version of the structural rules of weakening and contraction ([19] and Remark 3), so one can represent λ-terms having multiple occurrences of a same boolean variable without breaking the linearity features of MLL. In probabilistic programming, these occurrences duplicate the *samples* from a random variable, but not the random variable itself. Finally, we can allow semantical boxes expressing matrices indexed by finite data-types, which can express conditional probabilities. We call this system *quantitative* MLL (Section 2).

As for the semantics, let us focus on the $\mathbb{R}_{\geq 0}$*-weighted relation semantics* (see Section 3 and [24]), which is one of the most basic examples of quantitative semantics, allowing to model probabilistic programs over countable data-types. The denotation of a proof-net is then a vector of dimension equal to the number of the possible samples of a probabilistic distribution computed by the proof-net. This vector is computable for quantitative MLL and the standard semantical definitions yield a recursive procedure (Figure 1c) to compute it. In practice, this procedure is unfeasible, as it is exponential in time and in space with respect to the size of the proof-net. The goal of this paper is to inaugurate a new approach for improving it by taking inspiration from bayesian networks, which have partially a similar graph-theoretical structure as proof-nets.

For example, the $\mathbb{R}_{\geq 0}$-weighted denotation of a proof-net describing a probabilistic distribution over a tuple of n booleans is a vector of dimension 2^n (the number of the possible outcomes of a random variable over n booleans), independently whether the values of some of these booleans depend each other or not (Example 9). The proof-net carries very clearly these interdependences via paths over boolean edges: may we reduce the dimension of its denotation by following such a structure? On a different note, the composition of two proof-nets on a tuple of n booleans yields a sum of 2^n terms (Example 11). However, this composition can be ordered by following the switching paths over the corresponding cuts. May we refactor the sum according to this order and gain in efficiency by memorising some intermediate factors?

Similar questions are typical of the research on Bayesian networks ([27], see as reference [8]), these latter being directed graphs expressing the conditional dependences between different random variables. The benefit of this approach is to provide a battery of algorithms computing, e.g., marginal distributions in a quite efficient way by taking advantage of the graph-theoretical structure of a network. Our general goal is to inaugurate a new approach to quantitative semantics which pays attention to the cost of computing the semantics, and we do so by exploiting techniques form Bayesian inference. One paradigmatic example is the *sum-product variable elimination algorithm* [29]: we propose here a formalism for computing the semantics of a quantitative MLL proof-net by adapting this algorithm (here Algorithm 1).

[1] It is known that the space of random variables ranging over a *finite* set of outcomes of cardinality n can be described by the finite additive disjunction $\bigoplus_{i \leq n} \mathbf{1}$ of the tensor unit, see e.g. [17]. This formula is not in MLL, as $\oplus$ is not a multiplicative connective, but it appears in our setting as these spaces of finite random variables are associated with the positive atomic formulas of MLL (see Example 7).

Related works. Bayesian networks form, mutatis mutandis, a strict subset of quantitative MLL proof-nets (Remark 1), morally the set of those proof-nets which do not contain formulas alternating polarities, e.g. alternation of $\otimes$ and $\parr$ connectives. This correspondence has been already acknowledged, with a slight different terminology, by the recent literature about the semantical foundations of Bayesian programming. We mention in particular [3, 22] which represent Bayesian networks as string diagrams and analyse the notion of disintegration. The paper [26] proposes a game semantics based on event structures for a variant of the linear λ-calculus underlined by quantitative MLL. The paper [28] studies an equational theory and provides a denotational semantics based on matrices for this calculus when restricted to ground data-types. However, to our knowledge, our paper is the first time that the efficiency of computing the semantics is taken into consideration. Moreover, we show that the techniques of Bayesian networks can be adapted to the more general framework of quantitative MLL without so much effort.

Paper outline. Section 2 introduces quantitative MLL proof-nets and Section 3 its associated $\mathbb{R}_{\geq 0}$-weighted relational semantics. Section 4 revisits the standard notion of factor in Bayesian inference so to apply it to atomic proof-nets in Section 5 and to general proof-nets in Section 6. Section 7 concludes by mentioning some future developments.

2 Quantitative Multiplicative Linear Logic

Metavariables X, Y, Z will vary over a countable set of propositional variables. The grammar of the formulas of MLL is given by (together with its metavariables):

$$F, G, H ::= X^+ \mid X^- \mid 1 \mid \perp \mid F \otimes G \mid F \parr G. \tag{1}$$

We call X^+ (resp. X^-) a *positive atomic formula* (resp. *negative atomic formula*) over the variable X, the superscript symbol $+$ (resp. $-$) being its *polarity*. We will write X° for a generic atomic formula over X, if we do not want to precise its polarity. The linear logic negation is introduced as syntactical sugar: $(X^+)^\perp ::= X^-$, $1^\perp ::= \perp$, $(F \otimes G)^\perp ::= F^\perp \parr G^\perp$, and for the dual cases $(X^-, \perp, \parr)$, $(F^\perp)^\perp ::= F$.

A *sequent* is a finite sequence $F_1, \dots, F_n$ of MLL formulas. Capital Greek letters $\Gamma, \Delta, \dots$ will vary over sequents. Given a sequent $\Gamma = F_1, \dots, F_n$, we write $\Gamma^\perp$ for the sequent $F_1^\perp, \dots, F_n^\perp$. Moreover, if $n > 0$, we write $\parr\Gamma$ (resp. $\otimes\Gamma$) for the formula $F_1 \parr (\cdots \parr F_n)$ (resp. $F_1 \otimes (\cdots \otimes F_n)$). If Γ is empty (i.e. $n = 0$), $\parr\Gamma$ (resp. $\otimes\Gamma$) will mean $\perp$ (resp. 1).

As accustomed in linear logic, sequent proofs are represented by special graphs, called *proof-nets*. Figure 1e gives an example of two proof-nets: $\mathcal{N}$ at the left side of the arrow $\xrightarrow{*}$, and $\mathcal{N}_0$ at the right side. A proof-net is a labelled directed acyclic graph[2] (DAG for short) such that the edges are labelled by MLL formulas and the nodes by deduction rules of our extended MLL, i.e. by a symbol among: ax (axiom), cut (cut), 1 (one), $\otimes$ (tensor), $\perp$ (bottom), $\parr$ (par), w (weakening), c (contraction), b (semantical box or simply box). The nodes of the proof-nets in Figure 1e are represented just by their labels, except for the box which is depicted as a rectangular and labeled by an enumerated occurrence of b. The label of a node determines the number of incoming edges (called *premises* of the node) and the

[2] More formally, a directed graph is a quadruplet $(V, E, \mathtt{t}, \mathtt{s})$ of a set V of vertices and a set E of edges, and two maps $\mathtt{t}, \mathtt{s} : E \mapsto V$ associating an edge with a target and a source, respectively. We alllow directed graphs with pending edges, i.e. t and s may be partial partial. The edges not in the domain of t or s are called *pending*. A directed graph is acyclic (a DAG for short), if there is no directed cycle.

number of outgoing edges (called *conclusions* of the node), as well as the type of formulas labelling these edges, according to the rules sketched in Figure 1a. The edges will be oriented top-bottom, so that axioms, ones, bottoms, weakenings and boxes have no premises, while cuts have no conclusions. Figure 1e does not explicit all formulas labelling the edges of $\mathcal{N}$ and $\mathcal{N}_0$, in fact these formulas can be recovered by the axioms and boxes labelling and the rules sketched in Figure 1a. Proof-nets have edges without targets which are called *the conclusions of the proof-net.* Both $\mathcal{N}$ and $\mathcal{N}_0$ have one single conclusion, labelled by $X_4^+ \otimes X_5^+$.

Not all DAGs of MLL nodes are proof-nets: the set of *proof-nets* is the subset of the set of all DAGs which can be generated inductively by the rules sketched in Figure 1b. We call *atomic* a proof-net whose edges are only labelled with atomic formulas. Notice that atomic proof-nets can contain only axioms, cuts, weakening, contractions and semantical boxes.

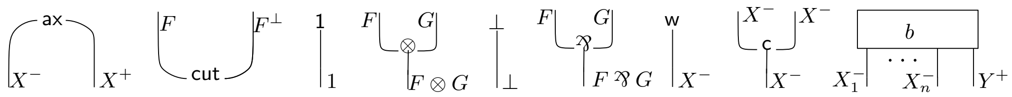
▶ **Example 1.** The (atomic) proof-net $\mathcal{N}_0$ in Figure 1e is mutatis mutandis an example of a Bayesian network as expressed by quantitative MLL. The propositional variables $X_1, \dots, X_5$ are place-holders for (sets of the possible outcomes of) random variables and the semantical boxes are place-holders for their associated "conditional probabilistic tables" (we borrow here the terminology of [8]). For example, the box b_4 is a place-holder for a probabilistic distribution over the variable X_4 conditioned by the outcomes of the variables X_2 and X_3. The polarities discriminate between input and output occurrences in a conditional probabilistic table. These place-holders will be instantiated with concrete conditional distributions by the semantics, as detailed in Section 3.

The acquainted reader in Bayesian graphs should be convinced that these latter are depicted plainly in this syntax just by adding cuts transforming outputs into inputs. Notice that by inverting the orientation of the edges labelled by negative atoms, we get exactly the same directed paths between the nodes of the corresponding Bayesian network. Of course, MLL allows for more nets than Bayesian graphs, for example the proof-net $\mathcal{N}$ at left of the $\xrightarrow{*}$ arrow is not bayesian, namely it has par nodes. But yet, Remark 54 will allude to a correspondence between $\mathcal{N}$ and a run of the sum-product algorithm over $\mathcal{N}_0$. Our goal is to show how Bayesian graph algorithms can be imported in this more general setting.

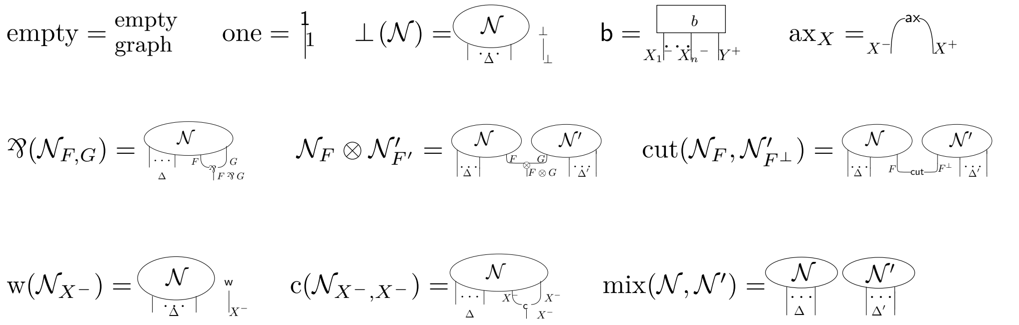
▶ Remark 2. Some papers, e.g. [3, 22], represent Bayesian graphs as string diagrams, which is a graphical syntax omitting the axiom and cut nodes. Although one can present MLL in a similar way by using Lafont's interaction nets [23], we prefer to keep axioms and cuts explicit as they condense the main threats to an efficient computation of the semantics which is a core topic of this paper.

▶ Remark 3. We allow for structural rules (weakening and contraction) on negative atomic formulas. In fact, as it will be clear in Section 3, negative atoms will be interpreted by finite products of bottoms $\binampersand_{x \in S} \bot$ (although we do not detail here the additive connectives & and $\oplus$ and the exponential modalities ? and !). It is well-known since the inception of polarized linear logic [19] that $\bot$ is isomorphic to the exponential formula ?0, so that the structural rules of ? can be lifted to $\binampersand_{x \in S} \bot$, extending the expressivity of MLL. One might even allow the structural rules to all formulas of negative polarity, but we preferred to restrict to atomic formulas to ease the presentation, namely cut reduction.

▶ Remark 4. A less standard extension is given by the semantical boxes b, which are place holders for conditional distributions or, more generally, matrices. For technical convenience, we restrict their conclusions (as well as those of MLL axioms) to be atomic formulas with exactly one occurrence of a positive formula. The structural rules of contractions and weakenings take then a precise operational meaning: a cut between the positive conclusion of a box and a contraction duplicates the *samples* of the probabilistic distribution associated with



(a) Labelling of MLL nodes with their incident edges. Edges are oriented top-down.


empty = empty graph
one =
$\perp(\mathcal{N})$ =
b =
ax_X =
$⅋(\mathcal{N}_{F,G})$ =
$\mathcal{N}_F \otimes \mathcal{N}'_{F'}$ =
$\mathrm{cut}(\mathcal{N}_F, \mathcal{N}'_{F^\perp})$ =
$\mathrm{w}(\mathcal{N}_{X^-})$ =
$\mathrm{c}(\mathcal{N}_{X^-,X^-})$ =
$\mathrm{mix}(\mathcal{N}, \mathcal{N}')$ =
$\mathcal{N}$
$\mathcal{N}'$
Δ
Δ'

(b) Sequent rules generating the set of proof-nets. The notation $\mathcal{N}_\Gamma$ in the subscript of a rule stands for the pair of a proof-net $\mathcal{N}$ and a sequence Γ of conclusions of $\mathcal{N}$, which will be "active" in the rule.

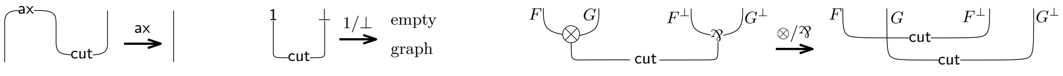
$$\llbracket \mathrm{empty} \rrbracket_\star = \llbracket \mathrm{one} \rrbracket_\star = 1 \qquad \llbracket \perp(\mathcal{N}) \rrbracket_{(\vec{d},\star)} = \llbracket \mathcal{N} \rrbracket_{\vec{d}} \qquad \llbracket \mathsf{b} \rrbracket_{(\vec{x},y)} = \iota(\mathsf{b})_{(\vec{x},y)} \qquad \llbracket \mathrm{ax}_X \rrbracket_{x,x'} = \delta_{x,x'}$$

$$\llbracket ⅋(\mathcal{N}) \rrbracket_{(\vec{d},(x,y))} = \llbracket \mathcal{N} \rrbracket_{(\vec{d},x,y)} \qquad \llbracket \mathcal{N} \otimes \mathcal{N}' \rrbracket_{(\vec{d},\vec{d}',(x,y))} = \llbracket \mathcal{N} \rrbracket_{(\vec{d},x)} \llbracket \mathcal{N}' \rrbracket_{(\vec{d}',y)}$$

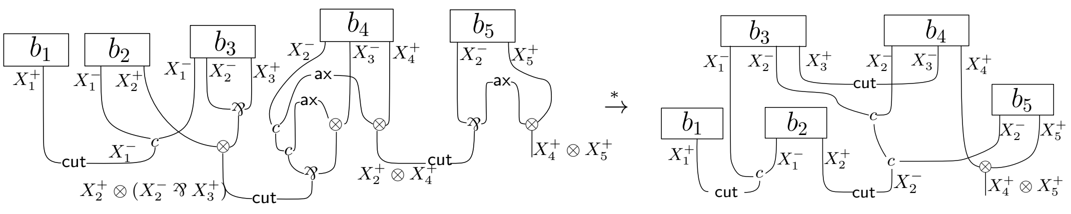
$$\llbracket \mathrm{cut}(\mathcal{N},\mathcal{N}') \rrbracket_{\vec{d},\vec{d}'} = \textstyle\sum_{x \in \llbracket F \rrbracket} \llbracket \mathcal{N} \rrbracket_{\vec{d},x} \llbracket \mathcal{N}' \rrbracket_{\vec{d}',x}$$

$$\llbracket \mathrm{w}(\mathcal{N}) \rrbracket_{(\vec{d},x)} = \llbracket \mathcal{N} \rrbracket_{\vec{d}} \qquad \llbracket \mathrm{c}(\mathcal{N}) \rrbracket_{(\vec{d},x)} = \llbracket \mathcal{N} \rrbracket_{(\vec{d},x,x)} \qquad \llbracket \mathrm{mix}(\mathcal{N},\mathcal{N}') \rrbracket_{(\vec{d},\vec{d}')} = \llbracket \mathcal{N} \rrbracket_{\vec{d}} \llbracket \mathcal{N}' \rrbracket_{\vec{d}'}$$

(c) Inductive definition of the interpretation $\llbracket \mathcal{N} \rrbracket^\iota$ by induction on a sequence of sequent rules giving $\mathcal{N}$, we omit to explicit the valuation ι as well as the active sequent in the sequent rule.



(d) MLL cut-reduction rewriting steps.



(e) Example of two proof-nets of conclusion $X_4^+ \otimes X_5^+$ such that $\mathcal{N} \xrightarrow{*} \mathcal{N}_0$. The labelling of some edges is omitted.

Figure 1 The proof-net syntax and weighted-relational semantics of quantitative MLL.

the box, while weakenings maginalise out this distribution. These operations are categorically axiomatised by the so-called *CD-structure*, for "*copier*" and "*discarder*", e.g. [22, 28]. We explicit here how the structural polarised linear logic rules perfectly fulfil this rôle, showing a natural Curry-Howard correspondence with Bayesian programming.

Given a proof-net $\mathcal{N}$ and an edge e of $\mathcal{N}$, we write by $e : F$ whenever e is labelled by the formula F. By ease of notation, we often write the sequent $F_1, \ldots, F_n$ synonymously for an enumeration $e_1 : F_1, \ldots, e_n : F_n$ of labelled edges, if the edges $e_1, \ldots, e_n$ are clear from the context or inessential. We write $\mathcal{N} : \Delta$ whenever the sequent Δ enumerates the (formulas labelling the) conclusions of $\mathcal{N}$, also speaking about Δ as simply the conclusions of $\mathcal{N}$.

Cut-reduction is defined as a graph-rewriting, replacing a subgraph containing a cut (the *redex*) with a new subgraph (the *contractum*) having the same pending edges. Figure 1d sketches the three different kinds of MLL redexes: ax, $1/\bot$, $\otimes/\parr$. We will write $\mathcal{N} \to \mathcal{N}'$ if $\mathcal{N}$ rewrites into $\mathcal{N}'$ by one single rewriting step. The fact that $\mathcal{N}'$ is still a proof-net is proven by using the so-called correctness criteria (see [18] for details). We denote by $\xrightarrow{*}$ the reflexive and transitive closure of $\to$. A *normal form* is a proof-net which contains no redex of any kind $\{\mathsf{ax}, 1/\bot, \otimes/\parr\}$. Cut-reduction is confluent and strong normalising [18].

▶ **Example 5.** Figure 1e gives an example of a proof-net $\mathcal{N}$ that rewrites into the normal form $\mathcal{N}_0$. Notice that cuts between structural nodes (weakening and contraction) and boxes are not reduced (see Remark 6) so that the normal form $\mathcal{N}_0$ yet contains some cuts. Notice also that different sequences of rewriting steps may start from $\mathcal{N}$ but all of them can be eventually completed into $\mathcal{N}_0$, in accordance with the confluence property.

▶ Remark 6. Weakening and contraction do not erase nor duplicate semantical boxes as this rewriting would break the correspondence with Bayesian networks mentioned in Example 1. In fact, if we rewrote a cut between a contraction and a box b into two distinct copies of b, then this would correspond to create two independent and identically distributed random variables out of a single one and not to duplicate a sample of this latter. The sharing nodes in bayesian networks share samples of random variables but do not duplicate random variables (see [7]). We will discuss this point also in Example 8 using the weighted relational semantics.

3 Weighted Relational Semantics

The quantitative semantics of linear logic refers to a family of denotational models based on linear algebra constructions (tensors, linear functions, direct sums, dual spaces, etc.). Many examples are known in the literature, such as finiteness and Koethe spaces [11, 10], weighted relations [24], probabilistic coherence spaces [6], coherent Banach spaces [20] etc. The common idea is to associate types with a mathematical structure underlying a notion of vector space (or a module) and the poofs with linear maps represented by matrices, or simply vectors in case of proof-nets. We consider here one of the most basic examples of quantitative semantics, the "relations" weighted by non-negative real numbers, but the results of this paper can be adapted trivially to any quantitative semantics mentioned above.

The model of $\mathbb{R}_{\geq 0}$-weighted relations is a variant of the relational semantics of linear logic (see e.g. [2]), where the notion of a subset of a set S, seen as a vector $(b_x)_{x \in S}$ of booleans expliciting whether an element $x \in S$ belongs or not to the subset, is generalised to a vector of non-negative real numbers. This model is known and we thus just sketch here the interpretation of quantitative MLL proof-nets, referring the reader to [24] for more details.

Let us fix some basic notation. Metavariables S, T, U range over finite sets[3]. We denote by $\mathbb{R}_{\geq 0}$ the cone of the non-negative real numbers. Metavariables ϕ, ψ, ξ will range over vectors in $\mathbb{R}_{\geq 0}^{S}$, ϕ_x denoting the scalar associated with $x \in S$ by $\phi \in \mathbb{R}_{\geq 0}^{S}$. The identity matrix over a set S, also called diagonal matrix or Kronecker delta, is denoted $\delta \in \mathbb{R}_{\geq 0}^{S \times S}$ and defined by $\delta_{a,a'} = 1$ if $a = a'$, otherwise $\delta_{a,a'} = 0$.

▶ **Example 7.** The simplest example we consider is the singleton set $\{\star\}$, for some irrelevant element $\star$. The singleton will be associated with multiplicative units 1 and $\perp$ and it induces the module of scalars, as ${\mathbb{R}_{\geq 0}}^{\{\star\}} \simeq \mathbb{R}_{\geq 0}$. The module of couples of non-negative real numbers is instead induced by any set of cardinality 2, like the set of booleans $\{\mathtt{t}, \mathtt{f}\}$: $\mathbb{R}_{\geq 0}^{\{\mathtt{t},\mathtt{f}\}} \simeq \mathbb{R}_{\geq 0}^{2}$. In all the examples of this paper, we will in fact associate the propositional variables with the set $\{\mathtt{t}, \mathtt{f}\}$, so that a proof-net with only one atomic conclusion will be interpreted with a vector $(\lambda_{\mathtt{t}}, \lambda_{\mathtt{f}})$, giving a "score" to the two booleans. Notice that $\mathbb{R}_{\geq 0}^{2} = \mathbb{R}_{\geq 0} \oplus \mathbb{R}_{\geq 0}$, with $\oplus$ denoting the direct sum over modules. This is reflected in linear logic by encoding the type of booleans with the formula $1 \oplus 1$, where $\oplus$ refers to the additive disjunction. We however avoid this notation as we do not consider the full additive connectives here.

More in general, the interpretation of a MLL formula F is a finite set $[\![F]\!]^{\iota}$ defined once we have fixed a *valuation* ι as a function mapping the propositional variables to finite sets. The definition of $[\![F]\!]^{\iota}$ is by induction on F, as follows:

$$[\![X^+]\!]^{\iota} = [\![X^-]\!]^{\iota} ::= \iota(X), \quad [\![1]\!]^{\iota} = [\![\perp]\!]^{\iota} ::= \{\star\}, \quad [\![F \otimes G]\!]^{\iota} = [\![F \parr G]\!]^{\iota} ::= [\![F]\!]^{\iota} \times [\![G]\!]^{\iota}.$$

It is easy to check that the usual isomorphisms of linear logic (like associativity and commutativity of the binary connectives) are validated by set isomorphisms. In particular, we can use tuples $(x_1, \dots, x_n)$ for denoting elements in the interpretation of a n-fold connective, e.g. $[\![F_1 \parr (\cdots \parr F_n)]\!]^{\iota} \simeq \{(x_1, \dots, x_n) \mid \forall i \leq n, x_i \in [\![F_i]\!]^{\iota}\}$.

Weighted relational semantics equates much more than just linear logic isomorphisms, as for example $[\![F]\!]^{\iota} = [\![F^{\perp}]\!]^{\iota}$ for any formula F. More precisely, this semantics has the structure of a compact closed category. There are more refined examples of quantitative semantics which are not compact closed, e.g. probabilistic coherence spaces. Let us stress that our results do not suppose compact closeness.

The interpretation $[\![\mathcal{N}]\!]^{\iota}$ of a proof-net $\mathcal{N}$ of conclusions Γ is a vector in $\mathbb{R}_{\geq 0}^{[\![\parr\Gamma]\!]^{\iota}}$, which can be equivalently seen as a multidimensional matrix indexed by the tuples in $[\![\parr\Gamma]\!]^{\iota}$. The interpretation can be given inductively as sketched by Figure 1c, once we have associated with each box b of conclusions $X_1^-, \dots, X_n^-, Y^+$ a vector $\iota(\mathsf{b}) \in \mathbb{R}_{\geq 0}^{[\![(\parr_i X_i^-) \parr Y^+]\!]^{\iota}}$. This interpretation is invariant under the cut-reduction rules of Figure 1d, i.e. $\mathcal{N} \to \mathcal{N}'$ implies $[\![\mathcal{N}]\!] = [\![\mathcal{N}']\!]$.

▶ **Example 8.** Consider the proof-net $\mathcal{N}'$ of conclusion $X_2^+ \otimes (X_2^- \parr X_3^+)$ contained in the proof-net $\mathcal{N}$ depicted at left of Figure 1e and characterised by the three boxes $\mathsf{b}_1, \mathsf{b}_2, \mathsf{b}_3$ and the tensor and par above the cut over $X_2^+ \otimes (X_2^- \parr X_3^+)$. Notice that there is only one sequence of the generating rules of Figure 1b producing this proof-net: one first applies a par rule under the b_3 conclusions X_2^- and X_3^+, then a tensor between the resulting proof-net and b_2, then a contraction between the two X_1^- conclusions and finally a cut between the conclusion of this contraction and b_1. Figure 1c applied to this sequence of rules gives:

$$[\![\mathcal{N}']\!]^{\iota}_{(x',(x'',x''))} = \sum_{y \in [\![X_1^+]\!]^{\iota}} \iota(\mathsf{b}_1)_y \iota(\mathsf{b}_2)_{(y,x')} \iota(\mathsf{b}_3)_{(y,(x'',x''))}.$$

[3] This kind of denotational semantics are defined for countable sets S in general. Infinite sets are necessary to model linear logic exponential modality as well as the full λ-calculus. Since we focus here to only MLL, we can restrict to finite sets.

Notice that the cut composes the semantics of b_1 with that of the proof-net containing b_2 and b_3, producing the sum over $y \in [\![X_1^+]\!]^\iota$. Notice also that the contraction imposes that the same index y is shared between the two different boxes (b_2 and b_3): contraction duplicates the indexes of the vectors, but it does not yield different copies of the vectors themselves. This is in accordance with Remarks 4 and 6: sharing of sampled values corresponds here to sharing vector indices, which is different from duplicating whole vectors. If we consider in fact the proof-net $[\![\mathcal{N}'']\!]^\iota$ given by a tensor between b_2 and b_3 and two *distinct* copies of b_1, one cut with the X_1^- conclusion of b_2 and the other one with that of b_3, then we would have:

$$[\![\mathcal{N}'']\!]^\iota_{(x',(x'',x''))} = \sum_{y,y' \in [\![X_1^+]\!]^\iota} \iota(\mathsf{b}_1)_y \iota(\mathsf{b}_2)_{(y',x')} \iota(\mathsf{b}_1)_{y'} \iota(\mathsf{b}_3)_{(y',(x'',x''))}.$$

▶ **Example 9.** Let us consider a proof-net $\mathcal{N}$ which is a bunch of $n+1$ axioms over a tree of n contractions, of which edges are labelled by X^-, so that $\mathcal{N}$ has conclusions $X^-, X^+, \dots, X^+$. The denotation $[\![\mathcal{N}]\!]^\iota$ is then a vector indexed by the $(n+2)$-tuples of elements in $\iota(X)$. In fact, by using Figure 1c, one can check that $[\![\mathcal{N}]\!]^\iota$ is a very sparse vector, having zero everywhere but on the tuples of equal elements, i.e. $(x, x, \dots, x)$ for $x \in \iota(X)$, in which case $[\![\mathcal{N}]\!]^\iota$ returns 1. We have here a first source of inefficiency of this kind of semantics, representing the denotation of a proof-net with a vector of dimension exponential in the number of its conclusions, where it would suffice a much more compact structure to store the same information. Section 5 will provide this structure with the notion of *component factor*.

If $\mathcal{N}$ has several cuts, the computation of $[\![\mathcal{N}]\!]^\iota$ can be considerably simplified by using the following lemma, which is reminiscent of the notion of experiment introduced in [18].

▶ **Lemma 10** (Cut bundles). *Let* $\mathrm{Cut}_\Gamma(\mathcal{N})$ *be a proof-net of conclusions* Δ *that can be decomposed into a proof-net* $\mathcal{N}$ *of conclusions* $\Delta, \Gamma, \Gamma^\perp$ *and a bundle of cuts between the formulas in* Γ *and* $\Gamma^\perp$*. Then, for every* $\vec{d} \in [\![\Delta]\!]^\iota$*, we have:* $[\![\mathrm{Cut}_\Gamma(\mathcal{N})]\!]^\iota_{\vec{d}} = \sum_{\vec{c} \in [\![\Gamma]\!]} [\![\mathcal{N}]\!]^\iota_{(\vec{d},\vec{c},\vec{c})}$.

▶ **Example 11.** Let us compute the semantics of the proof-net $\mathcal{N}_0$ in Figure 1e, by using Lemma 10 and Figure 1c. We have that, for any $(x_4, x_5) \in [\![X_4^+ \otimes X_5^+]\!]^\iota$:

$$[\![\mathcal{N}_0]\!]^\iota_{(x_4,x_5)} = \sum_{\substack{x_i \in \iota(X_i^+) \\ \text{for } i \in \{1,2,3\}}} \iota(\mathsf{b}_1)_{x_1} \iota(\mathsf{b}_2)_{(x_1,x_2)} \iota(\mathsf{b}_3)_{(x_1,x_2,x_3)} \iota(\mathsf{b}_4)_{(x_2,x_3,x_4)} \iota(\mathsf{b}_5)_{(x_2,x_5)}$$

With a bit more of effort (due to the presence of axioms) also $[\![\mathcal{N}]\!]^\iota$ can be associated with the above summation. If we suppose that for every i, $\iota(X_i) = \{\mathtt{t}, \mathtt{f}\}$, this summation has a total of 2^3 terms, so that computing the whole vector $[\![\mathcal{N}_0]\!]^\iota$ requires $\sim 2^5$ basic operations[4], i.e. a quantity exponential in the number of the semantical boxes.

By carefully inspecting the summation, one can however realise that it can be refactored so to split factors over independent variables, getting for example the expression:

$$\sum_{x_3} \Big(\sum_{x_2} \big(\sum_{x_1} \iota(\mathsf{b}_1)_{x_1} \iota(\mathsf{b}_2)_{(x_1,x_2)} \iota(\mathsf{b}_3)_{(x_1,x_2,x_3)} \big) \iota(\mathsf{b}_4)_{(x_2,x_3,x_4)} \Big) \iota(\mathsf{b}_5)_{(x_2,x_5)}$$

which, by memorising the intermediate sums, performs the same computation of $[\![\mathcal{N}]\!]^\iota$ in just $\sim 2^3$ operations.This kind of refactoring is at the core of many algorithms for exact inference in Bayesian graphs and the next sections will show how to import these methods.

[4] We are supposing that multiplication, addition and coefficient access are operations of constant cost.

4 Factors

We adapt from Bayesian networks (e.g. [8]) the notion of factor (Definition 18) and of product and projection of factors. A factor carries both a vector *and* a "sharing structure" about what entries of this vector will be shared with possibly other factors so that we avoid the dimension explosion which is the source of inefficiency in Example 9. Bayesian networks use random variables for expressing such a "sharing structure", while we reduce this latter into the very basic definition of set-family, which encompasses the former (Example 15) and generalises to whole quantitative MLL. The terminology "factor" is standard in Bayesian networks, in fact this notion refers to the terms in the multiplication giving a joint distribution as the outcome of the variable elimination algorithm (Algorithm 1). We introduce also a notion of renaming (Definition 29) and of factor renaming (Definition 34) necessary to follow the compositional structure of MLL (see discussion in Example 41).

▶ **Definition 12** (Set-family). *We call* set-family *a finite, indexed family of finite sets, i.e. a map $\mathbb{X}$ from a finite set $\mathcal{I}(\mathbb{X})$ of indices to a set* $\mathsf{Sets}(\mathbb{X})$ *of finite sets. We denote by $\mathbb{X}(a)$ the set associated with index $a \in \mathcal{I}(\mathbb{X})$ in $\mathbb{X}$. Meta-variables $\mathbb{X}$, $\mathbb{Y}$, $\mathbb{Z}$ will range over such set-families.*

Two families $\mathbb{X}$ and $\mathbb{Y}$ are compatible *whenever for all $a \in \mathcal{I}(\mathbb{X}) \cap \mathcal{I}(\mathbb{Y})$, $\mathbb{X}(a) = \mathbb{Y}(a)$. Set-theoretical operations lift to compatible set-families by applying the former to the graph of these latter, e.g. the intersection $\mathbb{X} \cap \mathbb{Y}$ is the set-family defined by $\mathcal{I}(\mathbb{X} \cap \mathbb{Y}) ::= \mathcal{I}(\mathbb{X}) \cap \mathcal{I}(\mathbb{Y})$ and $(\mathbb{X} \cap \mathbb{Y})(a) ::= \mathbb{X}(a) = \mathbb{Y}(a)$ for every $a \in \mathcal{I}(\mathbb{X} \cap \mathbb{Y})$. Similarly, we will consider the union $\mathbb{X} \cup \mathbb{Y}$ and the set-theoretical difference $\mathbb{X} \setminus \mathbb{Y}$. In the same spirit, we write $\mathbb{Y} \subseteq \mathbb{X}$, for $\mathcal{I}(\mathbb{Y}) \subseteq \mathcal{I}(\mathbb{X})$ and for every $a \in \mathcal{I}(\mathbb{Y})$, $\mathbb{Y}(a) = \mathbb{X}(a)$.*

Given a set-family $\mathbb{X}$, we denote by $[\![\mathbb{X}]\!]$ the cartesian product $\prod_{a\in\mathcal{I}(\mathbb{X})} \mathbb{X}(a)$ of the sets in $\mathsf{Sets}(\mathbb{X})$*, where the same set in* $\mathsf{Sets}(\mathbb{X})$ *can appear multiple times in the product if associated with multiple indices. We denote the elements of $[\![\mathbb{X}]\!]$ with the vectorial notation $\vec{x}$, to underline that it is an element in a cartesian product rather than in a simple set.*

▶ **Notation 13.** *Any element $\vec{x} \in [\![\mathbb{X}]\!]$ can be seen as a collection $(x_a)_{a\in\mathcal{I}(\mathbb{X})}$ of elements in* $\mathsf{Sets}(\mathbb{X})$*. In particular, given $\mathbb{Y} \subseteq \mathbb{X}$, we denote by $\vec{x}|_{\mathbb{Y}}$ the projected element $(x_a)_{a\in\mathcal{I}(\mathbb{Y})} \in [\![\mathbb{Y}]\!]$. Similarly, given two set-families $\mathbb{X}, \mathbb{Y}$ having disjoint sets of indexes, so clearly compatible, the elements of $[\![\mathbb{X} \uplus \mathbb{Y}]\!]$ can be written as $(\vec{x}, \vec{y})$, for $\vec{x} \in [\![\mathbb{X}]\!]$ and $\vec{y} \in [\![\mathbb{Y}]\!]$.*

Notice that if $\mathbb{X}$ is empty, then $[\![\mathbb{X}]\!]$ is the singleton set $\{()\}$.

▶ **Notation 14.** *Since finite, set-families can be given by enumerating their graph, like in $\mathbb{X} = \{(a_1, S_1), \dots, (a_n, S_n)\}$. In this case we have: $\mathcal{I}(\mathbb{X}) = \{a_1, \dots, a_n\}$ and* $\mathsf{Sets}(\mathbb{X}) = \{S_1, \dots, S_n\}$*. In this latter set, the possible repetitions are equated, so* $\mathsf{Sets}(\mathbb{X})$ *might have less than n elements.*

▶ **Example 15.** A finite set $\{X_1, \dots, X_n\}$ of finite random variables defines the set-family $\mathbb{X} = \{(X_1, [\![X_1]\!]), \dots, (X_n, [\![X_n]\!])\}$, where $[\![X_i]\!]$ denotes the finite set of the possible outcomes taken by the random variable X_i. Notice that $[\![\mathbb{X}]\!]$ is then the set of samples of the joint distribution over $X_1, \dots, X_n$. To be more explicit, suppose that each random variable X_i is boolean, i.e. $[\![X_i]\!] = \{\mathtt{t}, \mathtt{f}\}$ for all $i \leq n$, then $\mathsf{Sets}(\mathbb{X}) = \{\{\mathtt{t}, \mathtt{f}\}\}$, while $[\![\mathbb{X}]\!] = \{(b_1, \dots, b_n) \mid b_i \in \{\mathtt{t}, \mathtt{f}\}\}$.

▶ **Example 16.** Consider a sequent $\Gamma = X_1^\circ, \dots, X_n^\circ$ of atomic formulas. A natural set-family that can be associated with Γ and a valuation ι, has indices the sequent positions $\{1, \dots, n\}$ and it maps a position i to the set $\iota(X_i)$. This set-family however is not the only possible one: for example, one may take as indices the propositional variables $X_1, \dots, X_n$, where multiple occurrences of the same variable are equated, and map X_i to $\iota(X_i)$. The two set-families are quite different if Γ contains repetitions. Namely, let $\Gamma = X^+, X,^+ X^-, Y^-$, with $[\![X]\!] = [\![Y]\!] = \{\mathtt{t}, \mathtt{f}\}$. The two set-families are:

$$\mathbb{X} = \{(1, \{\mathtt{t}, \mathtt{f}\}), (2, \{\mathtt{t}, \mathtt{f}\}), (3, \{\mathtt{t}, \mathtt{f}\}), (4, \{\mathtt{t}, \mathtt{f}\})\}, \qquad \mathbb{Y} = \{(X, \{\mathtt{t}, \mathtt{f}\}), (Y, \{\mathtt{t}, \mathtt{f}\})\}.$$

▶ Remark 17. Notice that $\mathbb{Z}, \mathbb{Y} \subseteq \mathbb{X}$ implies that both $\mathbb{Z}$ and $\mathbb{Y}$ are compatible. Henceforth we will always consider families which are subset of a fixed "universal" family (underlined by a proof-net), so that the compatibility condition in Definition 12 is not an issue and hence will be often not mentioned.

▶ **Definition 18** (Factor). *A* generalised factor*, or simply* factor*, ϕ is a pair* $(\mathsf{Fam}(\phi), \mathrm{Fun}(\phi))$ *of a set-family* $\mathsf{Fam}(\phi)$ *and a function* $\mathrm{Fun}(\phi)$ *from the set* $[\![\mathsf{Fam}(\phi)]\!]$ *to* $\mathbb{R}_{\geq 0}$.

We will short the notation $\mathrm{Fun}(\phi)$ *by writing just ϕ when it is clear from the context that we are considering the function associated with a factor and not the whole pair* $(\mathsf{Fam}(\phi), \mathrm{Fun}(\phi))$*. We often consider* $\mathrm{Fun}(\phi)$ *as a vector indexed by the elements of its domain, so that $\phi_{\vec{x}}$ stands for* $\mathrm{Fun}(\phi)(\vec{x})$*, for every $\vec{x} \in [\![\mathsf{Fam}(\phi)]\!]$.*

▶ **Example 19.** Let us recall the set-family $\mathbb{Y} = \{(X, \{\mathtt{t}, \mathtt{f}\}), (Y, \{\mathtt{t}, \mathtt{f}\})\}$ of Example 16, and consider the function $\mathrm{Fun}(\phi)$ given by $\{(\mathtt{t}_X, \mathtt{t}_Y) \mapsto 0.2, (\mathtt{t}_X, \mathtt{f}_Y) \mapsto 0.25, (\mathtt{f}_X, \mathtt{t}_Y) \mapsto 0.25, (\mathtt{f}_X, \mathtt{f}_Y) \mapsto 0.3\}$. The pair $\phi = (\mathbb{Y}, \mathrm{Fun}(\phi))$ is an example of factor. Intuitively, ϕ can be seen as the presentation $0.2e_{(\mathtt{t}_X,\mathtt{t}_Y)} + 0.25e_{(\mathtt{t}_X,\mathtt{f}_Y)} + 0.25e_{(\mathtt{f}_X,\mathtt{t}_Y)} + 0.3e_{(\mathtt{f}_X,\mathtt{f}_Y)}$ of a vector in $\mathbb{R}^4_{\geq 0}$ with respect to a set of basis vectors $e_{(b_X,b_Y)}$ associated with the elements in $(b_X, b_Y) \in [\![\mathbb{Y}]\!]$.

▶ **Definition 20** (Factor projection). *Let ϕ be a factor and let $\mathbb{X}$ be a set-family compatible with* $\mathsf{Fam}(\phi)$*,* the projection of ϕ to $\mathbb{X}$ *is the factor $\pi_{\mathbb{X}}(\phi)$ defined by:*

$$\mathsf{Fam}(\pi_{\mathbb{X}}(\phi)) ::= \mathbb{X}, \qquad \pi_{\mathbb{X}}(\phi)_{\vec{x}} ::= \sum_{\vec{y} \in [\![\mathsf{Fam}(\phi) \setminus \mathbb{X}]\!]} \phi_{(\vec{x}|_{\mathsf{Fam}(\phi)}, \vec{y})}, \qquad \textit{for } \vec{x} \in [\![\mathbb{X}]\!].$$

▶ **Example 21.** Recall the set-family $\mathbb{Y}$ and the factor $\mathrm{Fun}(\phi)$ given in Example 19, let $\mathbb{X} = \{(X, \{\mathtt{t}, \mathtt{f}\})\} \subseteq \mathbb{Y}$. We have that $\pi_{\mathbb{X}}(\phi) = \{\mathtt{t}_X \mapsto 0.45, \mathtt{f}_X \mapsto 0.55\}$. Let now $\mathbb{Z} = \mathbb{X} \uplus \{(Z, \{\mathtt{t}, \mathtt{f}\})\}$, we have that $\pi_{\mathbb{Z}}(\phi) = \{(\mathtt{t}_X, \mathtt{t}_Z) \mapsto 0.45, (\mathtt{t}_X, \mathtt{f}_Z) \mapsto 0.45, (\mathtt{f}_X, \mathtt{t}_Z) \mapsto 0.55, (\mathtt{f}_X, \mathtt{f}_Z) \mapsto 0.55\}$. Notice in particular that the factor projection to a set-family $\mathbb{Z}$ does not preserve in general the property of being a probability mass function, unless $\mathbb{Z} \subseteq \mathsf{Fam}(\phi)$.

▶ Remark 22. With the notations of Definition 20, if $\mathbb{X} \subseteq \mathsf{Fam}(\phi)$, then $\pi_{\mathbb{X}}(\phi)$ corresponds to what is called in Bayesian programming *summing out* $\mathsf{Fam}(\phi) \setminus \mathbb{X}$, which gives the marginal distribution over $\mathbb{X}$. Suppose on the contrary that $\mathbb{X}$ and $\mathsf{Fam}(\phi)$ are disjoint, then for every $\vec{x} \in [\![\mathbb{X}]\!]$, $\pi_{\mathbb{X}}(\phi)_{\vec{x}}$ is the total mass of ϕ, i.e. $\sum_{\vec{y} \in [\![\mathsf{Fam}(\phi)]\!]} \phi_{\vec{y}}$.

▶ Remark 23. Suppose that $\mathsf{Fam}(\phi)$ has n indices and that k is the maximum cardinality of a set in $\mathsf{Sets}(\mathsf{Fam}(\phi))$, then the computation of the whole vector $\pi_{\mathbb{X}}(\phi)$ is in $O(k^n)$.

▶ **Definition 24** (Binary factor product). *Given two factors ϕ and ψ, such that* $\mathsf{Fam}(\phi)$ *and* $\mathsf{Fam}(\psi)$ *are compatible, we define their* factor product *as the factor $\phi \odot \psi$ given by:*

$$\mathsf{Fam}(\phi \odot \psi) ::= \mathsf{Fam}(\phi) \cup \mathsf{Fam}(\psi), \qquad (\phi \odot \psi)_{\vec{z}} ::= \phi_{\vec{z}|_{\mathsf{Fam}(\phi)}} \psi_{\vec{z}|_{\mathsf{Fam}(\psi)}}, \qquad \textit{for } \vec{z} \in [\![\mathsf{Fam}(\phi \odot \psi)]\!].$$

▶ Remark 25. If $\mathsf{Fam}(\phi) \cup \mathsf{Fam}(\psi)$ has n indices and k is the maximum cardinality of a set in $\mathsf{Sets}(\mathsf{Fam}(\phi) \cup \mathsf{Fam}(\psi))$, then the computation of the whole vector $\phi \odot \psi$ is in $O(k^n)$.

▶ **Example 26.** In terms of MLL operations, factor products correspond to a $\otimes$ product plus a bunch of contractions on the common indexes. For example, let us take as indexes the propositional variables and as sets just $\{\mathtt{t}, \mathtt{f}\}$ (recall Example 16) and consider $\mathsf{Fam}(\phi) = \{X_2, X_3, X_4\}$ and $\mathsf{Fam}(\psi) = \{X_2, X_5\}$ (this choice is reminiscent of the variables in the proof-nets in Figure 1e, in fact ϕ and ψ can be associated with the boxes, respectively, b_4 and b_5). Then, $\mathrm{Fun}(\phi \odot \psi)$ is over $\{X_2, X_3, X_4, X_5\}$, so of dimension 2^4, while $\mathrm{Fun}(\phi) \otimes \mathrm{Fun}(\psi)$ is a vector indexed by tuples of 5 booleans, so of dimension 2^5.

The next proposition states expected properties of factor projection and product that are fundamental in the sequel.

▶ **Proposition 27.** *Factor product is associative and commutative, with neutral element the empty factor* $(\emptyset, 1)$*. Moreover:*

1. $\pi_{\mathbb{X}\cup\mathbb{Z}}(\pi_{\mathbb{X}\cup\mathbb{Y}}(\phi)) = \pi_{\mathbb{X}\cup\mathbb{Z}}(\phi)$*, whenever* $\mathbb{Y} \subseteq \mathsf{Fam}(\phi)$ *and* $\mathbb{Z} \cap \mathsf{Fam}(\phi) = \emptyset$*;*
2. $\pi_{\mathbb{X}}(\phi \odot \psi) = \pi_{\mathbb{X}}(\phi) \odot \psi$*, whenever* $\mathsf{Fam}(\psi) \subseteq \mathbb{X}$*.*

▶ **Definition 28** (n-factor product)**.** *Let* I *be a finite set. Given a collection of pairwise compatible factors* $(\phi_i)_{i\in I}$*, we define their* factor product *as the factor* $\bigodot_{i\in I} \phi_i ::= \phi_{i_1} \odot \cdots \odot \phi_{i_n}$*, for some enumeration of* I*. This is well-defined independently from the chosen enumeration because of Proposition 27.*

Section 5 will associate factors to MLL proof-nets and in order to make this association compositional (Theorem 48) we introduce the following notion of renaming, as the contraction and cut rules of Figure 1b may change the sharing structure associated with a proof-net.

▶ **Definition 29** (Renaming)**.** *A* renaming f *from a set-family* $\mathbb{X}$ *to a set-family* $\mathbb{Y}$ *is a map from* $\mathcal{I}(\mathbb{X})$ *to* $\mathcal{I}(\mathbb{Y})$ *such that for all* $a \in \mathcal{I}(\mathbb{X})$*, we have* $\mathbb{X}(a) = \mathbb{Y}(f(a))$*. Any such renaming* f *induces the map* f° *from* $[\![\mathbb{Y}]\!]$ *to* $[\![\mathbb{X}]\!]$ *by:*

$$\text{for } \vec{y} \in [\![\mathbb{Y}]\!], \qquad f^\circ(\vec{y}) ::= (y_{f(a)})_{a\in\mathcal{I}(\mathbb{X})} \in [\![\mathbb{X}]\!]. \tag{2}$$

Moreover, we say that a point $\vec{x} \in [\![\mathbb{X}]\!]$ agrees on f *whenever, for every* $a, a' \in \mathcal{I}(\mathbb{X})$*,* $f(a) = f(a')$ *implies that* $x_a = x_{a'}$*.*

▶ Remark 30. The notion of “agreeing on a renaming f” generalises the notion in Bayesian programming of a set of samples that “agrees on the same random variables” as used in e.g. [8].

▶ **Notation 31.** *Given a renaming* f *from* $\mathbb{X}$ *to* $\mathbb{Y}$*, and a set-family* $\mathbb{X}' \subseteq \mathbb{X}$*, we denote by* $f(\mathbb{X}')$ *the set-family having as indices the set* $f(\mathcal{I}(\mathbb{X}')) \subseteq \mathcal{I}(\mathbb{Y})$ *and that it associates with any* $b \in f(\mathcal{I}(\mathbb{X}'))$ *the set* $\mathbb{Y}(b)$*. Notice that* $f(\mathbb{X}') \subseteq \mathbb{Y}$*.*

▶ **Proposition 32.** *Given a renaming from* $\mathbb{X}$ *to* $\mathbb{Y}$*, the image set of* f° *is the subset of* $[\![\mathbb{X}]\!]$ *of the elements which agree on* f*. If, moreover,* f *is surjective over* $\mathcal{I}(\mathbb{Y})$*, then* f° *is an injective map from* $[\![\mathbb{Y}]\!]$ *to* $[\![\mathbb{X}]\!]$*, hence a bijection from* $[\![\mathbb{Y}]\!]$ *to* $\{\vec{x} \in [\![\mathbb{X}]\!] \mid \vec{x}$ *agrees on* $f\}$*. In this case, we denote its inversion by* $f^\bullet$*.*

▶ **Example 33.** Recall the set-families $\mathbb{X}$ and $\mathbb{Y}$ in Example 16, associated with the sequent $\Gamma = X^+, X,^+ X^-, Y^-$. Let us consider the following two specific renamings from $\mathbb{X}$ to $\mathbb{Y}$:

$$f = \begin{cases} 1,2,3 & \mapsto X \\ 4 & \mapsto Y \end{cases}, \qquad\qquad g = \begin{cases} 1,2,3,4 & \mapsto X \end{cases}.$$

and take for example $\vec{y} = (\mathtt{t}_X, \mathtt{f}_Y) \in [\![\mathbb{Y}]\!]$. We have (we use the natural order (1,2,3,4) to represent elements in $[\![\mathbb{X}]\!]$):

$$f^\circ(\vec{y}) = (\mathtt{t}, \mathtt{t}, \mathtt{t}, \mathtt{f}), \qquad\qquad g^\circ(\vec{y}) = (\mathtt{t}, \mathtt{t}, \mathtt{t}, \mathtt{t}).$$

Notice in fact that $f^\circ(\vec{y})$ agrees on f but not on g, while $g^\circ(\vec{y})$ agrees on both f and g. Also notice that f is surjective and in fact f° is an injection, while g is not surjective and in fact g° is not a injection, for example: $g^\circ(\mathtt{t}_X, \mathtt{f}_Y) = g^\circ(\mathtt{t}_X, \mathtt{t}_Y)$.

▶ **Definition 34** (Factor renaming). *Let f be a renaming from $\mathsf{Fam}(\phi)$ to a set-family $\mathbb{X}$. The* renaming of ϕ along f *is the factor $f(\phi)$ defined by:*

$$\mathsf{Fam}(f(\phi)) ::= f(\mathsf{Fam}(\phi)), \qquad f(\phi)_{\vec{x}} ::= \phi_{f^\star(\vec{x})}, \qquad \textit{for } \vec{x} \in [\![f(\mathsf{Fam}(\phi))]\!].$$

▶ **Example 35.** Recall the renaming f of Example 33 between the set-families $\mathbb{X}$ and $\mathbb{Y}$ given in Example 16. Consider the factor ϕ over $\mathbb{X}$ defined by $\phi_{(b_1,b_2,b_3,b_4)} ::= 1$ if $b_1 = b_2 = b_3$, 0 otherwise. This factor corresponds to the interpretation of the proof-net having a weakening producing Y^- and the axiom on top of a contraction giving X^+, X^+, X^-. Then $f(\phi)$ is over $\mathbb{Y}$ and its map is the constant function giving 1.

▶ Remark 36. Notice that one can formalise the notions of this section in a categorical way, considering a category of renamings as morphisms between set-families. We did not develop this more abstract presentation as not needed in the sequel.

5 Weighted Semantics by Factors, Atomic Case

We apply to MLL the notions introduced in Section 4. It is convenient to restrict to atomic proof-nets and then to extend the results to the non-atomic case in Section 6. Definitions 37 and 45 associate two different set-families with an atomic proof-net $\mathcal{N}$, the edge and the component set-families. The edge set-family permits to consider the standard weighted interpretation $[\![\mathcal{N}]\!]$ as a factor (Definition 39), while the component set-family yields a more compressed representation of $[\![\mathcal{N}]\!]$, the component factor (Definition 40), which has a form of compositionality (Theorem 48) and hence can be computed directly (Theorem 49) without using the rules of Figure 1c. Henceforth, this section fixes a valuation ι and considers only atomic proof-nets. We recall that an atomic proof-net can contain only axioms, cuts, weakening, contractions and semantical boxes.

▶ **Definition 37** (Edge set-family). *Let $\mathcal{N}$ be an atomic proof-net and ι be a valuation. The* edge set-family of $\mathcal{N}$, *written by $\mathsf{Fam}^\iota_e(\mathcal{N})$, has the edges of $\mathcal{N}$ as indices and associates with an edge $e : X^\circ$ the set $\iota(X)$. Given a sequence Γ of edges $e_1 : X^\circ_1, \ldots, e_n : X^\circ_n$, we extend the metavariable Γ to denote also the edge set-family $\{(e_1, \iota(X_1)), \ldots, (e_n, \iota(X_n))\} \subseteq \mathsf{Fam}^\iota_e(\mathcal{N})$.*

▶ Remark 38. Let Γ be $e_1 : X^\circ_1, \ldots, e_n : X^\circ_n$. The convention of denoting by Γ both the underlined sequent $X^\circ_1, \ldots, X^\circ_n$ and the edge set-family $\{(e_1, \iota(X_1)), \ldots, (e_n, \iota(X_n))\}$ is coherent because the cartesian product $[\![\Gamma]\!]$ associated with the edge set-family is the same set as the weighted denotation $[\![\Gamma]\!]^\iota$, this latter also denoted simply by $[\![\Gamma]\!]$. This ease of notation is necessary to avoid a formalism overkill.

▶ **Definition 39.** *Given a valuation ι, the* edge factor *of an atomic proof-net $\mathcal{N}$ has as set-family the edge-set family induced by the conclusions of $\mathcal{N}$ and as function the weighted interpretation $[\![\mathcal{N}]\!]^\iota$. We take the liberty to denote this edge factor also by $[\![\mathcal{N}]\!]^\iota$.*

▶ **Definition 40** (Component set-family and renaming). *Let $\mathcal{N}^{\neg \mathsf{b}}$ denote the graph obtained from an atomic proof-net $\mathcal{N}$ by removing all its semantical boxes, so keeping their conclusions as pending edges of the graph. Given an undirected connected component $\mathcal{M}$ of $\mathcal{N}^{\neg \mathsf{b}}$, one can remark that all edges of $\mathcal{M}$ are atomic formulas over a unique variable, let us denote it $X_{\mathcal{M}}$. The* component set-family of $\mathcal{N}$, *written $\mathsf{Fam}^{\iota}_{c}(\mathcal{N})$, has as indices the undirected connected components of $\mathcal{N}^{\neg \mathsf{b}}$ and associates with a component $\mathcal{M}$ the set $\iota(X_{\mathcal{M}})$. The* component renaming of $\mathcal{N}$, *written $\ell_{\mathcal{N}}$, is the renaming from $\mathsf{Fam}^{\iota}_{e}(\mathcal{N})$ to $\mathsf{Fam}^{\iota}_{c}(\mathcal{N})$ mapping the edges of $\mathcal{N}$ to the connected component of $\mathcal{N}^{\neg \mathsf{b}}$ they belong to.*

▶ **Example 41.** Consider the atomic proof-net $\mathcal{N}_a$ obtained from the proof-net $\mathcal{N}_0$ in Figure 1e by removing the tensor node, so that $\mathcal{N}_a$ has conclusions X_4^+, X_5^+. Notice that ${\mathcal{N}_a}^{\neg \mathsf{b}}$ has five connected components which correspond to the five propositional variables $X_1, \dots, X_5$. If however we consider the sub proof-net $\mathcal{N}'_a$ obtained from $\mathcal{N}_a$ by removing the cut and the contraction insisting on the edges typed by, e.g., X_1^-, we have that ${\mathcal{N}'_a}^{\neg \mathsf{b}}$ has now seven connected components, in particular three of them supports the same variable X_1. This example shows that the generating rules of MLL (Figure 1b) require not to mix up the component set-family with the variables labelling the edges (see also Remark 55).

▶ Remark 42. The $\ell_{\mathcal{N}}$ information can be memorised once and for all by adding a further labelling over the edges of $\mathcal{N}$ giving the same index to the edges belonging to the same connected component of $\mathcal{N}^{\neg \mathsf{b}}$. This labelling can be computed in linear time with respect to the size of $\mathcal{N}$, by adapting one of the many connected component algorithms.

▶ **Notation 43.** *Let $\Gamma = e_1 : X_1^{\circ}, \dots, e_n : X_n^{\circ}$ be a set of edges of $\mathcal{N}$, so that Γ is also the set-family $\{(e_1, \iota(X_1)), \dots, (e_n, \iota(X_n))\}$ contained in $\mathsf{Fam}^{\iota}_{e}(\mathcal{N})$. By Notation 31, the writing $\ell_{\mathcal{N}}(\Gamma)$ denotes the set-family $\{(\ell_{\mathcal{N}}(e_1), \iota(X_1)), \dots, (\ell_{\mathcal{N}}(e_n), \iota(X_n))\} \subseteq \mathsf{Fam}^{\iota}_{c}(\mathcal{N})$. Notice that $\ell_{\mathcal{N}}$ is surjective on $\mathcal{I}(\ell_{\mathcal{N}}(\Gamma))$, so we can apply Proposition 32, getting the two maps:*

- *$\ell^{\circ}_{\mathcal{N}}$ from $[\![\ell_{\mathcal{N}}(\Gamma)]\!]$ to $[\![\Gamma]\!]$,*
- *its inverse $\ell^{\bullet}_{\mathcal{N}}$ from $\{\vec{d} \in [\![\Gamma]\!] \mid \vec{d}$ agrees on $\ell_{\mathcal{N}}\}$ to $[\![\ell_{\mathcal{N}}(\Gamma)]\!]$.*

▶ Remark 44. In general, given a set of edges Γ of an atomic proof-net $\mathcal{N}$, we have that: $[\![\ell_{\mathcal{N}}(\Gamma)]\!] = [\![\Gamma]\!]$ if, and only if, all edges in Γ belong to pairwise different connected components of $\mathcal{N}^{\neg \mathsf{b}}$. Moreover, if $[\![\ell_{\mathcal{N}}(\Gamma)]\!] = [\![\Gamma]\!]$, then every $\vec{d} \in [\![\Gamma]\!]$ agrees on $\ell_{\mathcal{N}}$.

▶ **Definition 45.** *Given a valuation ι, the* component factor *of an atomic proof-net $\mathcal{N}$ is the renaming $\ell_{\mathcal{N}}([\![\mathcal{N}]\!]^{\iota})$ of its edge factor, i.e. if Δ are the conclusions of $\mathcal{N}$, $\mathsf{Fam}(\ell_{\mathcal{N}}([\![\mathcal{N}]\!]^{\iota})) = \ell_{\mathcal{N}}(\Delta)$ and for $\vec{d} \in [\![\ell_{\mathcal{N}}(\Delta)]\!]$, $\mathrm{Fun}(\ell_{\mathcal{N}}([\![\mathcal{N}]\!]^{\iota})_{\vec{d}} = [\![\mathcal{N}]\!]^{\iota}_{\ell^{\circ}_{\mathcal{N}}(\vec{d})}$.*

▶ **Example 46.** Recall the proof-net $\mathcal{N}$ and the valuation ι of Example 9. Notice that $\mathsf{Fam}(\ell_{\mathcal{N}}([\![\mathcal{N}]\!]^{\iota}))$ is a singleton as the $n+2$ conclusions of $\mathcal{N}$ belong to the same component, and $\mathrm{Fun}(\ell_{\mathcal{N}}([\![\mathcal{N}]\!]^{\iota})) = (1_{\mathtt{t}}, 1_{\mathtt{f}})$, which is a more parsimonious object than $[\![\mathcal{N}]\!]^{\iota}$, this latter of dimension exponential in n.

Proposition 47 details how to recover the original denotation of $\mathcal{N}$ out of its component factor.

▶ **Proposition 47.** *Let $\mathcal{N}$ be an atomic proof-net of conclusions Δ. For every $\vec{d} \in [\![\Delta]\!]$, we have that:*

$$[\![\mathcal{N}]\!]^{\iota}_{\vec{d}} = \begin{cases} \ell_{\mathcal{N}}([\![\mathcal{N}]\!]^{\iota})_{\ell^{\bullet}_{\mathcal{N}}(\vec{d})} & \text{if } \vec{d} \text{ agrees on } \ell_{\mathcal{N}}, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

The following is the core theorem of this paper: MLL cuts correspond to a factor product plus a projection.

▶ **Theorem 48.** *Let $\mathcal{N} = \mathrm{Cut}_\Gamma(\mathcal{N}', \mathcal{N}'')$ be an atomic proof-net of conclusions Δ obtained by connecting by a bunch of cuts over a sequent Γ a sub proof-net $\mathcal{N}'$, of conclusions Γ, Δ', and $\mathcal{N}''$, of conclusions $\Gamma^\perp, \Delta''$, so that $\Delta = \Delta', \Delta''$. We have:*

$$\ell_{\mathcal{N}}(\llbracket \mathcal{N} \rrbracket^\iota) = \pi_{\ell_{\mathcal{N}}(\Delta)}(\ell_{\mathcal{N}}(\llbracket \mathcal{N}' \rrbracket^\iota) \odot \ell_{\mathcal{N}}(\llbracket \mathcal{N}'' \rrbracket^\iota)) \tag{4}$$

As a consequence, we can compute the component factor of $\mathcal{N}$ without passing via $\llbracket \mathcal{N} \rrbracket^\iota$:

▶ **Theorem 49.** *Let $\mathcal{N}$ be an atomic proof-net with conclusions Δ. We have: $\ell_{\mathcal{N}}(\llbracket \mathcal{N} \rrbracket^\iota) = \pi_{\ell_{\mathcal{N}}(\Delta)}(\bigodot_{\mathsf{b} \in \mathsf{b}(\mathcal{N})} \ell_{\mathcal{N}}(\iota(\mathsf{b})))$.*

▶ **Example 50.** Consider an atomic proof-net $\mathcal{N}$ of conclusions Δ such that no edge in Δ is connected in $\mathcal{N}^{\neg\mathsf{b}}$ with a conclusion of a box of $\mathcal{N}$. By Remark 22, $\pi_{\ell_{\mathcal{N}}(\Delta)}(\bigodot_{\mathsf{b} \in \mathsf{b}(\mathcal{N})} \iota(\mathsf{b}))$ is the constant function giving the total mass of the vectors associated with the boxes of $\mathcal{N}$.

Consider the atomic proof-net $\mathcal{N}_a$ obtained by removing the tensor node from the proof-net $\mathcal{N}_0$ of Figure 1e. If we apply Theorem 49 to $\mathcal{N}_a$, we will obtain exactly the same summation given in Example 11, in fact the edge and component set-families of the conclusions of $\mathcal{N}_a$ are the same. However, we have now the correct formalism to apply exact inference algorithms to refactor the expression $\pi_{\ell_{\mathcal{N}}(\Delta)}(\bigodot_{\mathsf{b} \in \mathsf{b}(\mathcal{N})} \ell_{\mathcal{N}}(\iota(\mathsf{b})))$, by taking advantage of the distributivity law of factor product over the projection (Proposition 27). We adapt here one among the simplest such algorithms, called the sum-product variable elimination algorithm, first introduced in [29], see [8] as a reference. The terminology "variable elimination" is because this procedure infers from a Bayesian network the marginal distribution of a random variable X out of a family[5] $\mathbb{X}$ of variables containing X, by "eliminating" all the other variables in $\mathbb{X}$. In our case, what we "eliminate" are the $\mathcal{N}^{\neg\mathsf{b}}$ components of the conclusions of the box factors containing no conclusion of the proof-net.

■ **Algorithm 1** MLL Sum-Product Algorithm.

```
input:
 1: $\mathcal{N}$                      ▷ an atomic proof-net of conclusions $\Delta$
 2: $\iota$                            ▷ a valuation map
 3: $\omega$      ▷ A linear order on the components in $\mathsf{Fam}_c^\iota(\mathcal{N})$ not having a conclusion in $\Delta$
output: the factor $\ell_{\mathcal{N}}(\llbracket \mathcal{N} \rrbracket^\iota)$
 4: $F \leftarrow \{\ell_{\mathcal{N}}(\iota(\mathsf{b})) \mid \mathsf{b} \in \mathsf{b}(\mathcal{N})\}$      ▷ Factors of $\mathsf{b}(\mathcal{N})$
 5: for $C$ in $\omega$ do
 6:     $F_c \leftarrow \{\phi \in F \mid C \in \mathcal{I}(\mathsf{Fam}(\phi))\}$
 7:     $\psi \leftarrow \bigodot_{\phi \in F_c} \phi$                  ▷ Product
 8:     $\rho \leftarrow \pi_{\mathsf{Fam}(\psi) \setminus \{C\}}(\psi)$              ▷ Sum-out
 9:     $F \leftarrow \{\rho\} \cup (F \setminus F_c)$
    return $\bigodot_{\phi \in F} \phi$
```

Algorithm 1 is our adaptation of the sum-product algorithm. Given a linear order ω over the connected components of $\mathcal{N}^{\neg\mathsf{b}}$ which contains no conclusion of $\mathcal{N}$, the algorithm proceeds as follows: line 4 initialises a variable F with the set of factors to compute; line 5

[5] Any resemblance to the notations in Section 4 is purely voluntary.

takes from ω the next connected component C to process; line 6 gathers in a variable F_c all factors in F which have C as an index (i.e. a conclusion in C); line 7 computes the product of these factors and line 8 projects it on the components different from C (a.k.a. summing out C); line 9 updates F by replacing the processed factors with the result of this projection and then it jumps back to line 5. At the end of this loop, F contains a set of factors indexed over the components of $\mathcal{N}^{\neg \mathsf{b}}$ connected with the conclusions in $\mathcal{N}$ and then it returns their product.

Soundness follows from Proposition 27 and Theorem 49:

▶ **Theorem 51.** *Algorithm 1 returns $\ell_{\mathcal{N}}([\![\mathcal{N}]\!]^{\iota})$ if fed with an atomic proof-net $\mathcal{N}$, a valuation ι and a linear order on the components in $\mathsf{Fam}^{\iota}_{c}(\mathcal{N})$ not containing any conclusion of $\mathcal{N}$.*

▶ **Example 52.** Consider the atomic proof-net $\mathcal{N}_a$ obtained by removing the tensor node from the proof-net $\mathcal{N}_0$ of Figure 1e (Example 1) and use the numbers $1, 2, 3, 4, 5$ to denote the five connected components of $\mathcal{N}_a{}^{\neg \mathsf{b}}$ such that component i is supported by variable X_i. The components to eliminate are $1, 2, 3$. By taking the order $\omega = 1 < 2 < 3$, Algorithm 1 will calculate the following intermediate factors: $\rho_1 = \pi_{\{2,3\}}(\iota(\mathsf{b}_1) \odot \iota(\mathsf{b}_2) \odot \iota(\mathsf{b}_3))$, $\rho_2 = \pi_{\{2,4\}}(\rho_1 \odot \iota(\mathsf{b}_4))$, $\rho_3 = \pi_{\{4,5\}}(\rho_2 \odot \iota(\mathsf{b}_5))$, the output being ρ_3. This yields exactly the factored equation in Example 11 and it allows to calculate the whole semantics of $\mathcal{N}_a$ in $O(k^3)$ basic operations, if k is the maximal cardinality of the sets associated with the propositional variables appearing in the proof-net.

▶ Remark 53. A run of Algorithm 1 depends on the chosen order ω. Different orders yield different factorisations and have different performances. For example, by taking the inverse order $3 < 2 < 1$ in Example 52 we get a run in $O(k^4)$, which is an order of magnitude slower than $1 < 2 < 3$, although yet more efficient than the immediate recursive algorithm induced by the standard semantics (Example 11).

In general, Algorithm 1 is in $O(nk^w)$, where n is the length of ω (i.e. the number of components to eliminate), k is the maximal cardinality of a set interpreting an atomic variable (in our examples we always suppose $k = 2$, for the two booleans) and w is the maximal cardinality of Fam(ϕ), for ϕ a factor created/used by the algorithm (this parameter depends on the chosen order ω).

The quest for optimal orders is a major topic in Bayesian networks, which is however known to be a NP-hard problem [4]. Since probabilistic MLL contain Bayesian networks, we should focus on heuristics that yield good performances in most cases.

▶ Remark 54. Recall the proof-net $\mathcal{N}$ in Figure 1e which cut reduces to $\mathcal{N}_0$. Notice that this proof-net does not resemble to a Bayesian network, e.g. it alternates par and tensor nodes. However, the reader may recognise the intermediate factors ρ_1, ρ_2 and ρ_3 computed by Algorithm 1 in Example 52 as the nested sub proof-nets of $\mathcal{N}$ of conclusions, respectively, $X_2^+ \otimes (X_2^- \parr X_3^+)$, $X_2^+ \otimes X_4^+$ and $X_4^+ \otimes X_5^+$. This is far from being a coincidence, as any run of Algorithm 1 can in fact be associated with a MLL proof-net, although this latter might need formulas with an arbitrary number of alternations between tensors and pars. We will investigate this point in a forthcoming paper.

▶ Remark 55. The component renaming $\ell_{\mathcal{N}}$ is omitted in the setting of Bayesian networks as encoded in the formula labelling, by imposing the following type constraint:

($\star$) any two edges of $\mathcal{N}$ which are supported by the same propositional variable lay in the same connected component of $\mathcal{N}^{\neg \mathsf{b}}$.

If $(\star)$ holds (e.g. as for the proof-net $\mathcal{N}_a$ discussed in Example 52), then $\ell_{\mathcal{N}}$ is equivalent to the renaming from the edge set-family to the variable set-family, an instance being given by the renaming f mentioned in Example 33. Such a shortcut is however misleading in our setting, as the rules generating MLL proof-nets (Figure 1b) are more "granular" than the ones for Bayesian networks, in particular $(\star)$ is not preserved by the $\mathrm{c}(\mathcal{N}_{X^-,X^-})$ rule (Example 41). A better alternative would be to introduce "term" variables, like the variables of simply typed λ-calculus, decorating edges.

6 The General Case

The results of the previous section can be extended to non-atomic proof-nets by using MLL cut-reduction. We just sketch here the main ideas, giving the details in the appendix. The reader can recall the proof-net $\mathcal{N}$ in Figure 1e to follow the reasoning with an example. Given a MLL proof-net $\mathcal{N}$ of conclusion Δ: (i) reduce $\mathcal{N}$ to its normal form $\mathcal{N}_0$ by using the cut-reduction rules of Figure 1d. (ii) Decompose $\mathcal{N}_0$ into the syntax forest $\mathcal{F}_\Delta$ of its conclusions and the atomic sub proof-net $\mathcal{N}_a$ of conclusions the atomic formulas $\mathsf{At}(\Delta)$ appearing in Δ, which are the leaves of $\mathcal{F}_\Delta$. Notice that there is a bijection between $[\![\Delta]\!]$ and $[\![\mathsf{At}(\Delta)]\!]$, relating an element in $\vec{d} \in [\![\Delta]\!]$ with a tuple $\mathsf{At}(\vec{d}) \in [\![\mathsf{At}(\Delta)]\!]$ enumerating the atomic components of $\vec{d}$. (iii) Apply Algorithm 1 in order to compute $\ell_{\overline{\mathcal{N}}}([\![\overline{\mathcal{N}}]\!])$. We have:

▶ **Corollary 56.** *Let $\mathcal{N}$ be a proof-net with conclusions Δ, and let $\mathcal{N}_0$ be the normal form of $\mathcal{N}$ and $(\mathcal{F}_\Delta, \mathcal{N}_a)$ be the decomposition of $\mathcal{N}_0$ described above. For every $\vec{d} \in [\![\Delta]\!]$, we have:*

$$[\![\mathcal{N}]\!]^{\iota}_{\vec{d}} = \ell_{\mathcal{N}}([\![\mathcal{N}_a]\!])_{\ell^{\bullet}_{\mathcal{N}_a}(\mathsf{At}(\vec{d}))}, \tag{5}$$

if $\mathsf{At}(\vec{d})$ agrees on $\ell_{\mathcal{N}_a}$, otherwise $[\![\mathcal{N}]\!]^{\iota}_{\vec{d}} = 0$.

The cut-reduction in step (i) is linear in the size of $\mathcal{N}_0$ as MLL cut-reduction shrinks the size of a proof-net. Also the construction of $\mathcal{N}_a$ out of $\mathcal{N}_0$, and the read of $\ell^{\bullet}_{\mathcal{N}_a}(\mathsf{At}(\vec{d}))$ out of $\vec{d} \in [\![\Delta]\!]$ are linear. So all the complexity of this procedure is the calculation of $\ell_{\mathcal{N}}([\![\mathcal{N}_a]\!])$ which has been discussed in the previous section.

7 Conclusion and Perspectives

We considered weighted relational semantics just as an instance of quantitative semantics, but these techniques can be applied verbatim to other web-based semantics, such as probabilistic coherence spaces [6] or finiteness or Köthe sequence spaces [11, 10].

One can wonder whether our results extend to richer linear logic fragments. The additive connectives $\oplus$ and & can be reasonably added to the picture. In fact, by adopting some form of additive boxes [9], one can revisit the sum-product algorithm as a refactorization of a proof-net modulo commutative additive cuts. The exponential modalities (so encompassing full simply typed probabilistic λ-calculus) are more challenging as they require infinite sets at the semantical level. This will deserve future investigation.

Related to the above point is the correspondence alluded to in Remark 54. We will detail in a future work how to map any run ρ of the sum-product algorithm on an atomic proof-net $\mathcal{N}_0$ into a non-atomic proof-net $\mathcal{N}_\rho$, which rewrites into $\mathcal{N}_0$ and such that the intermediate factors appearing in ρ correspond to sub-proof-nets of $\mathcal{N}_\rho$.

As mentioned by Remark 53, the performance of many exact inference algorithms, such as sum-product, depends on the order of the components to eliminate and the problem of finding optimal orders is known to be NP-hard [4]. Many heuristics have been given based on the graph-theoretical structure of Bayesian nets. One can wonder whether the additional proof-theoretical structure (e.g. switching paths, empires [18]) can suggest new heuristics.

References

1 Nick Benton, Gavin Bierman, Valeria de Paiva, and Martin Hyland. Linear lambda-calculus and categorical models revisited. In E. Börger, G. Jäger, H. Kleine Büning, S. Martini, and M. Richter, editors, *Proceedings of the Sixth Workshop on Computer Science Logic*, pages 61–84. Springer Verlag, 1993. URL: citeseer.ist.psu.edu/benton92linear.html.

2 Antonio Bucciarelli and Thomas Ehrhard. On phase semantics and denotational semantics: the exponentials. *Ann. Pure Appl. Logic*, 109(3):205–241, 2001.

3 Kenta Cho and Bart Jacobs. Disintegration and bayesian inversion via string diagrams. *Math. Struct. Comput. Sci.*, 29(7):938–971, 2019. doi:10.1017/S0960129518000488.

4 Gregory F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2):393–405, 1990. doi:10.1016/0004-3702(90)90060-D.

5 Raphaëlle Crubillé. Probabilistic stable functions on discrete cones are power series. 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 275–284. ACM, 2018. doi:10.1145/3209108.3209198.

6 Vincent Danos and Thomas Ehrhard. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Information and Computation*, 209(6):966–991, 2011.

7 Adnan Darwiche. Bayesian networks. In Frank van Harmelen, Vladimir Lifschitz, and Bruce W. Porter, editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 467–509. Elsevier, 2008. doi:10.1016/S1574-6526(07)03011-8.

8 Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009. URL: http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521884389.

9 Lorenzo Tortora de Falco. The additive mutilboxes. *Ann. Pure Appl. Log.*, 120(1-3):65–102, 2003. doi:10.1016/S0168-0072(02)00042-8.

10 Thomas Ehrhard. On Köthe sequence spaces and linear logic. *Math. Struct. Comput. Sci.*, 12:579–623, 2002.

11 Thomas Ehrhard. Finiteness spaces. *Math. Struct. Comput. Sci.*, 15(4):615–646, 2005.

12 Thomas Ehrhard. Differentials and distances in probabilistic coherence spaces. 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 17:1–17:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSCD.2019.17.

13 Thomas Ehrhard. Cones as a model of intuitionistic linear logic. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 370–383. ACM, 2020. doi:10.1145/3373718.3394758.

14 Thomas Ehrhard, Michele Pagani, and Christine Tasson. Probabilistic Coherence Spaces are Fully Abstract for Probabilistic PCF. In P. Sewell, editor, *The 41th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL14, San Diego, USA*. ACM, 2014. doi:10.1145/2535838.2535865.

15 Thomas Ehrhard, Michele Pagani, and Christine Tasson. Full abstraction for probabilistic pcf. *J. ACM*, 65(4), April 2018. doi:10.1145/3164540.

16 Thomas Ehrhard, Michele Pagani, and Christine Tasson. Measurable cones and stable, measurable functions: a model for probabilistic higher-order programming. *PACMPL*, 2(POPL):59:1–59:28, 2018. doi:10.1145/3158147.

17 Thomas Ehrhard and Christine Tasson. Probabilistic call by push value. *Log. Methods Comput. Sci.*, 15(1), 2019. doi:10.23638/LMCS-15(1:3)2019.

18 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

19 Jean-Yves Girard. A new constructive logic: classical logic. *Math. Struct. Comput. Sci.*, 1(3):255–296, 1991.

20 Jean-Yves Girard. Coherent banach spaces: a continuous denotational semantics. *Theor. Comput. Sci.*, 227:297, 1999.

21 Jean-Yves Girard. Between logic and quantic: a tract. In Thomas Ehrhard, Jean-Yves Girard, Paul Ruet, and Philip Scott, editors, *Linear Logic in Computer Science*, volume 316 of *London Math. Soc. Lect. Notes Ser.* CUP, 2004.

22 Bart Jacobs, Aleks Kissinger, and Fabio Zanasi. Causal inference by string diagram surgery. In Mikolaj Bojanczyk 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 313–329. Springer, 2019. doi:10.1007/978-3-030-17127-8_18.

23 Yves Lafont. From proof nets to interaction nets. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Math. Soc. Lect. Notes Ser.*, pages 225–247, 1995.

24 Jim Laird, Giulio Manzonetto, Guy McCusker, and Michele Pagani. Weighted relational models of typed lambda-calculi. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013.* IEEE Computer Society, June 2013.

25 François Lamarche. Quantitative domains and infinitary algebras. *Theor. Comput. Sci.*, 94(1):37–62, 1992. doi:10.1016/0304-3975(92)90323-8.

26 Hugo Paquet. Bayesian strategies: probabilistic programs as generalised graphical models. In Nobuko Yoshida, editor, *Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings*, volume 12648 of *Lecture Notes in Computer Science*, pages 519–547. Springer, 2021. doi:10.1007/978-3-030-72019-3_19.

27 Judea Pearl. *Probabilistic reasoning in intelligent systems - networks of plausible inference.* Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1989.

28 Dario Stein and Sam Staton. Compositional semantics for probabilistic programs with exact conditioning. *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13, 2021.

29 N. Zhang and D. Poole. A simple approach to bayesian network computations. In *Proceedings of the 10th Biennial Canadian Artificial Intelligence Conference*, pages 171–178. AAAI Press / The MIT Press, 1994.

Generalized Newman's Lemma for Discrete and Continuous Systems

Ievgen Ivanov ✉
Taras Shevchenko National University of Kyiv, Ukraine

Abstract

We propose a generalization of Newman's lemma which gives a criterion of confluence for a wide class of not-necessarily-terminating abstract rewriting systems. We show that ordinary Newman's lemma for terminating systems can be considered as a corollary of this criterion. We describe a formalization of the proposed generalized Newman's lemma in Isabelle proof assistant using HOL logic.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Theory of computation → Logic and verification; Theory of computation → Timed and hybrid models

Keywords and phrases abstract rewriting system, confluence, discrete-continuous systems, proof assistant, formal proof

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.9

Supplementary Material *Text*: `https://doi.org/10.5281/zenodo.7855691` [14]

1 Introduction

Newman's lemma [20, 10, 15, 2, 17] is a mathematical result that is well-known in computer science community and that is usually associated with analysis of discrete structures and the principle of well-founded[1], or, dually, Noetherian induction. However, Noetherian induction has a generalization to Raoult's open induction principle [24] which can be considered as an interesting example of unification of proof principles important for analysis of discrete and continuous structures. More specifically, both the Noetherian induction principle for proving properties of elements of a Noetherian poset and a variant of a real induction principle [4] for proving properties of real numbers in a bounded closed interval can be interpreted as applications of the open induction principle. Moreover, it is known [23] that real induction is relevant to analysis of continuous-time dynamical systems defined using ordinary differential equations.

Taking into account the above mentioned remarks, it is natural to consider the question of whether Newman's lemma can be generalized in such a way that its generalization can be applied to a range of discrete, continuous, and discrete-continuous dynamical models (e.g. [8]). Note that discrete-continuous (hybrid) models become increasingly important for computer science with the spread of such concepts as cyber-physical systems [3], Internet of Things, etc., and although mathematical systems theory and control theory study a variety of models and properties (e.g. reachability, stability, controllability, etc.) that have certain correspondences with computation-related notions considered in computer science, the task of combining modeling and reasoning approaches from different fields remains non-trivial.

[1] For example, as of the time of writing, the article on Newman's lemma in English Wikipedia [21] mentions: "*Today, this is seen as a purely combinatorial result based on well-foundedness due to a proof of Gérard Huet in 1980.*"

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 9; pp. 9:1–9:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

And to simplify this task one can search for such generalizations of important results from one field (e.g. computer science) that they begin to directly overlap with another field (e.g. control theory) and vice versa.

In literature one can find confluence conditions that can be considered as generalizations of Newman's lemma for certain classes of abstract rewriting systems (ARS) [26, 5, 6, 7]. A notable example is Van Oostrom's theorem [26, Theorem 3.7] and its corollary that any Decreasing Church-Rosser (DCR) ARS is confluent. However, most of such conditions are linked in some way to auxiliary well-foundness, countability, and/or cofinality assumptions that are more relevant to analysis of discrete models of computation or discrete-continuous dynamical models with limited nondeterminism.

In this paper we propose a different approach to the mentioned question which makes use of a topology on the preordered set $(X, \to^*)$ associated with an ARS $(X, \to)$ in a way reminiscent to how it is used in Raoult's open induction principle, where $\to^*$ denotes the reflexive transitive closure of a reduction relation $\to$. Using this approach we obtain a generalized Newman's lemma (Theorem 28 in Section 3) which gives a necessary and sufficient condition of confluence for the class of ARS $(X, \to)$ such that $(X, \to^*)$ is a strictly inductive [18] preordered set (i.e. a preordered set where every nonempty chain has a least upper bound). Ordinary Newman's lemma for terminating systems can be considered as a corollary of the proposed criterion (Corollary 29 from Theorem 28).

As a proof of the main result we give a machine-checked formal proof of a formalized statement of generalized Newman's lemma in Isabelle proof assistant using HOL logic [11, 22, 27] in supplementary material [14] for this paper. Note that this proof is not based on direct application of either Noetherian or open induction, however, Noetherian induction and a suitably adapted open induction principle can be used to characterize classes of ARS for which the ordinary and generalized Newman's lemmas can be used as confluence criteria (Propositions 7 and 14 in Section 2).

Isabelle is a generic proof assistant software with a small logical core that provides a meta-logic in which several object logics are encoded. Supported object logics include, in particular, higher-order logic (HOL) and Zermelo-Fraenkel set theory (ZF). They can be used to formalize statements and proofs from pure mathematics, but they also have applications in the domain of formal specification and verification of systems and software. A user can introduce new definitions and formulate statements (lemmas, theorems) using special formal notation. In simple cases, a proof of a valid statement can be obtained automatically by calling automated theorem provers, but in most non-trivial cases a user needs to guide the system using a proof script or a structured proof text, so that a complete formal proof can be constructed automatically from such a script/text.

Note that there exist other formalizations of ARS-related notions and results in Isabelle, e.g. [25, 1, 29], however, our formalization does not depend on them. Our formalization depends only on standard theories included in Isabelle distribution.

2 Preliminaries

Below we give definitions of the notions which we need to formulate and discuss the main result. Definitions and propositions given in this section and Theorem 28 in Section 3 (main result) are formulated using ordinary mathematical notation. Propositions 7-27 are accompanied by non-formalized proof sketches which explain main proof ideas. Such formulations and proof sketches are intended to simplify understanding of the main result and omit low-level details of our Isabelle formalization. A reader can assume that a background theory for understanding them is ZFC, however, our Isabelle formalization is based on HOL.

Formalized versions of Definitions 1-24 and statements of Propositions 7-27 and Theorem 28 for proof assistant software, as well as complete formal proof texts are given in [14] (supplementary material for this paper). Formal proofs can be checked automatically using Isabelle 2022 software [11].

2.1 Abstract rewriting systems

Let us recall several standard notions that appear in literature on rewriting systems (e.g. [15, 2, 17]). Note that we assume that the axiom of choice holds.

We will denote logical negation, disjunction, conjunction, and implication as $\neg$, $\vee$, $\wedge$, and $\Rightarrow$ respectively.

▶ **Definition 1.** *An abstract rewriting system (ARS) is a pair $(X, \to)$, where X is a set and $\to$ is a binary relation on X (called reduction).*

Note that some authors define an ARS to be a pair of a set and an indexed family of reduction relations $(\to_i)_{i\in I}$. We do not use this approach in this paper and instead restrict attention to ARS with a single reduction relation $\to$. Also, we allow X to be empty.

Let $(X, \to)$ be an ARS. Denote as $\to^+$ the transitive closure of $\to$, and denote as $\to^*$ the reflexive transitive closure of $\to$.

▶ **Definition 2.** *Let $x \in X$. Then*

(1) *x is reducible, if there exists $x' \in X$ such that $x \to x'$*

(2) *x is irreducible, if x is not reducible*

(3) *$x' \in X$ is a normal form of x, if $x \to^* x'$ and x' is irreducible.*

▶ **Definition 3.** *An ARS $(X, \to)$ is*

(1) *(weakly) normalizing, if for each $x \in X$ there exists $x' \in X$ such that x' is a normal form of x*

(2) *terminating (or, alternatively, strongly normalizing), if there is no infinite reduction sequence $x_1 \to x_2 \to \ldots$ $(x_i \in X)$.*

▶ **Definition 4.** *An ARS $(X, \to)$ is*

(1) *confluent, if*

$$\forall a, b, c \in X\ (a \to^* b \wedge a \to^* c \Rightarrow \exists d \in X\ (b \to^* d \wedge c \to^* d))$$

(2) *locally confluent, if*

$$\forall a, b, c \in X\ (a \to b \wedge a \to c \Rightarrow \exists d \in X\ (b \to^* d \wedge c \to^* d)).$$

2.2 Noetherian induction

It is known that the condition that an ARS $(X, \to)$ is terminating can be characterized in terms of soundness of a variant of Noetherian induction principle which can be used to show that a given property $P(x)$ holds for all $x \in X$.

We will use a variant of Noetherian induction principle similar to the one given in [17, paragraph 1.3.15], but will reformulate it in extensional form by assuming that the mentioned property $P(x)$ is represented as $x \in S$, where S is a set.

▶ **Definition 5.** *S is a Noetherian-inductive subset in ARS $(X, \to)$, if $S \subseteq X$ and*

$$\forall x \in X\ ((\forall y \in X\ (x \to^+ y \Rightarrow y \in S)) \Rightarrow x \in S).$$

▶ **Definition 6.** *$(X, \to)$ has sound Noetherian induction principle, if for every $S \subseteq X$, if S is a Noetherian-inductive subset in ARS $(X, \to)$, then $S = X$.*

▶ **Proposition 7.** *For any ARS $(X, \to)$ the following conditions are equivalent:*

1. *$(X, \to)$ is terminating*
2. *$(X, \to)$ has sound Noetherian induction principle.*

Proof sketch. Similar to the proof of [17, Theorem 1.3.16]. ◀

2.3 Strictly inductive ARS

Let $(X, \leq)$ be a preordered set (so $\leq$ is a reflexive and transitive binary relation on X).

▶ **Definition 8.** *Let $A \subseteq X$ and $x \in X$. Then x is*

(1) *an upper bound of A, if $\forall a \in A \; a \leq x$*
(2) *a least element of A, if $x \in A \wedge \forall a \in A \; x \leq a$*
(3) *a least upper bound of A, if x is a least element of the set of all upper bounds of A.*

▶ **Definition 9.** *A subset $A \subseteq X$ is*

(1) *a chain (in $(X, \leq)$), if $\forall x, y \in A \; (x \leq y \vee y \leq x)$*
(2) *closed (in $(X, \leq)$), if for every nonempty chain C in $(X, \leq)$, if C has a least upper bound $x \in X$ and $C \subseteq A$, then $x \in A$*
(3) *open (in $(X, \leq)$), if $X \backslash A$ is closed in $(X, \leq)$*
(4) *relatively open in B w.r.t. preorder relation $\leq$, where $B \subseteq X$ is a superset of A, if A is open in $(B, \leq \cap (B \times B))$.*

▶ Remark. A Scott-open subset in a poset is open in the above mentioned sense, but the converse may not hold. More information on topologies on ordered sets can be found in [9].

Some examples that illustrate Definition 9 are given below.

If $X = [0, 1]$, where $[0, 1]$ denotes the real unit interval, and $\leq$ is the standard order on real numbers restricted to X, then

- every subset of X is a chain
- the set $\{1\}$ is closed: every nonempty chain in $\{1\}$ contains only 1, so its supremum is 1, and $1 \in \{1\}$
- the set $\{1\}$ is *not* open: the element $1 \in \{1\}$ can be approached from below using a nonempty chain of elements outside $\{1\}$, e.g. 0.9, 0.99, 0.999, ...
- the set $(0, 1)$ is open: no number in $(0, 1)$ can be approached from below using a nonempty chain that has no elements in $(0, 1)$.

If X is the set of all finite strings in the alphabet $\{0, 1\}$ (e.g. $01010 \in X$), and $s \leq s'$ if and only if s is a prefix of s' (e.g. $01 \leq 010$, but $01 \not\leq 110$), then

- the set $\{0, 00\}$ is a chain
- the set $\{0, 1\}$ is *not* a chain: the elements 0, 1 are incomparable
- the set $\{1\}$ is closed and open.

▶ **Definition 10.** *A preordered set $(X, \leq)$ is strictly inductive, if every nonempty chain in $(X, \leq)$ has a least upper bound (i.e. for every nonempty chain C there exists $x \in X$ such that x is a least upper bound of C).*

The term "strictly inductive" is adapted from [18]. Note that a dcpo is a strictly inductive preordered set.

▶ **Definition 11.** *An ARS* $(X, \to)$ *is*

(1) *strictly inductive, if* $(X, \to^*)$*, considered as a preordered set, is strictly inductive*

(2) *acyclic [7], if for each* $x, y \in X$*, if* $x \to^+ y$*, then* $x \neq y$*.*

▶ Remark. In some literature, e.g. [15], an *inductive* ARS is defined as an ARS where for every reduction sequence $x_1 \to x_2 \to ...$ there exists $x \in X$ such that $x_n \to^* x$ for all n. In general, this condition is weaker than the condition that $(X, \to^*)$ is a strictly inductive preordered set, so our terminology is consistent with such literature: a strictly inductive ARS is inductive in the mentioned sense, but the converse may not hold.

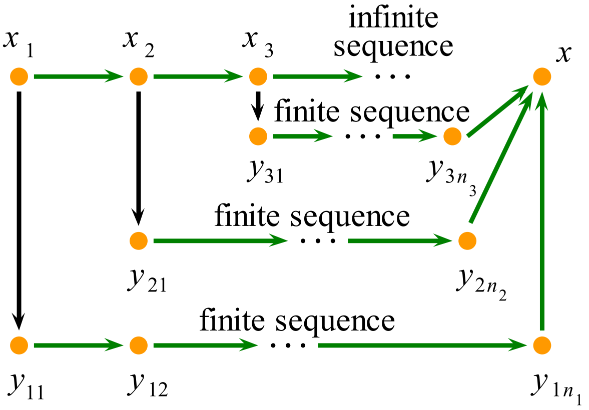


Figure 1 Illustration of the notion of an *inductive* ARS: for every reduction sequence $x_1 \to x_2 \to ...$ there exists $x \in X$ such that $x_n \to^* x$ for all n. Such an element x is an upper bound of $\{x_1, x_2, ...\}$ in the preordered set $(X, \to^*)$. The notion of a *strictly indutive* ARS further requires that $\{x_1, x_2, ...\}$ has a least upper bound, and, moreover, every nonempty chain in $(X, \leq)$ has a least upper bound.

▶ **Example 12.** Let $(X, \to) = ([0, 1], < \cap ([0, 1] \times [0, 1]))$, where $[0, 1]$ is the real unit interval and $<$ is the standard strict order on real numbers. Then $(X, \to)$ is a strictly inductive ARS. Moreover, $(X, \to)$ is acyclic.

▶ **Definition 13.** *An ARS* $(X, \to)$ *has sound open induction principle, if for every open* $S \subseteq X$ *in the preordered set* $(X, \to^*)$*, if* S *is a Noetherian-inductive subset in ARS* $(X, \to)$*, then* $S = X$*.*

▶ **Proposition 14.** *For any ARS* $(X, \to)$ *the following conditions are equivalent:*

1. $(X, \to)$ *is a strictly inductive and acyclic ARS*
2. $(X, \to)$ *has sound open induction principle.*

Proof sketch. A proof that Item 1 implies Item 2 is analogous to a proof of Raoult's theorem [24, Theorem 3.3]. A proof that Item 2 implies Item 1 consists of two parts.

- A proof that an ARS with sound open induction principle is strictly inductive is analogous to the proof of a converse open induction principle proposed in [13, Theorem 1].
- A proof that an ARS with sound open induction principle is acyclic can be performed by contradiction (by assuming that $x \to^+ x$ holds for some $x \in X$ and considering $S = X \backslash \{y \in X \mid y \to^* x\}$ in Definition 13). ◀

▶ **Proposition 15.** *Any terminating ARS is strictly inductive and acyclic.*

Proof sketch. Follows immediately from Propositions 7, 14 by noting that an ARS with sound Noetherian induction principle also has sound open induction principle (see Definitions 6 and 13). ◀

▶ **Proposition 16.** *If $(X, \rightarrow)$ is an ARS and X is finite, then $(X, \rightarrow)$ is strictly inductive.*

Proof sketch. Follows from Definition 10 by noting that a nonempty finite chain in a preordered set has a least upper bound. ◀

2.4 Quasi-normal forms

In contrast to terminating ARS, strictly inductive ARS are not necessarily weakly normalizing (e.g., the ARS ({0},{(0,0)}) is strictly inductive, but is not weakly normalizing). However, the following notions can be used to preserve a certain similarity between the studies of properties of terminating and strictly inductive ARS in the general case.

▶ **Definition 17.** *Let $(X, \rightarrow)$ be an ARS and $x, x' \in X$. Then*

(1) *x is quasi-irreducible, if for each $y \in X$, $x \rightarrow^* y$ implies $y \rightarrow^* x$*

(2) *x' is a quasi-normal form of x (or, shortly, x' is a QNF of x), if $x \rightarrow^* x'$ and x' is quasi-irreducible*

(3) *x, x' are QNF-equivalent, if*

$$\{y \in X \mid y \textit{ is a QNF of } x\} = \{y \in X \mid y \textit{ is a QNF of } x'\}.$$

▶ **Definition 18.** *An ARS $(X, \rightarrow)$ is*

(1) *quasi-normalizing, if for each $x \in X$ there exists $x' \in X$ such that x' is a quasi-normal form of x.*

(2) *openly quasi-normalizing, if $(X, \rightarrow)$ is quasi-normalizing and for each $x, x' \in X$ such that x' is a quasi-normal form of x, the set*

$$\{y \in X \mid x \rightarrow^* y \ \wedge \ y \textit{ and } x' \textit{ are QNF-equivalent }\}$$

is relatively open in $\{y \in X \mid x \rightarrow^* y\}$ w.r.t. preorder relation $\rightarrow^*$.

The quasi-normalization condition has an obvious analogy with the (weak) normalization condition. In the general case, open quasi-normalization condition is a stroger requirement, the importance of which for checking confluence will be shown below in Theorem 28 and Example 31 (in particular, the ARS given in Example 31 is strictly inductive and quasi-normalizing, but is neither openly quasi-normalizing, nor confluent, and in this example the lack of confluence is linked to the lack of open quasi-normalization).

▶ **Proposition 19.** *Any strictly inductive ARS is quasi-normalizing.*

Proof sketch. Follows from Zorn's lemma. ◀

▶ **Proposition 20.** *Let $(X, \rightarrow)$ be a terminating ARS. Then for any sets A, B such that $A \subseteq B \subseteq X$, A is relatively open in B w.r.t. preorder relation $\rightarrow^*$.*

Proof sketch. Follows from Definition 9 by noting that for a terminating $(X, \rightarrow)$, a nonempty chain C in $(X, \rightarrow^*)$ has a greatest element (i.e. $x \in C$ such that $\forall c \in C \ c \rightarrow^* x$). ◀

▶ **Proposition 21.** *Any terminating ARS is openly quasi-normalizing.*

Proof sketch. Follows immediately from Propositions 15, 19, 20 and Definition 18. ◀

Another simple sufficient condition which guarantees that an ARS is openly quasi-normalizing is given below.

▶ **Definition 22.** *An ARS $(X, \to)$ is finitely normalizing, if it is normalizing and the set of all irreducible elements in $(X, \to)$ is finite.*

▶ **Proposition 23.** *Any finitely normalizing ARS is openly quasi-normalizing.*

Proof sketch. Follows from Definition 18 by noting that for a finitely normalizing $(X, \to)$, the set

$$\{y \in X \mid x \to^* y \ \wedge\ y \text{ and } x' \text{ are QNF-equivalent } \}$$

can be expressed as a complement of a finite union of closed sets in $\{y \in X \mid x \to^* y\}$ considered as an induced preordered set (from $(X, \to^*)$). ◀

2.5 Quasi-local confluence

The notion of local confluence loses its usefulness when a reduction relation $\to$ is reflexive and transitive, because in this case the definitions of confluence and local confluence become trivially equivalent (and equivalent to the diamond property condition). In some of such and other situations related to non-terminating ARS, the notion of quasi-local confluence introduced below can be used as a replacement for local confluence.

▶ **Definition 24.** *An ARS $(X, \to)$ is quasi-locally confluent, if for each $a \in X$ there exists a subset $S \subseteq \{x \in X \mid a \to^+ x\}$ such that the following 2 conditions hold:*

1. *two-consistency condition:*

$$\forall b, c \in S\ \exists d \in X\ (b \to^* d \wedge c \to^* d)$$

2. *coinitiality condition:*

$$\forall x \in X\ (a \to^+ x \Rightarrow (x \to^* a) \vee (\exists b \in S\ b \to^* x \wedge \neg(b \to^* a))).$$

Intuitively, a set S in Definition 24 can be thought of as a substitute for the set of successors of a: $\{x \in X \mid a \to x\}$. The latter set actually satisfies the conditions 1 and 2 in the case of locally confluent acyclic ARS as Proposition 26 below states. Basically, the conditions 1 and 2 axiomatize some properties of the mentioned set to allow one to obtain useful generalizations of confluence criteria for wider classes of ARS.

The two-consistency condition is an adaptation of the following notion of a *2-consistent set* for preordered sets to ARS: a subset $A \subseteq X$ in a preordered set $(X, \leq)$ is 2-consistent, if for each $a_1, a_2 \in A$ there exists $x \in X$ such that $a_1 \leq x$ and $a_2 \leq x$ (when $A \neq \emptyset$ this is reminiscent to the notion of a directed set, however x is not required to belong to A). Thus the two-consistency condition expresses the requirement that S is a 2-consistent set in the preordered set $(X, \to^*)$.

The coinitiality condition is an adaptation of the notion of a *coinitial subset* for preordered sets to ARS: if $(X, \leq)$ is a preordered set and $A, B \subseteq X$, then A is a coinitial subset of B, if $A \subseteq B$ and for each $b \in B$ there exists $a \in A$ such that $a \leq b$. In particular, when a reduction relation $\to$ coincides with some strict partial order $<$ on X, the coinitiality condition together with the condition $S \subseteq \{x \in X \mid a \to^+ x\}$ expresses the requirement that S is a coinitial subset of $\{x \in X \mid a < x\}$ in the poset $(X, \leq)$, where $x \leq y \Leftrightarrow (x = y \vee x < y)$.

▶ **Proposition 25.** *Any confluent ARS is quasi-locally confluent.*

Proof sketch. Assume that an ARS $(X, \to)$ is confluent. Then it is straightforward to check that for any $a \in X$, the set $S = \{x \in X \mid a \to^* x \wedge \neg(x \to^* a)\}$ satisfies the conditions of Definition 24. So $(X, \to)$ is quasi-locally confluent. ◀

▶ **Proposition 26.** *Let $(X, \to)$ be a locally confluent acyclic ARS and $a \in X$. Then $S = \{x \in X \mid a \to x\}$ satisfies the conditions 1-2 of Definition 24.*

Proof sketch. A proof is straightforward and follows from the relevant definitions. ◀

▶ **Proposition 27.** *Any locally confluent acyclic ARS is quasi-locally confluent.*

Proof sketch. Follows immediately from Definition 24 and Proposition 26. ◀

3 Main Result

▶ **Theorem 28** (Generalized Newman's lemma)**.** *Let $(X, \to)$ be a strictly inductive ARS. Then $(X, \to)$ is confluent if and only if $(X, \to)$ is openly quasi-normalizing and quasi-locally confluent.*

Proof sketch.

"If" part. Assume that $(X, \to)$ is a strictly inductive, openly quasi-normalizing, and quasi-locally confluent ARS. Let us show that $(X, \to)$ is confluent.

Let $a \in X$. To show that for any b, c with $a \to^* b \wedge a \to^* c$ there exists $d \in X$ such that $b \to^* d \wedge c \to^* d$, it is sufficient to show that all maximal elements in $X' = \{x \in X \mid a \to^* x\}$, considered as an induced preordered subset of the preordered set $(X, \to^*)$, are equivalent w.r.t. the equivalence relation $\approx$, where $x \approx x'$ if and only if $x \to^* x' \wedge x' \to^* x$.

Consider any two such maximal elements $M, M' \in X'$ (they will also be maximal elements in the preordered set $(X, \to^*)$ and quasi-normal forms in ARS $(X, \to)$) and introduce an auxiliary preordered set $(X', \leq')$, where

$$x \leq' y \Leftrightarrow (x \to^* y \vee ((\exists x' \in X' \; x \to^* x' \wedge \neg(x' \to^* M)) \wedge M' \to^* y))).$$

Then for each $x \in X'$, $x \leq' M \vee x \leq' M'$ holds.

The main reason for introducing the preordered set $(X', \leq')$ is that it simplifies further analysis of relations between M and M' (note that $(X', \leq')$ does not have any elements incomparable with both M and M').

The preordered set $(X', \leq')$ can be shown to be strictly inductive using the assumption that the ARS $(X, \to)$ is openly quasi-normalizing.

The sets $\{x \in X' \mid x \leq' M\}$, $\{x \in X' \mid x \leq' M'\}$ are closed in $(X, \leq')$. Denote their intersection as D. Then D is closed in $(X', \leq')$, and is also nonempty since $(X', \leq')$ has a least element. Since $(X, \leq')$ is strictly inductive (implicitly using Zorn's lemma) one can conclude that there is some $\leq'$-maximal element $m \in D$.

As before, for any preordered set $(P, \leq)$ we call a subset $A \subseteq P$ 2-consistent, if for each $a_1, a_2 \in A$ there exists $x \in P$ such that $a_1 \leq x \wedge a_2 \leq x$. It is straightforward to show the following criterion of 2-consistency: a subset $A \subseteq P$ is 2-consistent if and only if for any down-closed sets $X_1, X_2 \subseteq P$ the condition $P = X_1 \cup X_2$ implies $A \subseteq X_1 \vee A \subseteq X_2$.

Using the assumption that the ARS $(X, \to)$ is quasi-locally confluent one can show that there is some coinitial subset N of $\{x \in X' \mid m \leq' x \wedge \neg x \leq' m\}$ which is 2-consistent in $(X', \leq')$. The above mentioned 2-consistency criterion implies that $N \subseteq \{y \in X' \mid y \leq' M\} \vee N \subseteq \{y \in X' \mid y \leq' M'\}$.

Suppose that M and M' are not comparable w.r.t. $\leq'$. Then using coinitiality of N, there exist $b \in N$ and $c \in N$ such that $b \leq' M$ and $c \leq' M'$. As we have mentioned above, $N \subseteq \{y \in X' \mid y \leq' M\} \vee N \subseteq \{y \in X' \mid y \leq' M'\}$.

If $N \subseteq \{y \in X' \mid y \leq' M\}$, then $c \leq' M$, whence $m \leq' c \wedge \neg c \leq' m$ and $c \in D$.

If $N \subseteq \{y \in X' \mid y \leq' M'\}$, then $b \leq' M'$, whence $m \leq' b \wedge \neg b \leq' m$ and $b \in D$.

In both cases we have a contradiction with $\leq'$-maximality of m in D. Thus M and M' are comparable w.r.t. $\leq'$, which implies that $M \approx M'$.

"Only if" part. Assume that an ARS $(X, \rightarrow)$ is strictly inductive and confluent ARS. Let us show that $(X, \rightarrow)$ is openly quasi-normalizing and quasi-locally confluent.

From Proposition 19 it follows that $(X, \rightarrow)$ is quasi-normalizing. Then it is straightforward to show that $(X, \rightarrow)$ is openly quasi-normalizing by noting that in the case of a confluent $(X, \rightarrow)$, for any $x \in X$ and for any $x', x'' \in X$ which are quasi-normal forms of x, $x' \rightarrow^* x'' \wedge x'' \rightarrow^* x'$ holds. Also, $(X, \rightarrow)$ is quasi-locally confluent by Proposition 25. ◀

The following statements can be considered as corollaries from Theorem 28.

▶ **Corollary 29.** *(Newman's lemma) Any locally confluent terminating ARS is confluent.*

Proof. Follows immediately from Theorem 28 and the following propositions from Section 2:

- any terminating ARS is strictly inductive and acyclic (Proposition 15)
- any terminating ARS is openly quasi-normalizing. (Proposition 21)
- any locally confluent acyclic ARS is quasi-locally confluent (Proposition 27).

Note that straightforward proofs of these propositions do not rely on the ordinary Newman's lemma itself. ◀

▶ **Corollary 30.** *Any locally confluent ARS which is strictly inductive, acyclic and finitely normalizing is confluent.*

Proof. Follows immediately from Theorem 28 and the following propositions from Section 2:

- any finitely normalizing ARS is openly quasi-normalizing (Proposition 23)
- any locally confluent acyclic ARS is quasi-locally confluent (Proposition 27). ◀

Note that in the general case, the open quasi-normalization condition cannot be omitted from the statement of Theorem 28 as the following example shows.

▶ **Example 31.** Consider an ARS $(P, \rightarrow)$, where $P = [0,1] \times [0,1]$ (where $[0,1]$ denotes the real unit interval) and the reduction relation $\rightarrow$ is defined as follows (where components of elements of P are denoted as x and t):

$$(x,t) \rightarrow (x',t') \text{ if and only if } (x \leq x' \;\wedge\; t < t' \;\wedge\; x' - x \leq t' - t).$$

It is straightforward to check that $(P, \rightarrow)$ is strictly inductive, quasi-locally confluent, but is *not* confluent. Then $(P, \rightarrow)$ is *not* openly quasi-normalizing by Theorem 28, however, $(P, \rightarrow)$ is quasi-normalizing by Proposition 19. Thus in this example one can link the lack of confluence to the lack of open quasi-normalization. The described ARS is illustrated in Figure 2.

▶ Remark. A possible interpretation of the ARS $(P, \rightarrow)$ given in Example 31 in terms of physics (special relativity) is as follows. Consider x as space coordinate, t as time coordinate, P as a region of spacetime. The relation

$$t < t' \;\wedge\; |x' - x| \leq t' - t,$$

which can also be expressed as

$$(t' - t) > 0 \;\wedge\; c^2(t' - t)^2 - (x' - x)^2 \geq 0,$$

where $c = 1$, is a strict causal precedence between events in (1+1) dimensional Minkowski spacetime[2], so a reduction $(x, t) \to (x', t')$ is a conjunction of this relation and a spatial direction condition $x \leq x'$, restricted to P.

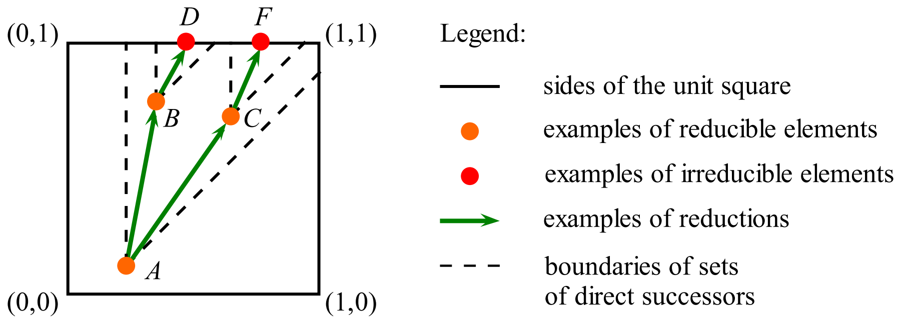


Figure 2 Illustration of the ARS $(P, \to)$ from Example 31. Horizontal axis is x, vertical axis is t.

▶ Remark. Using a slight generalization of a chain completion construction for posets given in [18], Theorem 28 may be indirectly applied to ARS which are not strictly inductive.

▶ **Example 32.** Consider an ARS $(X, \to)$, where $X = [0, 1] \times [0, 1]$ (where $[0, 1]$ denotes a real interval) and $\to \subseteq X \times X$ is a relation such that for each $x, y, x', y' \in [0, 1]$,

$(x, y) \to (x', y')$ if and only if $(x < x' \;\vee\; (x = x' \wedge y' < y \wedge (x < 1 \vee y < 1))) \wedge (x - y \leq x' - y') \wedge (x = y \Rightarrow x' = y') \wedge (x \leq y \Rightarrow x' \leq y')$.

The behavior of $\to$ is illustrated in Figure 3 below.

It is straightforward to check that this ARS is strictly inductive and acyclic and that $(1, 0)$ and $(1, 1)$ are the only irreducible elements. Then using Proposition 19 one can conclude that $(X, \to)$ is finitely normalizing, so from Proposition 23 and Theorem 28 it follows that for $(X, \to)$ confluence and quasi-local confluence conditions are equivalent. To show quasi-local confluence, for any point $a \in X$ it is sufficient to choose some set S (e.g. the set of $p \in X$ such that $a \to^+ p$ and $d(a, p) \in (0, \varepsilon)$ for a small positive number ε which may depend on a, where d is Euclidean distance) which satisfies the coinitiality condition from Definition 24 and make sure that it satisfies the two-consistency condition, i.e. pairs of elements in S can be joined using finite reduction sequences which are allowed to end outside S. Note that the given ARS $(X, \to)$ is indeed confluent.

▶ Remark. Further examples of ARS can be obtained from reachability relations on states of various (possibly nondeterministic) dynamical models. For example, consider a hybrid automaton [8] illustrated in Figure 4 that represents a simple model of motion of a bouncing ball. Note that it is usual to consider a simplified version of such an automaton with a single discrete state (Discrete state 1 in Figure 4), however, a model with a single discrete state is deterministic and has Zeno behavior [8] (semi-formally, performs infinitely many discrete

[2] Note that there exist known links [19] between this notion and models of distributed systems in the sense of computer science.

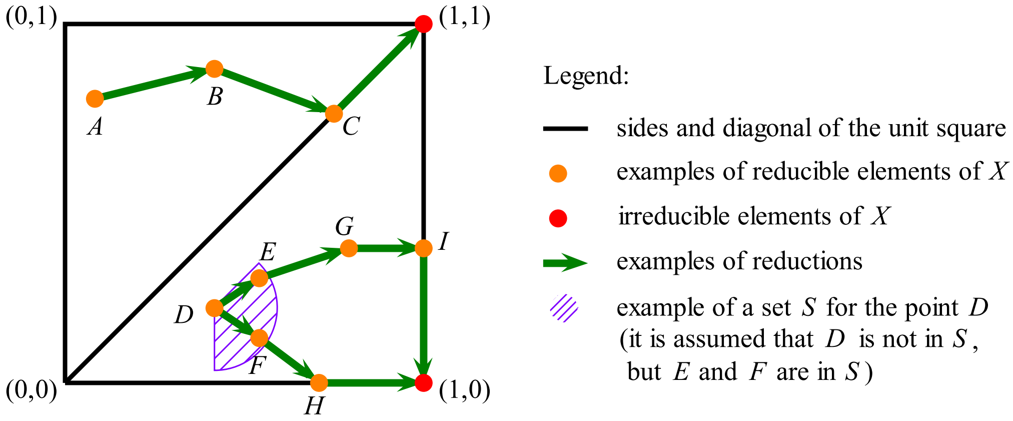

(0,1)
(1,1)
A
B
C
G
E
I
D
F
(0,0)
H
(1,0)
Legend:
sides and diagonal of the unit square
examples of reducible elements of X
irreducible elements of X
examples of reductions
example of a set S for the point D
(it is assumed that D is not in S,
but E and F are in S)

Figure 3 Illustration for Example 32. Elements of the ARS are points in the square $[0,1] \times [0,1]$. The reduction relation $\to$ is described in Example 32. Horizontal axis is x, vertical axis is y. Points above or on the diagonal shown in the figure have the point $(1,1)$ as a unique normal form. Points strictly below the diagonal shown in the figure have $(1,0)$ as a unique normal form. For this ARS quasi-local confluence implies confluence. Semi-formally, the quasi-local confluence condition for this (acyclic) ARS can be checked by making sure that for every point (x,y) (e.g. (x,y) is a point D) there exists a set S (e.g. intersection of $\{(x',y') \mid (x,y) \to^{+} (x',y')\}$ and a neighborhood of D) such that any two points in S (e.g. points E, F) can be joined using some finite reduction sequences (e.g. $E \to G \to I \to (1,0)$ and $F \to H \to (1,0)$), and any point which is reachable from (x,y) using a nonempty finite reduction sequence can be reached from some element of S using a finite (possibly empty) reduction sequence. Such a set S is a generalization of the set of direct successors of an element and the mentioned conditions replace the local confluence condition.

transitions over a bounded interval of continuous time). Since one cannot expect such a model to adequately represent the behavior of a real object when very short time intervals and distances are involved, one can introduce a new discrete state [16] (Discrete state 2, stopped ball) and a nondeterministic transition between discrete states that represents an uncertain moment when the ball stops. This does not exclude Zeno executions, but allows one to acknowledge limitations of the model and of its ability to predict the behavior of a real object (there exist other ways to address Zeno behavior issue [8], however, we will not focus on them in this paper).

With the resulting hybrid model one associate an ARS $(X, \to)$ such that $X = [0, +\infty) \times \mathbb{R}$ (continuous state space) and $(y_1, v_1) \to (y_2, v_2)$, if (y_2, v_2) can be reached from (y_1, v_1)

- either via continuous evolution within one discrete state (see Figure 6),
- or as a result of a single transition between discrete states (from Discrete state 1 to Discrete state 1 or 2).

Often, important properties of the reachability relation (bounds on reachable sets, etc.) can be determined from a model without explicitly solving the equations that determine evolution of state in time (e.g. ordinary differential equations associated with discrete states). In particular, methods related to real induction can be used to prove bounds on reachable sets [23]. In this example such methods are not required, because the model is very simple, however, it is still possible to determine that (y_2, v_2) reachable from (y_1, v_1) via continuous evolution within Discrete state 1 satisfies $\frac{1}{2}v_2^2 - \frac{1}{2}v_1^2 = g(y_1 - y_2)$ using energy conservation considerations (without solving differential equations explicitly). The obtained properties of the reduction $\to$ can be used in confluence analysis. Note that the described ARS is confluent since the final position and velocity of the ball are both zero.

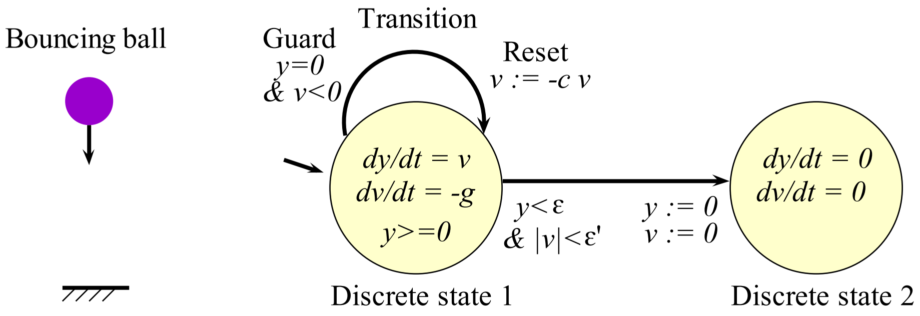


Figure 4 Illustration of a simple bouncing ball hybrid automaton with 2 discrete states. Here t is time, y is a vertical position, v is a vertical velocity, g is the gravity constant, c is a restitution coefficient, $\varepsilon, \varepsilon'$ are positive constants.

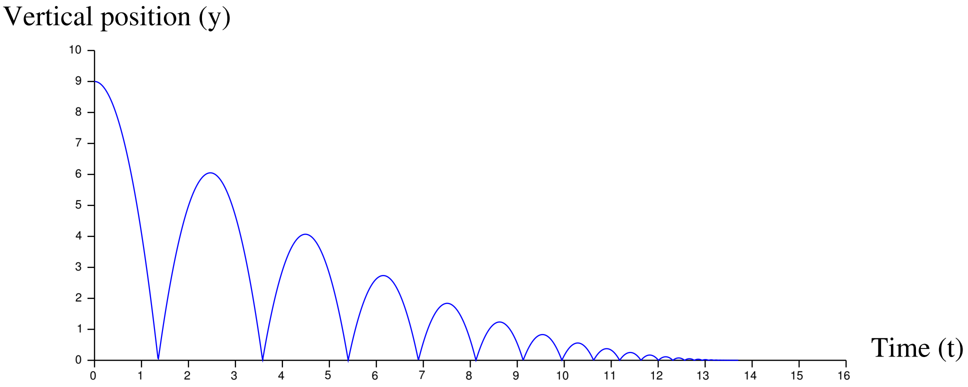


Figure 5 An example of an execution of the hybrid model illustrated in Figure 4.

4 Isabelle Formalization

Below we give formalized versions of definitions from Section 2 that are needed to formulate Theorem 28 and a formalized version of the statement of Theorem 28 (main result). Other formal definitions, Propositions 7-27, and formal proof texts for Theorem 28 and Propositions 7-27 are included in [14] (supplementary material for this paper).

The notions of an ARS and a confluent ARS are formalized as follows.

```
— Definition 1: (X,σ) is an abstract rewriting system (ARS) with a single reduction relation
— σ represents a reduction relation (->) as a predicate
— this formal definition enforces the types of X and σ and the condition -> ⊆ X×X
definition is_ars :: "'a set ⇒ ('a ⇒ 'a ⇒ bool) ⇒ bool"
("_,_ is ARS")
  where "X,σ is ARS ≡ (∀ x y. σ x y ⟶ x ∈ X ∧ y ∈ X)"

— Definition 4(1): (X,σ) is a confluent ARS
definition is_confl_ars  ("_,_ is CONFLUENT ARS")
  where "X,σ is CONFLUENT ARS ≡ (X,σ is ARS) ∧
      (∀a∈X. ∀b∈X. ∀c∈X. (σ^** a b ∧ σ^** a c ⟶
                                (∃d∈X. σ^** b d ∧ σ^** c d)))"
```

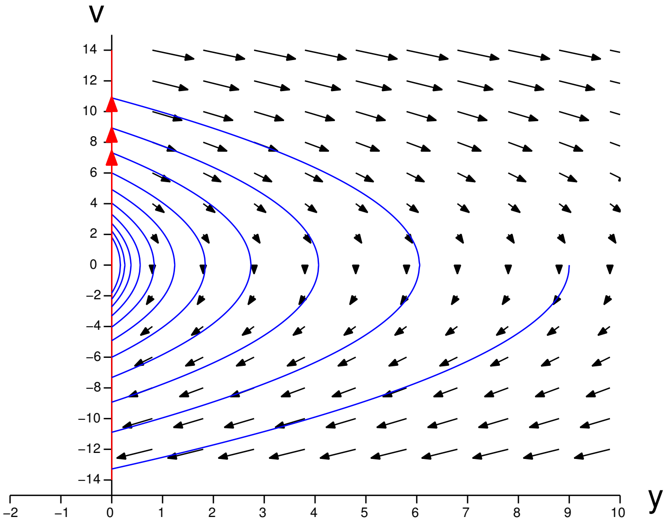


Figure 6 Continuous state space and example trajectories for the hybrid model illustrated in Figure 4 (transitions between Discrete states 1 and 2 are not illustrated).

Formalizations of basic preorder-related notions are given below.

```
— Definition 8(1): x is an upper bound of A (in preordered set (X,ϱ))
— ϱ represents a preorder relation ≤ as a predicate
— assumptions: A ⊆ X and ϱ is reflexive and transitive
abbreviation is_upperbound ::
      "'a ⇒ 'a set ⇒ 'a set ⇒ ('a ⇒ 'a ⇒ bool) ⇒ bool"
      ("_ is anUPPER BOUND of _ in _,_")
where
   "x is anUPPER BOUND of A in X,ϱ ≡ x∈X ∧ (∀a∈A. ϱ a x)"

— Definition 8(2): x is a least element of A (in preordered set (X,ϱ))
— assumptions: A ⊆ X and ϱ is reflexive and transitive
abbreviation is_leastelem ::
      "'a ⇒ 'a set ⇒ 'a set ⇒ ('a ⇒ 'a ⇒ bool) ⇒ bool"
("_ is LEAST of _ in _,_")
where
   "x is LEAST of A in X,ϱ ≡ x∈A ∧ (∀a∈A. ϱ x a)"

— Definition 8(3): x is a least upper bound of A (in preordered set (X,ϱ))
— assumptions: A ⊆ X and ϱ is reflexive and transitive
definition is_lub ("_ is l.u.b. of _ in _,_")
where
   "x is l.u.b. of A in X,ϱ ≡ (x is LEAST of
      {u. u is anUPPER BOUND of A in X,ϱ} in X,ϱ)"

— Definition 9(1): A is a chain (in preordered set (X,ϱ))
— assumptions: ϱ is reflexive and transitive
definition is_achain ("_ is aCHAIN in _,_")
```

```
  where "A is aCHAIN in X,ϱ ≡ A⊆X ∧ (∀x∈A. ∀y∈A. ϱ x y ∨ ϱ y x)"

— Definition 9(2): A is a closed subset (in preordered set (X,ϱ))
— assumptions: ϱ is reflexive and transitive
definition is_cclosedset ("_ is CLOSED SUBSET in _,_")
where
  "A is CLOSED SUBSET in X,ϱ ≡ A ⊆ X ∧
      (∀ C x. (C is aCHAIN in X,ϱ) ∧ C ≠ {} ∧
      (x is l.u.b. of C in X,ϱ) ∧ C ⊆ A ⟶ x ∈ A)"

— Definition 9(3): A is an open subset (in preordered set (X,ϱ))
— assumptions: ϱ is reflexive and transitive
definition is_copenset ("_ is OPEN SUBSET in _,_")
where
  "A is OPEN SUBSET in X,ϱ ≡ A ⊆ X ∧ ((X-A) is CLOSED SUBSET in X,ϱ)"

— Definition 9(4): A is relatively open in B w.r.t. preorder ϱ
— assumptions: ϱ is reflexive and transitive
definition is_relopenset ("_ is REL.OPEN SUBSET in _ wrt _")
where
  "A is REL.OPEN SUBSET in B wrt ϱ ≡
      A is OPEN SUBSET in B,(λ x y. ϱ x y ∧ x ∈ B ∧ y ∈ B)"

— Definition 10: (X,ϱ) is a strictly inductive preordered set
— ϱ represents a preorder relation ≤ as a predicate
— this formal definition enforces that ≤ is reflexive and transitive
— but does not enforce that ≤ ⊆ X×X
definition is_siprset ("_,_ is S.I.PREORDERED")
where
  "X,ϱ is S.I.PREORDERED ≡
    (∀x∈X. ϱ x x) ∧
     (∀x∈X. ∀y∈X. ∀z∈X. ϱ x y ∧ ϱ y z ⟶ ϱ x z) ∧
     (∀C. (C is aCHAIN in X,ϱ) ∧ C ≠ {} ⟶ (∃x. (x is l.u.b. of C in X,ϱ)))"

— Definition 11(1): (X,σ) is a strictly inductive ARS
— Note that σ^** includes a diagonal relation which may relate
— elements which do not belong to X, but which are of the same type as members of X.
— However, existence of such elements is not significant for this formal definition.
definition is_si_ars ("_,_ is S.I.ARS")
  where "X,σ is S.I.ARS ≡ (X,σ is ARS) ∧ (X,(σ^**) is S.I.PREORDERED)"
```

The notions related to quasi-normal forms are formalized as follows.

```
— Definition 17(1): x is quasi-irreducible in (X,σ)
— assumptions: (X,σ) is ARS and x ∈ X
abbreviation is_qirreduc :: "'a ⇒ 'a set ⇒ ('a ⇒ 'a ⇒ bool) ⇒ bool"
("_ is Q.IRREDUCIBLE in _,_")
  where "x is Q.IRREDUCIBLE in X,σ ≡ ∀y∈X. σ^** x y ⟶ σ^** y x"

— Definition 17(2): x' is a quasi-normal form of x in (X,σ)
— assumptions: (X,σ) is ARS and x ∈ X
definition is_qnform  ("_ is Q.N.F. of _ in _,_")
  where "x' is Q.N.F. of x in X,σ ≡
        x'∈X ∧ σ^** x x' ∧ (x' is Q.IRREDUCIBLE in X,σ)"
```

```
— Definition 17(3): x,x' are quasi-equivalent in (X,σ)
— assumptions: (X,σ) is ARS and x ∈ X and x' ∈ X
definition are_qequiv  ("_,_ are QNF-EQUIVALENT in _,_")
  where "x,x' are QNF-EQUIVALENT in X,σ ≡
           {y∈X. y is Q.N.F. of x in X,σ} = {y∈X. y is Q.N.F. of x' in X,σ}"

— Definition 18(1): (X,σ) is a quasi-normalizing ARS
definition is_qn_ars  ("_,_ is Q.N.ARS")
  where "X,σ is Q.N.ARS ≡ (X,σ is ARS) ∧
            (∀x∈X. ∃x'∈X. (x' is Q.N.F. of x in X,σ))"

— Definition 18(2): (X,σ) is an openly quasi-normalizing ARS
definition is_oqn_ars  ("_,_ is O.Q.N.ARS")
  where "X,σ is O.Q.N.ARS ≡ (X,σ is Q.N.ARS) ∧
           (∀x∈X. ∀x'∈X. (x' is Q.N.F. of x in X,σ) ⟶
            ({y∈X. σ^** x y ∧ (y,x' are QNF-EQUIVALENT in X,σ)}
              is REL.OPEN SUBSET in {y∈X. σ^** x y} wrt (σ^**)))"
```

Quasi-local confluence is formalized as follows.

```
— A set S satisfies the two-consistency condition in ARS (X,σ)
— assumptions: (X,σ) is ARS and S ⊆ X
definition sat_two_consist_cond ("_ sat.2-CONSISTENCY in _,_")
where
  "S sat.2-CONSISTENCY in X,σ ≡ (∀b∈S. ∀c∈S. ∃d∈X. σ^** b d ∧ σ^** c d)"

— A set S satisfies the coinitiality condition in ARS (X,σ) w.r.t. element a
— assumptions: X,σ is ARS and S ⊆ X and a ∈ X
definition sat_coinit_cond ("_ sat.COINITIALITY in _,_ wrt _")
where
  "S sat.COINITIALITY in X,σ wrt a ≡
     (∀x∈X. σ^++ a x ⟶ ((σ^** x a) ∨ (∃b∈S. σ^** b x ∧ ¬ (σ^** b a))))"

— Definition 24: (X,σ) is a quasi-locally confluent ARS
definition is_qlconfl_ars  ("_,_ is Q.L.CONFLUENT ARS")
  where "X,σ is Q.L.CONFLUENT ARS ≡ (X,σ is ARS) ∧
         (∀a∈X. ∃ S ⊆ {x∈X. σ^++ a x}.
         (S sat.2-CONSISTENCY in X,σ) ∧ (S sat.COINITIALITY in X,σ wrt a))"
```

The statement of the main result is formalized as follows.

```
theorem thm_28:
— Theorem 28 (main result, generalized Newman's lemma)
fixes X σ
assumes "X,σ is S.I.ARS"
shows "(X,σ is CONFLUENT ARS) = ((X,σ is O.Q.N.ARS) ∧ (X,σ is Q.L.CONFLUENT ARS))"
```

5 Discussion

From a practical perspective, usual confluence criteria for ARS associated with discrete models of computation can be used in software tools that allow a user to make sure that an output of a nondeterministic program does not depend on its execution path. Program nondeterminism is frequently caused by concurrent mode of execution, so such tools can aid automated testing or formal verification of concurrent and distributed software. Similar reasons of nondeterminism exist for models of cyber-physical systems (CPS), however, for such models other sources of nondeterminism may also have significant role, e.g. abstractions and approximations used in dynamical models of physical components of a system can lead to non-uniqueness of solutions of state evolution equations [16]. Thus the reasons of development of confluence checkers for models of CPS can be generally analogous to the reasons of development of such checkers for discrete models of computation. Practical implementation of an automated confluence checker for models of CPS is out of scope of this paper, however, we consider this an important problem which requires further work, and the results obtained in this paper can be used to approach it.

One may question whether the notion of an ARS is adequate for modeling CPS. In our opinion, ARS can be used as an auxiliary model of dynamics for such systems, assuming that a reduction relation or its transitive closure represents a reachability relation on states of a primary dynamical model of a CPS (e.g. described as a hybrid dynamical system, etc.). This type of auxiliary model is most useful when a primary model satisfies a nondeterministic version of Markovian property [28, 12] (an adaptation of the respective notion known from the theory of stochastic processes to nondeterministic dynamical systems without any associated probability measures), which, informally speaking, generalizes (to non-discrete-time cases) the following condition which obviously holds for ARS: the set of possible reduction sequences which begin at a given element x does not depend on how x has been reached. When such a (generalized) property holds for a primary model, a correspondence between various properties of interest of an auxiliary model/ARS (like confluence) and of a primary model becomes particularly simple. Note that many types of (non-discrete-time) models relevant for CPS domain can be represented using nondeterministic dynamical systems which have Markovian property in the mentioned sense [12].

6 Conclusion

We have proposed a criterion of confluence for the class of abstract rewriting systems $(X, \to)$ such that $(X, \to^*)$ is a strictly inductive preordered set, where $\to^*$ is the reflexive transitive closure of the reduction relation $\to$. Ordinary Newman's lemma for terminating abstract rewriting systems can be considered as a corollary of this criterion. However, this criterion can also be applied to a wide class of non-terminating abstract rewriting systems.

We have described a formalization of this criterion in Isabelle proof assistant using HOL logic and have discussed its potential applications.

References

1 An Isabelle/HOL Formalization of Rewriting for Certified Tool Assertions. `http://cl-informatik.uibk.ac.at/isafor/`.

2 Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge university press, 1999.

3 Radhakisan Baheti and Helen Gill. Cyber-physical systems. *The impact of control technology*, 12(1):161–166, 2011.

4 Pete L. Clark. The instructor's guide to real induction. *Mathematics Magazine*, 92(2):136–150, 2019.

5 Patrick Dehornoy and Vincent van Oostrom. Z, proving confluence by monotonic single-step upperbound functions. *Logical Models of Reasoning and Computation (LMRC-08)*, page 85, 2008.

6 Jörg Endrullis, Jan Willem Klop, and Roy Overbeek. Decreasing diagrams with two labels are complete for confluence of countable systems. In *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

7 Jörg Endrullis, Jan Willem Klop, and Roy Overbeek. Decreasing diagrams for confluence and commutation. *Logical Methods in Computer Science*, 16, 2020.

8 Rafal Goebel, Ricardo G Sanfelice, and Andrew R Teel. Hybrid dynamical systems. *IEEE control systems magazine*, 29(2):28–93, 2009.

9 Jean Goubault-Larrecq. *Non-Hausdorff topology and domain theory: Selected topics in point-set topology*, volume 22. Cambridge University Press, 2013.

10 Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.

11 Isabelle proof assistant. `http://isabelle.in.tum.de`. [Online].

12 Ievgen Ivanov. On representations of abstract systems with partial inputs and outputs. In T.V. Gopal, M. Agrawal, A. Li, and S.B. Cooper, editors, *Theory and Applications of Models of Computation*, volume 8402 of *Lecture Notes in Computer Science*, pages 104–123. Springer, 2014.

13 Ievgen Ivanov. On induction for diamond-free directed complete partial orders. In *CEUR-WS.org*, volume 2732, pages 70–73, 2020.

14 Ievgen Ivanov. Formalization of Generalized Newman's Lemma (supplementary material for this paper). `https://doi.org/10.5281/zenodo.7855691`, 2023. [Online].

15 Jan Willem Klop. *Term rewriting systems*. Centrum voor Wiskunde en Informatica, 1990.

16 Edward A. Lee. Fundamental limits of cyber-physical systems modeling. *ACM Transactions on Cyber-Physical Systems*, 1(1), 2016.

17 Philippe Malbos. Lectures on algebraic rewriting. `http://hal.archives-ouvertes.fr/hal-02461874`, 2019.

18 George Markowsky. Chain-complete posets and directed sets with applications. *Algebra universalis*, 6(1):53–68, 1976.

19 F. Mattern. On the relativistic structure of logical time in distributed systems. *Parallel and Distributed Algorithms*, 1992.

20 Maxwell Herman Alexander Newman. On theories with a combinatorial definition of "equivalence". *Annals of mathematics*, pages 223–243, 1942.

21 Newman's lemma — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/Newman_lemma`. [Online].

22 Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer, 2002.

23 André Platzer and Yong Kiam Tan. Differential equation axiomatization: The impressive power of differential ghosts. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 819–828, 2018.

24 Jean-Claude Raoult. Proving open properties by induction. *Information processing letters*, 29(1):19–23, 1988.

25 Christian Sternagel and Rene Thiemann. Abstract rewriting. *Archive of Formal Proofs*, June 2010.

26 Vincent van Oostrom. Confluence by decreasing diagrams. *Theoretical computer science*, 126(2):259–280, 1994.

27 Freek Wiedijk. *The seventeen provers of the world: Foreword by Dana S. Scott*, volume 3600. Springer, 2006.

28 J. C. Willems. Paradigms and puzzles in the theory of dynamical systems. *IEEE Transactions on Automatic Control*, 36(3):259–294, 1991.

29 Harald Zankl. Decreasing diagrams. *Archive of Formal Proofs*, 2013.

E-Unification for Second-Order Abstract Syntax

Nikolai Kudasov
Innopolis University, Tatarstan Republic, Russia

Abstract

Higher-order unification (HOU) concerns unification of (extensions of) λ-calculus and can be seen as an instance of equational unification (E-unification) modulo $\beta\eta$-equivalence of λ-terms. We study equational unification of terms in languages with arbitrary variable binding constructions modulo arbitrary second-order equational theories. Abstract syntax with general variable binding and parametrised metavariables allows us to work with arbitrary binders without committing to λ-calculus or use inconvenient and error-prone term encodings, leading to a more flexible framework. In this paper, we introduce E-unification for second-order abstract syntax and describe a unification procedure for such problems, merging ideas from both full HOU and general E-unification. We prove that the procedure is sound and complete.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting

Keywords and phrases E-unification, higher-order unification, second-order abstract syntax

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.10

Related Version *Extended Version*: `https://arxiv.org/abs/2302.05815`

Acknowledgements I thank Nikolay Shilov for encouragement and fruitful discussions about the contents of the paper. I also thank Alexander Chichigin, Violetta Sim, Timur Fayzrahmanov, Alexey Stepanov, and Dale Miller for discussions of early versions of this work. I thank Alexander Chichigin, Timur Fayzrahmanov, and Ikechi Ndukwe for proofreading the paper. Finally, I thank the anonymous reviewers of FSCD-2023 for their valuable and detailed comments.

1 Introduction

Higher-order unification (HOU) is a process of solving symbolic equations with functions. Consider the following equation in untyped λ-calculus that we want to solve for M:

$$\mathrm{M}\ g\ (\lambda z.z\ a) \overset{?}{=} g\ a \tag{1}$$

A solution to this problem (called a unifier) is the substitution $\theta = [\mathrm{M} \mapsto \lambda x.\lambda y.y\ x]$. Indeed, applying θ to the equation we get β-equivalent terms on both sides:

$$\theta(\mathrm{M}\ g\ (\lambda z.z\ a)) = (\lambda x.\lambda y.y\ x)\ g\ (\lambda z.z\ a) \equiv_\beta (\lambda y.y\ g)\ (\lambda z.z\ a) \equiv_\beta (\lambda z.z\ a)\ g \equiv_\beta g\ a = \theta(g\ a)$$

Higher-order unification has many applications, including type checking [22] and automatic theorem proving in higher-order logics [24]. In general, HOU is undecidable [12] and searching for a unifier can be rather expensive without non-trivial optimizations. For some problems, a decidable fragment is sufficient to solve for. For instance, Miller's higher-order pattern unification [23] and its variations [13, 37] are often used for dependent type inference.

Traditionally, HOU algorithms consider only one binder – λ-abstraction. A common justification is an appeal to Higher-Order Abstract Syntax (HOAS) [27]:

> It is well-known that λ-abstraction is general enough to represent quantification in formulae, abstraction in functional programs, and many other variable-binding constructs [27]. (Nipkow and Prehofer [25, Section 1])

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 10; pp. 10:1–10:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

However, HOAS has received some criticism from both programming language implementors and formalisation researchers, who argue that HOAS and its variants [2, 36] have some practical issues [19, 5], such as being hard to work under binders, having issues with general recursion [18], and lacking a formal foundation [9].

Fiore and Szamozvancev [9] argue that existing developments for formalising, reasoning about, and implementing languages with variable bindings 'offer some relief, however at the expense of inconvenient and error-prone term encodings and lack of formal foundations'. Instead, they suggest to consider *second-order abstract syntax* [10], that is, abstract syntax with variable binding and parametrised metavariables. Indeed, Fiore and Szamozvancev [9] use second-order abstract syntax to generate metatheory in Agda for languages with variable bindings.

In this paper, we develop a mechanisation of equational reasoning for second-order abstract syntax. We take inspiration in existing developments for HOU and E-unification. Although we cannot directly reuse all HOU ideas that rely heavily on the syntax of λ-calculus, we are still able to adapt many of them, since second-order abstract syntax provides *parametrised metavariables* which are similar to *flex* terms in HOU.

1.1 Related Work

To the best of our knowledge, there does not exist a mechanisation of equational reasoning for second-order abstract syntax. Thus, we compare our approach with existing HOU algorithms that encompass equational reasoning. Snyder's higher-order E-unification [28] extends HOU with first-order equational theories. Nipkow and Prehofer [25] study higher-order rewriting and (higher-order) equational reasoning. As mentioned, these rely on λ-abstraction and a HOAS-like encoding to work with other binding constructions. In contrast, we work with arbitrary binding constructions modulo a second-order equational theory.

Dowek, Hardin, and Kirchner [8] present higher-order unification as first-order E-unification in $\lambda\sigma$-calculus (a variant of λ-calculus with explicit substitutions) modulo $\beta\eta$-reduction. Their idea is to use explicit substitutions and de Bruijn indices so that metavariable substitution cannot result in name capture and reduces to *grafting* (first-order substitution). In this way, algorithms for first-order E-unification (such as *narrowing*) can be applied. Kirchner and Ringeissen [17] develop that approach for higher-order equational unification with first-order axioms. In our work, parametrised metavariables act in a similar way to metavariables with explicit substitutions in $\lambda\sigma$-calculus. While it should be possible to encode second-order equations as first-order equations in σ-calculus (with explicit substitution, but without λ-abstraction and application), it appears that this approach requires us to also encode rules of our unification procedure.

As some equational theories can be formulated as term rewriting systems, a line of research combining rewrite systems and type systems exists, stemming from the work of Tannen [33], which extends simply-typed λ-calculus with higher-order rewrite rules. Similar extensions exist for the Calculus of Constructions [1, 35, 29, 30] and $\lambda\Pi$-calculus [7]. Cockx, Tabareau, and Winterhalter [6] introduce Rewriting Type Theory (RTT) which is an extension of Martin-Löf Type Theory with (first-order) rewrite rules. Chrząszcz and Walukiewicz-Chrząszcz [3] discuss how to extend Coq with rewrite rules. Cockx [4] reports on a practical extension of Agda with higher-order non-linear rewrite rules, based on the same ideas as RTT [6]. Rewriting is especially useful in proof assistants that compare types (and terms) through *normalisation by evaluation* (NbE) rather than higher-order unification. Contrary to type theories extended with rewrite rules, our approach relies on simply-typed syntax, but allows for an arbitrary second-order equational theory, enabling unification even in the absence of a confluent rewriting system.

Kudasov [20] implements higher-order (pre)unification and dependent type inference in Haskell for an intrinsically scoped syntax using so-called *free scoped monads* to generate the syntax of the object language from a data type describing node types. Such a definition is essentially a simplified presentation of second-order abstract syntax. Kudasov's pre-unification procedure contains several heuristics, however no soundness or completeness results are given in the preprint.

1.2 Contribution

The primary contribution of this paper is the introduction of E-unification for second-order abstract syntax and a sound and complete unification procedure. The rest of the paper is structured as follows:

- In Section 2, we briefly revisit second-order abstract syntax, equational logic, and term rewriting à la Fiore and Hur [10].
- In Section 3, we generalise traditional E-unification concepts of an E-unification problem and an E-unifier for a set of second-order equations E.
- In Section 4, we define the unification procedure that enumerates solutions for any given E-unification problem and prove it sound.
- In Section 5, we prove completeness of our unification procedure, taking inspiration from existing research on E-unification and HOU.
- Finally, we discuss some potential pragmatic changes for a practical implementation as well as limitations of our approach in Section 6.

2 Second-Order Abstract Syntax

In this section, we recall second-order abstract syntax, second-order equational logic, and second-order term rewriting of Fiore and Hur [10].

2.1 Second-Order Terms

We start by recalling a notion of second-order signature, which essentially contains information about the syntactic constructions (potentially with variable bindings) of the object language.

A ***second-order signature*** [10, Section 2] $\Sigma = (T, O, |-|)$ is specified by a set of types T, a set of operators[1] O, and an arity[2] function $|-| : O \to (T^* \times T)^* \times T$. For an operator $\mathsf{F} \in O$, we write $\mathsf{F} : (\overline{\sigma_1}.\tau_1, \ldots, \overline{\sigma_n}.\tau_n) \to \tau$ when $|\mathsf{F}| = ((\overline{\sigma_1}, \tau_1), \ldots, (\overline{\sigma_n}, \tau_n), \tau)$. Intuitively, this means that an operator F takes n arguments each of which binds $n_i = |\overline{\sigma_i}|$ variables of types $\sigma_{i,1}, \ldots, \sigma_{i,n_i}$ in a term of type τ_i.

For the rest of the paper, we assume an ambient signature Σ, unless otherwise stated.

A ***typing context*** [10, Section 2] $\Theta \mid \Gamma$ consists of metavariable typings Θ and variable typings Γ. Metavariable typings are parametrised types: a metavariable of type $[\sigma_1, \ldots, \sigma_n]\tau$, when parametrised by terms of type $\sigma_1, \ldots, \sigma_n$, will yield a term of type τ. We will write a centered dot $(\cdot)$ for the empty (meta)variable context.

For example, this context has a metavariable M with two parameters and variables x, y: $\Theta \mid \Gamma = (\mathrm{M} : [\sigma, \sigma \Rightarrow \tau]\tau \mid x : \sigma \Rightarrow \tau, y : \sigma)$.

[1] In literature on E-unification, authors use the term *functional symbol* instead.
[2] We follow the terminology of Fiore and Hur.

$$\frac{x : \tau \in \Gamma}{\Theta \mid \Gamma \vdash x : \tau}\ \text{variables} \qquad \frac{\begin{array}{c}\mathrm{M} : [\sigma_1, \ldots, \sigma_n]\tau \in \Theta \\ \text{for all } i = 1, \ldots, n \quad \Theta \mid \Gamma \vdash t_i : \sigma_i\end{array}}{\Theta \mid \Gamma \vdash \mathrm{M}[t_1, \ldots, t_n] : \tau}\ \text{metavariables}$$

$$\frac{\begin{array}{c}\mathsf{F} : (\overline{\sigma_1}.\tau_1, \ldots, \overline{\sigma_n}.\tau_n) \to \tau \\ \text{for all } i = 1, \ldots, n \quad \Theta \mid \Gamma, \overline{x_i} : \overline{\sigma_i} \vdash t_i : \tau_i\end{array}}{\Theta \mid \Gamma \vdash \mathsf{F}(\overline{x_1}.t_1, \ldots, \overline{x_n}.t_n) : \tau}\ \text{operators}$$

Figure 1 Second-order terms in context.

▶ **Definition 1** ([10, Section 2]). *A judgement for typed* **terms** *in context* $\Theta \mid \Gamma \vdash - : \tau$ *is defined by the rules in Figure 1. Variable substitution on terms is defined in a usual way, see [10, Section 2] for details.*

Let $\Theta = (\mathrm{M}_i : [\overline{\sigma_i}]\tau_i)^{i \in \{1,\ldots,n\}}$*, and consider a term* $\Theta \mid \Gamma \vdash t : \tau$*, and for all* $i \in \{1, \ldots, n\}$ *a term in extended*[3] *context* $\Xi \mid \Gamma, \Delta, \overline{z_i} : \overline{\sigma_i} \vdash t_i : \tau_i$*. Then,* **metavariable substitution** $t[\mathrm{M}_i[\overline{z_i}] \mapsto t_i]^{i \in \{1,\ldots,n\}}$ *is defined recursively on the structure of* t*:*

$$\begin{aligned}
x[\mathrm{M}_i[\overline{z_i}] \mapsto t_i]^{i \in \{1,\ldots,n\}} &= x \\
\mathrm{M}_k[\overline{s}][\mathrm{M}_i[\overline{z_i}] \mapsto t_i]^{i \in \{1,\ldots,n\}} &= t_k[\overline{z_k} \mapsto \overline{s[\mathrm{M}_i[\overline{z_i}] \mapsto t_i]^{i \in \{1,\ldots,n\}}}] \\
&\quad \textit{when } k \in \{1, \ldots, n\} \textit{ and } |\overline{s}| = |\overline{z_k}| \\
\mathrm{N}[\overline{s}][\mathrm{M}_i[\overline{z_i}] \mapsto t_i]^{i \in \{1,\ldots,n\}} &= \mathrm{N}[\overline{s[\mathrm{M}_i[\overline{z_i}] \mapsto t_i]^{i \in \{1,\ldots,n\}}}] \qquad \textit{when } \mathrm{N} \notin \{\mathrm{M}_1, \ldots, \mathrm{M}_n\} \\
\mathsf{F}(\overline{x}.s)[\mathrm{M}_i[\overline{z_i}] \mapsto t_i]^{i \in \{1,\ldots,n\}} &= \mathsf{F}(\overline{\overline{x}.s[\mathrm{M}_i[\overline{z_i}] \mapsto t_i]^{i \in \{1,\ldots,n\}}})
\end{aligned}$$

We write $\theta : \Theta \mid \Gamma \to \Xi \mid \Gamma, \Delta$ *for a substitution* $\theta = [\mathrm{M}_i[\overline{z_i}] \mapsto t_i]^{i \in \{1,\ldots,n\}}$.

When both Γ *and* Δ *are empty, we write* $\theta : \Theta \to \Xi$ *as a shorthand for* $\theta : \Theta \mid \cdot \to \Xi \mid \cdot$.

For single metavariable substitutions in a larger context we will omit the metavariables that map to themselves. That is, we write $[\mathrm{M}_k[\overline{z}] \mapsto t_k] : \Theta \mid \Gamma \to \Xi \mid \Gamma, \Delta$ *to mean that* $t_i = \mathrm{M}_i[\overline{z_i}]$ *for all* $i \neq k$*.*

2.2 Second-Order Equational Logic

We now define second-order equational presentations and rules of second-order logic, following Fiore and Hur [10, Section 5]. This provides us with tools for reasoning modulo second-order equational theories, such as $\beta\eta$-equivalence of λ-calculus.

An ***equational presentation*** [10, Section 5] is a set of axioms each of which is a pair of terms in context.

▶ **Example 2.** Terms of simply-typed λ-calculus are generated with a family of operators (for all σ, τ) – $\mathsf{abs}^{\sigma,\tau} : \sigma.\tau \to (\sigma \Rightarrow \tau)$ and $\mathsf{app}^{\sigma,\tau} : (\sigma \Rightarrow \tau, \sigma) \to \tau$. And equational presentation for simply-typed λ-calculus is given by a family of axioms:

$$\mathrm{M} : [\sigma]\tau, \mathrm{N} : []\sigma \mid \cdot \vdash \mathsf{app}(\mathsf{abs}(x.\mathrm{M}[x]), \mathrm{N}[]) \equiv \mathrm{M}[\mathrm{N}[]] : \tau \qquad (\beta)$$

$$\mathrm{M} : []\sigma \Rightarrow \tau \mid \cdot \vdash \mathsf{abs}(x.\mathsf{app}(\mathrm{M}[], x)) \equiv \mathrm{M}[] : \sigma \Rightarrow \tau \qquad (\eta)$$

[3] Here we slightly generalise the definition of Fiore and Hur by allowing arbitrary extension of context to Γ, Δ in the resulting term. This is useful in particular when Γ is empty. See Definition 26.

$$\frac{(\Theta \mid \Gamma \vdash s \equiv t : \tau) \in E}{\Theta \mid \Gamma \vdash s \equiv_E t : \tau}\ \text{axiom} \qquad \frac{\Theta \mid \Gamma \vdash t : \tau}{\Theta \mid \Gamma \vdash t \equiv_E t : \tau}\ \text{refl}$$

$$\frac{\Theta \mid \Gamma \vdash s \equiv_E t : \tau}{\Theta \mid \Gamma \vdash t \equiv_E s : \tau}\ \text{sym} \qquad \frac{\Theta \mid \Gamma \vdash s \equiv_E t : \tau \qquad \Theta \mid \Gamma \vdash t \equiv_E u : \tau}{\Theta \mid \Gamma \vdash s \equiv_E u : \tau}\ \text{trans}$$

$$\frac{\begin{array}{c}\mathrm{M}_1 : [\overline{\sigma_1}]\tau_1, \ldots, \mathrm{M}_n : [\overline{\sigma_n}]\tau_n \mid \Gamma \vdash s \equiv_E t : \tau \\ \text{for all } i \in \{1, \ldots, n\} \quad \Theta \mid \Delta, \overline{x_i} : \overline{\sigma_i} \vdash s_i \equiv_E t_i : \tau_i\end{array}}{\Theta \mid \Gamma, \Delta \vdash s[\mathrm{M}_1[\overline{x_1}] \mapsto s_1, \ldots, \mathrm{M}_n[\overline{x_n}] \mapsto s_n] \equiv_E t[\mathrm{M}_1[\overline{x_1}] \mapsto t_1, \ldots, \mathrm{M}_n[\overline{x_n}] \mapsto t_n] : \tau}\ \text{subst}$$

Figure 2 Rules of the second-order equational logic.

Note that the types here do not depend on the context, so it makes sense to only allow equating terms of the same type. This is in contrast to dependently typed systems, where terms can have different (but equivalent) types.

An equational presentation E generates a ***second-order equational logic*** [10, Fig. 2]. Rules for second-order equational logic are given in Figure 2.

In their paper, Fiore and Hur note that metavariables with zero parameters are equivalent to regular variables. Indeed, we can ***parametrise*** every term $\Theta \mid \Gamma \vdash t : \tau$ to yield a term $\Theta, \widehat{\Gamma} \mid \cdot \vdash \widehat{t} : \tau$ where for $\Gamma = (x_1 : \sigma_1, \ldots, x_n : \sigma_n)$ we have

$$\widehat{\Gamma} = (\mathrm{X}_1 : []\sigma_1, \ldots, \mathrm{X}_n : []\sigma_n) \qquad \widehat{t} = t[x_1 \mapsto \mathrm{X}_1[], \ldots, x_n \mapsto \mathrm{X}_n[]]$$

Applying parametrisation to an equational presentation E yields a set of parametrised equations $\widehat{E}$. Note that the following are equivalent:

$$\Theta \mid \Gamma \vdash s \equiv_E t : \tau \qquad \text{iff} \qquad \Theta, \widehat{\Gamma} \mid \cdot \vdash \widehat{s} \equiv_{\widehat{E}} \widehat{t} : \tau$$

Thus, from now on, we assume that axioms have empty variable context.

2.3 Second-Order Term Rewriting

Finally, for the proof of completeness in Section 5, it will be helpful to rely on chains of term rewrites rather than derivation trees of equality modulo E. Fiore and Hur introduce the *second-order term rewriting* relation [10, Section 8].

An equational presentation E generates a ***second-order term rewriting relation*** $\longrightarrow_E$ [10, Fig. 4]. We write $s \xrightarrow{*}_E t$ if there is a sequence of terms $u_1, \ldots, u_n$ such that $s = u_1 \longrightarrow_E \ldots \longrightarrow_E u_n = t$. We write $s \longleftrightarrow_E t$ if either $s \longrightarrow_E t$ or $t \longrightarrow_E s$. We write $s \overset{*}{\longleftrightarrow}_E t$ if there is a sequence of terms $u_1, \ldots, u_n$ such that $s = u_1 \longleftrightarrow_E \ldots \longleftrightarrow_E u_n = t$.

Since we only care about substitutions of metavariables in axioms (variable context is empty), a simplified version of the rules is given in Figure 3.

An important result of Fiore and Hur is that of soundness and completeness of second-order term rewriting [10, Section 8]: $\Theta \mid \Gamma \vdash s \equiv_E t : \tau$ iff $\Theta \mid \Gamma \vdash s \overset{*}{\longleftrightarrow}_E t : \tau$.

3 E-unification with Second-Order Equations

In this section, we formulate the equational unification problem for second-order abstract syntax, describe what constitutes a solution to such a problem and whether it is complete. We also recognise a subclass of problems in solved form, i.e. problems that have an immediate solution. For the most part, this is a straightforward generalisation of standard concepts of E-unification [11].

$$\frac{\begin{array}{c}(\mathrm{M}_1 : [\overline{\sigma_1}]\tau_1, \ldots, \mathrm{M}_k : [\overline{\sigma_k}]\tau_k \mid \cdot \vdash l \equiv r : \tau) \in E \\ \Theta \mid \Gamma, \overline{x_i} : \overline{\sigma_i} \vdash t_i : \tau_i : \quad \text{for } i \in \{1, \ldots, k\}\end{array}}{\Theta \mid \Gamma \vdash l[\mathrm{M}_i[\overline{z_i}] \mapsto t_i]^{i \in \{1,\ldots,k\}} \longrightarrow r[\mathrm{M}_i[\overline{z_i}] \mapsto t_i]^{i \in \{1,\ldots,k\}} : \tau}$$

$$\frac{\mathrm{M} : [\overline{\sigma}]\tau \qquad \Theta \mid \Gamma \vdash s_i \longrightarrow t_i : \sigma_i}{\Theta \mid \Gamma \vdash \mathrm{M}[\ldots, s_i, \ldots] \longrightarrow \mathrm{M}[\ldots, t_i, \ldots] : \tau}$$

$$\frac{\mathsf{F} : (\overline{\sigma_1}.\tau_1, \ldots, \overline{\sigma_n}.\tau_n) \to \tau \qquad \Theta \mid \Gamma, \overline{x_i} : \overline{\sigma_i} \vdash s_i \longrightarrow t_i : \tau_i}{\Theta \mid \Gamma \vdash \mathsf{F}(\ldots, \overline{x_i}.s_i, \ldots) \longrightarrow \mathsf{F}(\ldots, \overline{x_i}.t_i, \ldots) : \tau}$$

Figure 3 Rules of the second-order term rewriting (simplified). In the second and third rules, the subterms under $(\ldots)$ are kept unchanged, so only one subterm is rewritten per rule.

▶ **Definition 3.** *A* **second-order constraint** $\Theta \mid \Gamma_\exists, \Gamma_\forall \vdash s \overset{?}{=} t : \tau$ *is a pair of terms in a context, where variable context is split into two components:* $\Gamma = (\Gamma_\exists, \Gamma_\forall)$.

The idea is that $\Gamma_\exists$ contains variables that we need to solve for (free variables), while $\Gamma_\forall$ contains variables that we cannot substitute (bound variables). Metavariables are always treated existentially, so we do not split metavariable context. Similarly to equational representations, we can parametrise (a set of) constraints, yielding $\Theta, \widehat{\Gamma_\exists} \mid \Gamma_\forall \vdash s \overset{?}{=} t : \tau$. Thus, from now on, we will assume $\Gamma = \Gamma_\forall$ (i.e. $\Gamma_\exists$ is empty) for all constraints.

▶ **Example 4.** Assume $\alpha = \sigma \Rightarrow \tau, \beta = (\sigma \Rightarrow \tau) \Rightarrow \tau$. The following are equivalent:

1. For all $g : \alpha, a : \sigma$, find $m : \alpha \Rightarrow \beta \Rightarrow \tau$ such that $m\ g\ (\lambda z.z\ a) = g\ a$.
2. $\cdot \mid m : \alpha \Rightarrow \beta \Rightarrow \tau, g : \alpha, a : \sigma \vdash \mathsf{app}(\mathsf{app}(m, g), \mathsf{abs}(z.\mathsf{app}(z, a))) \overset{?}{=} \mathsf{app}(g, a) : \tau$
3. $\mathrm{M} : []\alpha \Rightarrow \beta \Rightarrow \tau \mid g : \alpha, a : \sigma \vdash \mathsf{app}(\mathsf{app}(\mathrm{M}[], g), \mathsf{abs}(z.\mathsf{app}(z, a))) \overset{?}{=} \mathsf{app}(g, a) : \tau$
4. $\mathrm{M} : [\alpha, \beta]\tau \mid g : \alpha, a : \sigma \vdash \mathrm{M}[g, \mathsf{abs}(z.\mathsf{app}(z, a))] \overset{?}{=} \mathsf{app}(g, a) : \tau$

Here, Item 2 is a direct encoding of Item 1 as a second-order constraint. Item 3 is a parametrised version of Item 2. Item 4 is equivalent to Item 3 modulo β-equality, witnessed by metasubstitutions $[\mathrm{M}[] \mapsto \mathsf{abs}(x.\mathsf{abs}(y.\mathsf{app}(x, y)))]$ and $[\mathrm{M}[x, y] \mapsto \mathsf{app}(\mathsf{app}(\mathrm{M}[], x), y)]$.

▶ **Definition 5.** *Given an equational presentation E, an* E**-unification problem** $\langle\Theta, S\rangle$ *is a finite set S of second-order constraints in a shared metavariable context Θ. We present an E-unification problem as a formula of the following form:*

$$\exists(\mathrm{M}_1 : [\overline{\sigma_1}]\tau_1, \ldots, \mathrm{M}_n : [\overline{\sigma_n}]\tau_n).(\forall(\overline{z_1} : \overline{\rho_1}).s_1 \overset{?}{=} t_1 : \tau_1) \wedge \ldots \wedge (\forall(\overline{z_k} : \overline{\rho_k}).s_k \overset{?}{=} t_k : \tau_k)$$

▶ **Definition 6.** *A metavariable substitution $\xi : \Theta \to \Xi$ is called an* E**-unifier** *for an E-unification problem $\langle\Theta, S\rangle$ if for all constraints $(\Theta \mid \Gamma_\forall \vdash s \overset{?}{=} t : \tau) \in S$ we have*

$$\Xi \mid \Gamma_\forall \vdash \xi s \equiv_E \xi t : \tau$$

We write $U_E(S)$ for the set of all E-unifiers for $\langle\Theta, S\rangle$.

▶ **Example 7.** Consider unification problem $\langle\Theta, S\rangle$ for the simply-typed λ-calculus:

$$\Theta = \mathrm{M} : [\sigma \Rightarrow \tau, (\sigma \Rightarrow \tau) \Rightarrow \tau]\tau$$

$$S = \{\Theta \mid g : \sigma \Rightarrow \tau, y : \sigma \vdash \mathrm{M}[g, \mathsf{abs}(x.\mathsf{app}(x, y))] \overset{?}{=} \mathsf{app}(g, y) : \tau\}$$

Substitution $[\mathrm{M}[z_1, z_2] \mapsto \mathsf{app}(z_2, z_1)] : \Theta \to \cdot$ is an E-unifier for $\langle\Theta, S\rangle$.

3.1 Unification Problems in Solved Form

Here, we recognise a class of trivial unification problems. The idea is that a constraint that looks like a metavariable substitution can be uniquely unified. A unification problem can be unified as long as substitutions for constraints are sufficiently disjoint. More precisely:

▶ **Definition 8.** *An E-unification problem $\langle\Theta, S\rangle$ is in* **solved form** *when S consists only of constraints of the form $\Theta, \mathrm{M} : [\overline{\sigma}]\tau \mid \Gamma_\forall \vdash \mathrm{M}[\overline{z}] \overset{?}{=} t : \tau$ such that*

1. $\overline{z} : \overline{\sigma} \subseteq \Gamma_\forall$ *(parameters of* M *are* distinct *variables from* $\Gamma_\forall$*)*
2. $\Theta \mid \overline{z} : \overline{\sigma} \vdash t : \tau$ *(*M *and variables not occurring in* $\overline{z}$ *do not occur in* t*)*
3. *all constraints have distinct metavariables on the left hand side*

▶ **Example 9.** Let $\Theta = (\mathrm{M} : [\sigma, \sigma]\sigma)$. Then

1. $\{\Theta \mid x : \sigma, y : \sigma \vdash \mathrm{M}[y, x] \overset{?}{=} \mathsf{app}(\mathsf{abs}(z.x), y) : \sigma\}$ is in solved form;
2. $\{\Theta \mid x : \sigma, y : \sigma \vdash \mathrm{M}[x, x] \overset{?}{=} \mathsf{app}(\mathsf{abs}(z.x), y) : \sigma\}$ is not in solved form, since parameters of M are not *distinct* variables and also since variable y occurs on the right hand side, but does not occur in parameters of M;
3. $\{\Theta \mid f : \sigma \Rightarrow \sigma, y : \sigma \vdash \mathrm{M}[y, \mathsf{app}(f, y)] \overset{?}{=} \mathsf{app}(f, y) : \sigma\}$ is not in solved form, since second parameter of M is not a variable.

▶ **Proposition 10.** *An E-unification problem $\langle\Theta, S\rangle$ in solved form has an E-unifier.*

Proof. Assume $\Theta = \{\Theta \mid \Gamma_i \vdash \mathrm{M}_i[\overline{z_i}] \overset{?}{=} t_i : \tau_i\}^{i \in \{1,\ldots,n\}}$. Let $\xi_S = [\mathrm{M}_i[\overline{z_i}] \mapsto t_i]^{i \in \{1,\ldots,n\}}$. Note that ξ_S is a well formed metasubstitution since, by assumption, each $\overline{z_i}$ is a sequence of distinct variables, t_i does not reference other variables or M_i, and each metavariable M_i is mapped only once in ξ_S. Applying ξ_S to each constraint we get trivial constraints, which are satisfied by reflexivity: $\Theta \mid \Gamma_i \vdash t_i \equiv_E t_i : \tau_i$. Thus, ξ_S is an E-unifier for $\langle\Theta, S\rangle$. ◀

Later, we will refer to the E-unifier constructed in the proof of Proposition 10 as ξ_S.

3.2 Comparing E-unifiers

In general, a unification problem may have multiple unifiers. Here, we generalise the usual notion of comparing E-unifiers [11] to the second-order abstract syntax using the *subsumption* order, leading to a straightforward generalisation of the ideas of *the most general unifier* and *a complete set of unifiers*. We do not consider generalising *essential unifiers* [14, 32] or *homeomorphic embedding* [31], although these might constitute a prospective future work.

▶ **Definition 11.** *Two metavariable substitutions $\theta, \xi : \Theta \to \Xi$ are said to be* **equal modulo** E *(notated $\theta \equiv_E \xi$) if for all metavariables $\mathrm{M} : [\overline{\sigma}]\tau \in \Theta$, any context Γ, and any terms $\Theta \mid \Gamma \vdash t_i : \sigma_i$ (for all $i \in \{1, \ldots, n\}$) we have*

$$\Xi \mid \Gamma \vdash \theta\mathrm{M}[t_1, \ldots, t_n] \equiv_E \xi\mathrm{M}[t_1, \ldots, t_n] : \tau$$

We say that θ is **more general modulo** E **than** ξ *(notated $\theta \preccurlyeq_E \xi$) when there exists a substitution $\eta : \Xi \to \Xi$ such that $\eta \circ \theta \equiv_E \xi$.*

Empty substitution is more general than any substitution. A more interesting example may be found in λ-calculus. Let

$$\theta_1 = [\mathrm{M}[x, y] \mapsto \mathsf{app}(\mathrm{N}[x], y)]$$
$$\theta_2 = [\mathrm{M}[x, y] \mapsto \mathsf{app}(\mathsf{abs}(z.x), y), \mathrm{N}[x] \mapsto \mathsf{abs}(z.x)]$$
$$\theta_3 = [\mathrm{M}[x, y] \mapsto x, \mathrm{N}[x] \mapsto \mathsf{abs}(z.x)]$$

Then, $\theta_1 \preccurlyeq_E \theta_2$, $\theta_2 \equiv_E \theta_3$, and $\theta_1 \preccurlyeq_E \theta_3$ (witnessed by $[\mathrm{N}[x] \mapsto \mathsf{abs}(z.x)] \circ \theta_1 \equiv_E \theta_3$).

▶ **Proposition 12.** *If $\theta \equiv_E \xi$ then for any E-unification problem $\langle \Theta, S \rangle$ we have $\theta \in U_E(S)$ iff $\xi \in U_E(S)$.*

Proof. For each constraint $\Theta \mid \Gamma_\forall \vdash s \overset{?}{=} t : \tau$, by induction on the structure of s and t it is straightforward to show that $\Xi \mid \Gamma \vdash \theta s \equiv_E \theta t : \tau$ iff $\Xi \mid \Gamma \vdash \xi s \equiv_E \xi t : \tau$. ◀

▶ **Corollary 13.** *If $\theta \preccurlyeq_E \xi$ and $\theta \in U_E(S)$ then $\xi \in U_E(S)$.*

Not all substitutions can be compared. Consider untyped lambda calculus with $\star$ being the type of any term. Let $\Theta = (\mathrm{M} : [\star, \star]\star)$ and

$$\theta = [\mathrm{M}[z_1, z_2] \mapsto \mathsf{app}(z_2, z_1)]$$
$$\xi = [\mathrm{M}[z_1, z_2] \mapsto \mathsf{app}(z_1, \mathsf{app}(z_2, \mathsf{abs}(z.z)))]$$

None of these substitutions is more general modulo equational theory E of untyped λ-calculus than the other. At the same time, both are E-unifiers for the problem

$$\exists(\mathrm{M} : [\star, \star]\star).\ \forall(g : \star, y : \star).\ \mathrm{M}[g, \mathsf{abs}(x.\mathsf{app}(x, y))] \overset{?}{=} \mathsf{app}(g, y) : \star$$

3.3 Complete Sets of E-unifiers

While there is sometimes more than one solution to an E-unification problem, we may often hope to collect several sufficiently general unifiers into a single set:

▶ **Definition 14.** *Given an E-unification problem $\langle \Theta, S \rangle$, a (minimal)* **complete set of E-unifiers** *for $\langle \Theta, S \rangle$ (notated $\mathsf{CSU}_E(S)$) is a subset of $U_E(S)$ such that*
1. *(completeness) for any $\eta \in U_E(S)$ there exists $\theta \in \mathsf{CSU}_E(S)$ such that $\theta \preccurlyeq_E \eta$;*
2. *(minimality) for any $\theta, \xi \in \mathsf{CSU}_E(S)$ if $\theta \preccurlyeq_E \xi$ then $\theta = \xi$.*

We reserve the notation $\mathsf{CSU}_E(S)$ to refer to minimal complete sets of E-unifiers (i.e. satisfying both conditions).

▶ **Example 15.** The E-unification problem $\langle \Theta, S \rangle$ in untyped λ-calculus has an infinite $\mathsf{CSU}_E(S)$:

$$\begin{aligned}
\langle \Theta, S \rangle &= \exists(\mathrm{M} : [\star, \star]\star).\ \forall(g : \star, y : \star).\ \mathrm{M}[g, \mathsf{abs}(x.\mathsf{app}(x, y))] \overset{?}{=} \mathsf{app}(g, y) : \star \\
\mathsf{CSU}_E(S) &= \{[\mathrm{M}[z_1, z_2] \mapsto \mathsf{app}(z_2, z_1)], \\
&\qquad [\mathrm{M}[z_1, z_2] \mapsto \mathsf{app}(z_1, \mathsf{app}(z_2, \mathsf{abs}(x.x)))], \\
&\qquad [\mathrm{M}[z_1, z_2] \mapsto \mathsf{app}(\mathsf{app}(z_2, \mathsf{abs}(x.\mathsf{abs}(f.\mathsf{app}(f, x)))), z_1)], \ldots\}
\end{aligned}$$

▶ **Proposition 16.** *For any two minimal complete sets of E-unifiers $\mathsf{CSU}^1_E(S)$ and $\mathsf{CSU}^2_E(S)$, there exists a bijection $f : \mathsf{CSU}^1_E(S) \longleftrightarrow \mathsf{CSU}^2_E(S)$ such that*

$$\forall \theta \in \mathsf{CSU}^1_E(S).\quad \theta \equiv_E f(\theta)$$

Thus, $\mathsf{CSU}_E(S)$ is unique up to a bijection modulo E, so from now on we will refer to *the* complete set of E-unifiers.

▶ **Definition 17.** *When the complete set of E-unifiers $\mathsf{CSU}_E(S)$ is a singleton set, then we refer to its element as* **the most general E-unifier** *of S (notated $\mathsf{mgu}_E(S)$).*

▶ **Example 18.** Consider this E-unification problem $\langle \Theta, S \rangle$ in simply-typed λ-calculus:

$$\exists(\mathrm{M} : [\sigma \Rightarrow \tau, (\sigma \Rightarrow \tau) \Rightarrow \tau]\tau).\ \forall(g : \sigma \Rightarrow \tau, y : \sigma).\ \mathrm{M}[g, \mathsf{abs}(x.\mathsf{app}(x, y))] \overset{?}{=} \mathsf{app}(g, y) : \tau$$

For this problem the most general E-unifier exists: $\mathsf{mgu}_E(S) = [\mathrm{M}[z_1, z_2] \mapsto \mathsf{app}(z_2, z_1)]$. This example differs from Example 15 as here we work in simply-typed lambda calculus.

▶ **Proposition 19.** *If $\langle \Theta, S \rangle$ is an E-unification problem in solved form, then* $\mathsf{mgu}_E(S) \equiv_E \xi_S$.

Proof. It is enough to check that for any E-unifier $\theta \in U_E(S)$ we have $\xi_S \preccurlyeq_E \theta$. Observe that $\theta \equiv_E \theta \circ \xi_S$ since for any constraint $(\Theta \mid \Gamma_\forall \vdash M[\overline{z}] \overset{?}{=} t : \tau) \in S$ such that $\mathrm{M} : [\overline{\sigma}]\tau \in \Theta$, any context Γ, and any terms $\Theta \mid \Gamma \vdash t_i : \sigma_i$ (for all $i \in \{1, \ldots, n\}$) we have

$$\Xi \mid \Gamma \vdash \theta\mathrm{M}[\overline{t}] \equiv_E \theta t[\overline{z} \mapsto \overline{t}] \equiv_E \theta(\xi_S\mathrm{M}[\overline{z}])[\overline{z} \mapsto \overline{t}] \equiv_E \theta(\xi_S\mathrm{M}[\overline{t}]) : \tau$$ ◀

4 Unification Procedure

In this section, we introduce a unification procedure to solve arbitrary E-unification problems over second-order abstract syntax. We show that the procedure is sound at the end of this section, and we devote Section 5 for the completeness result.

Our unification procedure has features inspired by classical E-unification and HOU algorithms. For the equational part, we took inspiration from the complete sets of transformations for general (first-order) E-unification of Gallier and Snyder [11]. For unification of metavariables, we took inspiration from Huet's higher-order pre-unification [15] and Jensen and Pietrzykowski's procedure [16]. Some key insights from the recent work by Vukmirovic, Bentkamp, and Nummelin [34] give us the opportunity to improve the algorithm further, however, we are not attempting to achieve an *efficient* E-unification for second-order abstract syntax in this paper.

Note that we cannot directly reuse HOU ideas in our procedure, since we do not have full λ-calculus at our disposal. Instead we only have parametrised metavariables $\mathrm{M}[t_1, \ldots, t_n]$ which are analogous to applications of variables in HOU $(m\ t_1\ \ldots\ t_n)$. Still, we can adapt some ideas if they do not rely on normalisation or specific syntax of λ-calculus. For other ideas, we introduce simpler, yet more general versions. This allows us to preserve completeness, perhaps, sacrificing some efficiency, making the search space larger. While we believe it is possible to optimise our procedure to have virtually the same running time for unification problems in λ-calculus as HOU algorithms mentioned above, we leave such optimisations for future work.

To produce the unification procedure we follow and generalise some of the common steps that can be found in literature on HOU and first-order E-unification:

1. Classify substitutions that will constitute partial solutions for certain classes of constraints. The idea is that an overall solution will emerge as a composition of partial solutions.
2. Define transition rules that make small steps towards a solution.
3. Determine when to stop (succeed or fail).
4. If possible, organize rules in a proper order, yielding a unification procedure.

4.1 Bindings

Now we define different elementary substitutions that will serve as partial solutions for some constraints in our unification procedure. Here, we generalise a list of bindings collected by Vukmirovic, Bentkamp, and Nummelin [34]. From that list, Huet-style projection (also known as *partial binding* in HOU literature) is not used. Instead, imitation for axioms and JP-style projection bindings cover all substitutions that can be generated by Huet-style projection bindings[4]. We also use a simplified version of iteration binding here, again, since it generates all necessary bindings when considered together with generalised imitation binding.

▶ **Definition 20.** *We define the following types of bindings ζ:*

JP-style projection for M. *If* $\mathrm{M} : [\sigma_1, \ldots, \sigma_k]\tau$ *and* $\sigma_i = \tau$ *then*
$\zeta = [\mathrm{M}[\overline{z}] \mapsto z_i]$ *is a JP-style projection binding*

Imitation for M. *If* $\mathrm{M} : [\sigma_1, \ldots, \sigma_k]\tau$, $\mathsf{F} : (\overline{\alpha_1}.\beta_1, \ldots, \overline{\alpha_n}.\beta_n) \to \tau$ *and* $\mathrm{M}_i : [\sigma_1, \ldots, \sigma_k, \overline{\alpha_i}]\beta_i$ *for all* i*,*
$\zeta = [\mathrm{M}[\overline{z}] \mapsto \mathsf{F}(\overline{x_1}.\mathrm{M}_1[\overline{z}, \overline{x_1}], \ldots, \overline{x_n}.\mathrm{M}_n[\overline{z}, \overline{x_n}])]$ *is an imitation binding*

Elimination for M. *If* $\mathrm{M} : [\sigma_1, \ldots, \sigma_k]\tau$ *and* $1 \leq j_1 < j_2 < \ldots < j_{n-1} < j_n \leq k$ *such that* $\mathrm{E} : [\sigma_{j_1}, \ldots, \sigma_{j_n}]\tau$ *then*
$\zeta = [\mathrm{M}[\overline{z}] \mapsto \mathrm{E}[z_{j_1}, \ldots, z_{j_n}]]$ *is a (parameter) elimination binding*

Identification of M ***and*** N. *If* $\mathrm{M} : [\sigma_1, \ldots, \sigma_k]\tau$, $\mathrm{N} : [\nu_1, \ldots, \nu_l]\tau$, $\mathrm{I} : [\sigma_1, \ldots, \sigma_k, \nu_1, \ldots, \nu_l]\tau$, $\mathrm{M}_i : [\sigma_1, \ldots, \sigma_k]\nu_i$ *for all* $i \in \{1, \ldots, l\}$*, and* $\mathrm{N}_j : [\nu_1, \ldots, \nu_l]\sigma_j$ *for all* $j \in \{1, \ldots, k\}$ *then*
$\zeta = [\mathrm{M}[\overline{z}] \mapsto \mathrm{I}[\overline{z}, \mathrm{M}_1[\overline{z}], \ldots, \mathrm{M}_l[\overline{z}]],\ \ \mathrm{N}[\overline{y}] \mapsto \mathrm{I}[\mathrm{N}_1[\overline{y}], \ldots, \mathrm{N}_k[\overline{y}], \overline{y}]]$ *is an identification binding*

Iteration for M. *If* $\mathrm{M} : [\sigma_1, \ldots, \sigma_k]\tau$, $\mathsf{F} : (\overline{\alpha_1}.\beta_1, \ldots, \overline{\alpha_n}.\beta_n) \to \gamma$, $\mathrm{H} : [\sigma_1, \ldots, \sigma_k, \gamma]\tau$*, and* $\mathrm{M}_i : [\sigma_1, \ldots, \sigma_k, \overline{\alpha_i}]\beta_i$ *for all* i*, then*
$\zeta = [\mathrm{M}[\overline{z}] \mapsto \mathrm{H}[\overline{z}, \mathsf{F}(\overline{x_1}.\mathrm{M}_1[\overline{z}, \overline{x_1}], \ldots, \overline{x_n}.\mathrm{M}_n[\overline{z}, \overline{x_n}])]]$ *is an iteration binding*

The iteration bindings allow to combine parameters of a metavariable in arbitrary ways. This is also particularly influenced by the fact that the type γ used in the bindings may be arbitrary. This type of bindings introduce arbitrary branching in the procedure below, so should be used with caution in pragmatic implementations. Intuitively, we emphasize two distinct use cases for the iteration bindings:

1. To extract a new term from one or more parameters by application of an axiom. In this case, we use iteration, where the root of one of the sides of an axiom is used as an operator F.
2. To introduce new variables in scope. In this case, any operator that introduces at least one variable into scope is used in an iteration. This use case is important for the completeness of the procedure. See Example 32.

4.2 Transition Rules

We will write each transition rule of the unification procedure in the form $(\Theta \mid \Gamma_\forall \vdash s \overset{?}{=} t : \tau) \xrightarrow{\theta} \langle \Xi, S \rangle$, where $\theta : \Theta \to \Xi$ is a metavariable substitution and S is a new set of constraints that is supposed to replace $s \overset{?}{=} t$. We will often write S instead of $\langle \Xi, S \rangle$ when Ξ is understood from context.

We will now go over the rules that will constitute the E-unification procedure when put in proper order. The first two rules are straightforward.

▶ **Definition 21** (delete)**.** *If a constraint has the same term on both sides, we can* **delete** *it:*

$$(\Theta \mid \Gamma_\forall \vdash t \overset{?}{=} t : \tau) \xrightarrow{\mathsf{id}} \varnothing$$

[4] Note, that Huet-style projection cannot be formulated in pure second-order abstract syntax as it explicitly relies on abs and app. Thus, in E-unification we can recover such projections only by using axioms in some form. Kudasov [20] implements a heuristic that resembles a generalisation of Huet-style projections. We leave proper generalisations for future work.

▶ **Definition 22** (decompose). *We define two variants of this rule:*

1. *Let* $\mathsf{F} : (\overline{\sigma_1}.\tau_1, \ldots, \overline{\sigma_n}.\tau_n) \to \tau$*, then we can* **decompose** *a constraint with* F *on both sides into a set of constraints for each pair of (scoped) subterms:*

$$(\Theta \mid \Gamma_\forall \vdash \mathsf{F}(\overline{x.t}) \overset{?}{=} \mathsf{F}(\overline{x.s}) : \tau) \xrightarrow{\mathsf{id}} \{\Theta \mid \Gamma_\forall, \overline{x_i} : \overline{\sigma_i} \vdash t_i \overset{?}{=} s_i : \tau_i\}^{i \in \{1,\ldots,n\}}$$

2. *Let* $\mathrm{M} : [\sigma_1, \ldots, \sigma_n]\tau$*, then we can* **decompose** *a constraint with* M *on both sides into a set of constraints for each pair of parameters:*

$$(\Theta \mid \Gamma_\forall \vdash \mathrm{M}[\overline{t}] \overset{?}{=} \mathrm{M}[\overline{s}] : \tau) \xrightarrow{\mathsf{id}} \{\Theta \mid \Gamma_\forall \vdash t_i \overset{?}{=} s_i : \sigma_i\}^{i \in \{1,\ldots,n\}}$$

▶ **Example 23.**

$$\begin{aligned}
&\Theta \mid \Gamma = (\mathrm{M} : [\sigma]\sigma \Rightarrow \sigma \mid f : \sigma \Rightarrow \sigma) \\
&\{\Theta \mid \Gamma \vdash \mathsf{abs}(x.\mathsf{app}(\mathrm{M}[x], x)) \overset{?}{=} \mathsf{abs}(x.\mathsf{app}(f, x))\} \\
\xrightarrow{\mathsf{id}} &\{\Theta \mid \Gamma, x : \sigma \vdash \mathsf{app}(\mathrm{M}[x], x)) \overset{?}{=} \mathsf{app}(f, x)\}\} && \text{(decompose)} \\
\xrightarrow{\mathsf{id}} &\{\Theta \mid \Gamma, x : \sigma \vdash \mathrm{M}[x] \overset{?}{=} f, \quad \Theta \mid \Gamma, x : \sigma \vdash x \overset{?}{=} x\} && \text{(decompose)} \\
\xrightarrow{\mathsf{id}} &\{\Theta \mid \Gamma, x : \sigma \vdash \mathrm{M}[x] \overset{?}{=} f\} && \text{(delete)}
\end{aligned}$$

The next two rules are second-order versions of *imitate* and *project* rules used in many HOU algorithms. The idea is that a metavariable can either imitate the other side of the constraint, or simply project one of its parameters:

▶ **Definition 24** (imitate). *For constraints with a metavariable* $\mathrm{M} : [\overline{\sigma_s}]\tau$ *and an operator* $\mathsf{F} : (\overline{\sigma_1}.\tau_1, \ldots, \overline{\sigma_n}.\tau_n) \to \tau$ *we can* **imitate** *the operator side using an imitation binding (metavariables* $\overline{\mathrm{T}}$ *are fresh):*

$$(\Theta \mid \Gamma_\forall \vdash \mathrm{M}[\overline{s}] \overset{?}{=} \mathsf{F}(\overline{x.t}) : \tau) \xrightarrow{[\mathrm{M}[\overline{z_s}] \mapsto \mathsf{F}(\overline{x.\mathrm{T}[\overline{z_s}, \overline{x}]})]} \{\Theta \mid \Gamma_\forall \vdash \mathsf{F}(\overline{x.\mathrm{T}[\overline{s}, \overline{x}]}) \overset{?}{=} \mathsf{F}(\overline{x.t}) : \tau\}$$

Note that **(imitate)** can be followed up by an application of the **(decompose)** rule.

▶ **Definition 25** (project). *For constraints with a metavariable* $\mathrm{M} : [\overline{\sigma_s}]\tau$ *and a term* $u : \tau$*, if* $\sigma_i = \tau$ *then we can produce a JP-style projection binding for the parameter at position* i*:*

$$(\Theta \mid \Gamma_\forall \vdash \mathrm{M}[\overline{s}] \overset{?}{=} u : \tau) \xrightarrow{[\mathrm{M}[\overline{z}] \mapsto z_i]} \{\Theta \mid \Gamma_\forall \vdash s_i \overset{?}{=} u : \tau\}$$

The next rule is concerned with matching one side of a constraint against one side of an axiom. When matching with an axiom, we need to *instantiate* it to the particular use (indeed, an axiom serves as a schema!). However, it is not sufficient to simply map metavariables of the axiom into fresh metavariables of corresponding types. Since we are instantiating axiom for a particular constraint which may have a non-empty $\Gamma_\forall$, it is important to add all those variables to each of the fresh metavariables[5]:

▶ **Definition 26.** *Let* $\Gamma_\forall = (\overline{x} : \overline{\alpha})$ *and* $\xi : \Xi \mid \cdot \to \Theta \mid \Gamma_\forall$*.*
We say ξ **instantiates the axiom** $\Xi \mid \cdot \vdash l \equiv r : \tau$ **in context** $\Theta \mid \Gamma_\forall$ *if*

1. *for any* $(\mathrm{M}_i : [\overline{\sigma}]\tau) \in \Xi$*,* ξ *maps* $\mathrm{M}_i[\overline{t}]$ *to* $\mathrm{N}_i[\overline{t}, \overline{x}]$*;*
2. $\mathrm{N}_i = \mathrm{N}_j$ *iff* $i = j$ *for all* i, j*.*

[5] This is different to E-unification with first-order axioms, where metavariables do not carry their own context and can be unified with an arbitrary variable later.

▶ **Example 27.** Let $\xi = [\mathrm{M}[z] \mapsto \mathrm{M}_1[z, g, y], \mathrm{N}[] \mapsto \mathrm{N}_1[g, y]]$. Then, ξ instantiates the axiom

$$\mathrm{M} : [\sigma]\tau, \mathrm{N} : []\sigma \mid \cdot \vdash \mathsf{app}(\mathsf{abs}(x.\mathrm{M}[x]), \mathrm{N}[]) \equiv \mathrm{M}[\mathrm{N}[]] : \tau$$

in context $\mathrm{M}_1 : [\sigma, \sigma \Rightarrow \tau, \sigma]\tau, \mathrm{N}_1 : [\sigma \Rightarrow \tau, \sigma]\sigma \mid g : \sigma \Rightarrow \tau, y : \sigma$.

▶ **Definition 28** (mutate). *For constraints where one of the sides matches*[6] *an axiom in E:*

$$\Xi \mid \cdot \vdash l \equiv r : \tau$$

We rewrite the corresponding side (here, ξ instantiates the axiom in context $\Theta \mid \Gamma_\forall$).

$$(\Theta \mid \Gamma_\forall \vdash t \overset{?}{=} s : \tau) \overset{\mathsf{id}}{\rightarrow} \{\Theta \mid \Gamma_\forall \vdash t \overset{?}{=} \xi l : \tau\} \uplus \{\Theta \mid \Gamma_\forall \vdash \xi r \overset{?}{=} s : \tau\}$$

In general, we may rewrite in both directions. However, it may be pragmatic to choose a single direction to some of the axioms (e.g. $\beta\eta$-reductions), while keeping others bidirectional (e.g. commutativity and associativity axioms). Note that, unlike previous rules, the **(mutate)** rule can lead to infinite transition sequences.

The remaining rules deal with constraints with metavariables on both sides. One rule attempts to unify distinct metavariables:

▶ **Definition 29** (identify). *When a constraint consists of a pair of distinct metavariables $\mathrm{M} : [\sigma_1, \ldots, \sigma_k]\tau$ and $\mathrm{N} : [\gamma_1, \ldots, \gamma_l]\tau$, we can use an identification binding (metavariables $\mathrm{I}, \overline{\mathrm{M}'}, \overline{\mathrm{N}'}$ are fresh):*

$$(\Theta \mid \Gamma_\forall \vdash \mathrm{M}[\overline{s}] \overset{?}{=} \mathrm{N}[\overline{t}]) \xrightarrow{[\mathrm{M}[\overline{z}]\mapsto\mathrm{I}[\overline{z},\overline{\mathrm{M}'[\overline{z}]}],\mathrm{N}[\overline{y}]\mapsto\mathrm{I}[\overline{\mathrm{N}'[\overline{y}]},\overline{y}]]} \{\Theta \mid \Gamma_\forall \vdash \mathrm{I}[\overline{s}, \overline{\mathrm{M}'[\overline{s}]}] \overset{?}{=} \mathrm{I}[\overline{\mathrm{N}'[\overline{u}]}, \overline{u}]\}$$

Another rule attempts to unify identical metavariables with distinct lists of parameters:

▶ **Definition 30** (eliminate). *When a constraint has the same metavariable $\mathrm{M} : [\sigma_1, \ldots, \sigma_n]\tau$ on both sides and there is a sequence $(j_k)_{k=1}^n$ such that $s_{j_k} = t_{j_k}$ for all $k \in \{1, \ldots, n\}$, then we can* **eliminate** *every other parameter and leave the remaining terms identical (metavariable E is fresh):*

$$(\Theta \mid \Gamma_\forall \vdash \mathrm{M}[\overline{s}] \overset{?}{=} \mathrm{M}[\overline{t}]) \xrightarrow{[\mathrm{M}[\overline{z}]\mapsto\mathrm{E}[z_{j_1},\ldots,z_{j_n}]]} \varnothing$$

The idea of the final rule is to extend a list of parameters with some combination of those that exist already. For example, consider constraint $\forall x, y, z.\mathrm{M}[\mathsf{pair}(x, y), z] \overset{?}{=} \mathrm{N}[x, z]$. It is clear, that if we can work with a pair of x and y, then we can work with them individually, since we can extract x using fst and y using snd. Thus, a substitution $[\mathrm{M}[p, z] \mapsto \mathrm{M}_1[p, z, \mathsf{fst}(p)]]$ would result in a new constraint $\forall x, y, z.\mathrm{M}_1[\mathsf{pair}(x, y), z, \mathsf{fst}(\mathsf{pair}(x, y))] \overset{?}{=} \mathrm{N}[x, z]$. This one can now be solved by applying **(identify)**, **(eliminate)**, and **(decompose)** rules that will lead us to $\forall x, y, z.\mathsf{fst}(\mathsf{pair}(x, y)) \overset{?}{=} x : \sigma$ which will be processed using **(mutate)** rule.

▶ **Definition 31** (iterate). *When a constraint consists of a pair of (possibly, identical) metavariables $\mathrm{M} : [\sigma_1, \ldots, \sigma_k]\tau$ and $\mathrm{N} : [\gamma_1, \ldots, \gamma_l]\tau$, we can use an iteration binding (metavariables $\mathrm{H}, \overline{\mathrm{K}}$ are fresh):*

$$(\Theta \mid \Gamma_\forall \vdash \mathrm{M}[\overline{s}] \overset{?}{=} \mathrm{N}[\overline{t}]) \xrightarrow{[\mathrm{M}[\overline{z}]\mapsto\mathrm{H}[\overline{z},\mathsf{F}(\overline{\overline{x}.\mathrm{K}[\overline{z},\overline{x}]})]]} \{\Theta \mid \Gamma_\forall \vdash \mathrm{H}[\overline{s}, \mathsf{F}(\overline{\overline{x}.\mathrm{K}[\overline{z}, \overline{x}]})] \overset{?}{=} \mathrm{N}[\overline{t}]\}$$

[6] We check that the roots of terms match. Technically, we do not have to perform this check and apply **(mutate)** rule for any axiom (non-deterministically), since full matching will be performed by the unification procedure.

The following example demonstrates the importance of iteration by an arbitrary operator to introduce variables into scope:

▶ **Example 32.** Consider a unification problem for simply-typed λ-calculus:

$$\begin{aligned}&\exists \mathrm{M} : [\sigma \Rightarrow \sigma \Rightarrow \tau](\sigma \Rightarrow \tau)\\ &\quad \forall f : \sigma \Rightarrow \sigma \Rightarrow \sigma \Rightarrow \sigma \Rightarrow \tau.\\ &\qquad \mathrm{M}[\lambda x.\lambda y.f\ x\ y\ x\ x] \overset{?}{=} \mathrm{M}[\lambda x.\lambda y.f\ y\ y\ x\ y] : \sigma \Rightarrow \tau\end{aligned}$$

It has the following E-unifier: $\zeta = [\mathrm{M}[g] \mapsto \lambda z.g\ z\ z]$. To construct this unifier from bindings, we start with iteration binding $[\mathrm{M}[g] \mapsto \mathrm{I}[g, \lambda z.\mathrm{M}_1[g, z]]]$, introducing the lambda abstraction, which is followed by a projection $[\mathrm{I}[g, r] \mapsto r]$, which is followed by another iteration (to introduce application), and so on.

Finally, we compile all transition rules into the unification procedure:

▶ **Definition 33.** *The **E-unification procedure** over an equational presentation E is defined by repeatedly applying the following transitions (non-deterministically) until a stop:*

1. *If no constraints are left, then stop **(succeed)**.*
2. *If possible, apply **(delete)** rule.*
3. *If possible, apply **(mutate)** or **(decompose)** rule (non-det.).*
4. *If there is a constraint consisting of two non-metavariables and none of the above transitions apply, stop **(fail)**.*
5. *If there is a constraint $\mathrm{M}[\ldots] \overset{?}{=} \mathsf{F}(\ldots)$, apply **(imitate)** or **(project)** rules (non-det.).*
6. *If there is a constraint $\mathrm{M}[\ldots] \overset{?}{=} x$, apply **(project)** rules (non-det.).*
7. *If possible, apply **(identify)**, **(eliminate)**, or **(iterate)** rules (non-det.).*
8. *If none of the rules above are applicable, then stop **(fail)**.*

Many HOU algorithms [23, 21] implement a rule (typically called *eliminate*) that allows to eliminate metavariables, when a corresponding constraint is in solved form. Such a rule is not necessary here, as it is covered by a combination of **(imitate)**, **(decompose)**, **(delete)**, **(identify)**, and **(eliminate)** rules. However, it simplifies presentation of examples and also serves as a practical optimisation, so we include it as an optional rule:

▶ **Definition 34** (eliminate*)**.** *When a constraint $C = (\Theta \mid \Gamma_\forall \vdash \mathrm{M}[\overline{z}] \overset{?}{=} u)$ is in solved form, we can eliminate it with a corresponding unifier $\xi_{\{C\}} = [\mathrm{M}[\overline{z}] \mapsto u]$:*

$$(\Theta \mid \Gamma_\forall \vdash \mathrm{M}[\overline{s}] \overset{?}{=} u) \xrightarrow{[\mathrm{M}[\overline{z}] \mapsto u]} \varnothing$$

The **(eliminate*)** rule should have the same priority as **(delete)** in the procedure.

▶ **Lemma 35.** *In the procedure defined in Definition 33, each step is sound. That is, if $S \xrightarrow{\theta} S'$ is a single-step transition that the procedure takes and $\xi \in U_E(S')$ then $\xi \circ \theta \in U_E(S)$.*

Proof. It is sufficient to show that each step is sound with respect to the constraint it acts upon. That is, we consider the step $\{C\} \xrightarrow{\theta} S''$ such that $C \in S$ and $S'' \subseteq S'$. By assumption $\xi \in U_E(S')$ and thus also $\xi \in U_E(S'')$. Note that for any constraint $D \in (S - \{C\})$ we have a corresponding constraint $D' \in (S' - S'')$ such that $D' = \theta D$. Since ξ unifies D' it follows that $\xi \circ \theta$ unifies D. Thus, it is enough for us to show that $\xi \circ \theta$ unifies $U_E(\{C\})$.

We now go over the list of possible steps:

- **(delete)**: it is clear that any substitution unifies C;
- **(decompose)**: since ξ unifies all subterm pairs in S'', it also unifies C;
- **(imitate)**, **(project)**, **(identify)**, **(eliminate)**, **(iterate)**: all of these rules simply make a decision on how to substitute some metavariables (choose θ) and immediately apply that substitution. So, $S'' = \{\theta C\}$ and since ξ unifies S' then $\xi \circ \theta$ unifies S.
- **(mutate)**: let $C = (\Theta \mid \Gamma_\forall \vdash s \overset{?}{=} t : \tau)$ and we mutate according to axiom $(\Xi \mid \cdot \vdash l \equiv r : \tau) \in E$ with substitution ζ instantiating this axiom. By assumption, ξ unifies both $s \overset{?}{=} \zeta l$ and $\zeta r \overset{?}{=} t$. Also, $\Theta \mid \Gamma_\forall \vdash \zeta l \equiv_E \zeta r : \tau$. In this rule, $\theta = \mathsf{id}$, and so we can show that $\xi \circ \theta = \xi$ unifies $s \overset{?}{=} t$: $\xi s \equiv_E \xi(\zeta l) \equiv_E \xi(\zeta r) \equiv_E \xi t$ ◀

▶ **Theorem 36.** *The procedure defined in Definition 33 is sound. That is, if* $S \xrightarrow{\theta_1} S_1 \xrightarrow{\theta_2} \ldots \xrightarrow{\theta_n} \varnothing$ *is a path produced by the procedure, then* $\theta_1 \circ \theta_2 \circ \ldots \circ \theta_n \in U_E(S)$*.*

Proof. Direct corollary of Lemma 35. ◀

5 Proof of Completeness

In this section we prove our main theorem, showing that our unification procedure is complete.

We start with a definition of mixed operators:

▶ **Definition 37.** *We say that an operator* $\mathsf{F} : (\overline{\alpha_1}.\beta_1, \ldots, \overline{\alpha_n}.\beta_n) \to \gamma$ *is* **mixed** *iff* α_i *is empty and* α_j *is not empty for some* i *and* j*.*

Dealing with mixed operators can be very non-trivial. In the following theorem, we assume that all operators either introduce scopes in all subterms, or in none. That is, for each operator $\mathsf{F} : (\overline{\alpha_1}.\beta_1, \ldots, \overline{\alpha_n}.\beta_n) \to \gamma$, either $|\overline{\alpha_i}| = 0$ for all i or $|\overline{\alpha_i}| > 0$ for all i. The assumption is justified since we can always encode a mixed operator as a combination of non-mixed operators. For example, $\mathsf{let}(t_1, x.t_2)$ can be encoded as $\mathsf{let}(t_1, \mathsf{block}(x.t_2))$.

▶ **Theorem 38.** *Assuming no mixed operators are used, the procedure described in Definition 33 is complete, meaning that all paths from a root to all* **(success)** *leaves in the search tree constructed by the procedure, form a complete (but not necessarily minimal) set of E-unifiers. More specifically, let* E *be an equational presentation and* $\langle \Theta, S \rangle$ *be an E-unification problem. Then for any E-unifier* $\theta \in U_E(S)$ *there exists a path* $S \xrightarrow{\xi} \varnothing$ *such that* $\xi \preccurlyeq_E \theta$*.*

▶ Remark 39. The unification procedure may produce redundant unifiers. For example, consider the following unification problem in simply-typed λ-calculus:

$$\exists \mathrm{M} : [\sigma, \sigma]\tau, \mathrm{N} : [\sigma](\sigma \Rightarrow \tau).\ \forall x : \sigma.\ \mathrm{M}[x, x] \overset{?}{=} \mathsf{app}(\mathrm{N}[x], x)$$

Depending on whether we start with the **(imitate)** rule or the **(mutate)** first, we can arrive at the following unifiers:

$$\theta_1 = [\mathrm{M}[z_1, z_2] \mapsto \mathsf{app}(\mathrm{N}[z_1], z_2)]$$
$$\theta_2 = [\mathrm{M}[z_1, z_2] \mapsto \mathrm{T}[z_2, z_1],\ \mathrm{N}[z_1] \mapsto \mathsf{abs}(z.\mathrm{T}[z_1, z])]$$

It is clear that $\theta_1 \neq \theta_2$, but $\theta_1 \preccurlyeq_E \theta_2$ (witnessed by $[\mathrm{N}[z_1] \mapsto \mathsf{abs}(z.\mathrm{T}[z_1, z])]$). Hence, the set of E-unifiers produced for this unification problem is not minimal (by Definition 14).

Our proof is essentially a combination of the two approaches: one by Gallier and Snyder in their proof of completeness for general (first-order) E-unification [11], and another one by Jensen and Pietrzykowski (JP) [16], refined by Vukmirovic, Bentkamp, and Nummelin

(VBN) [34] for full higher-order unification. In particular, we need to reuse some of the ideas from the latter when dealing with parametrised metavariables. However, we cannot reuse the idea of JP's ω-simplicity or VBN's base-simplicity, as those are dependent crucially on the η-long terms, λ-abstraction, function application, which are not accessible to us in a general second-order abstract syntax. Instead, we reuse the ideas of Gallier and Snyder to understand when it is okay to decompose terms. To understand when to apply **(iterate)** rule, we also look at the rewrite sequence instead of ω-simplicity of terms.

The main idea of the proof is to take the unification problem S together with its E-unifier θ and then choose one of the rules of the procedure guided by θ. Applying a rule updates constraints and the remaining substitution is also updated. To show that this process terminates, we introduce a measure that strictly decreases with each rule application.

▶ **Definition 40.** *Let $\theta \in U_E(S)$. Then, define the* **measure** *on pairs $\langle S, \theta\rangle$ as the lexicographic comparison of*

1. *sum of lengths of the rewriting sequences $\theta s \overset{*}{\longleftrightarrow}_E \theta t$ for all $s \overset{?}{=} t$ of S;*
2. *total number of operators used in θ;*
3. *total number of metavariables used in θ;*
4. *sum of sizes of terms in S.*

We denote the quadruple above as $\mathsf{ord}(S, \theta)$.

The following definition helps us understand when we should apply the **(project)** rule:

▶ **Definition 41.** *A metavariable $\mathrm{M} : [\overline{\sigma}]\tau$ is* **projective at** *j* **relative to** *θ if $\theta\mathrm{M}[\overline{z}] = z_j$.*

One of the crucial points in the proof is to understand whether we can apply **(identify)** or **(eliminate)** rules for constraints with two metavariables on both sides. The following lemma provides precise conditions for this, allowing for **(identify)** when metavariables are distinct or **(eliminate)** when they are equal.

▶ **Lemma 42.** *Let $s = \mathrm{M}[\overline{u}]$ and $t = \mathrm{N}[\overline{v}]$ such that $\zeta s \overset{*}{\longleftrightarrow}_E \zeta t$. Let $s_1, \ldots, s_n$ be the subterms of ζs, $t_1, \ldots, t_n$ the subterms of ζt such that the rewriting sequence $\zeta s \overset{*}{\longleftrightarrow}_E \zeta t$ corresponds to the union of independent rewritings $s_i \overset{*}{\longleftrightarrow}_E t_i$ for all $i \in \{1, \ldots, n\}$. If for all i we have either that s_i is a subterm of an occurrence of ζu_{j_i} or that t_i is a subterm of an occurrence of ζv_{j_i}, then there exist terms w, $\overline{u'}$, $\overline{v'}$ such that*

1. *$\overline{u'}[\overline{y} \mapsto \zeta\overline{v}] \equiv_E \zeta\overline{u}$ and $\overline{v'}[\overline{z} \mapsto \zeta\overline{u}] \equiv_E \zeta\overline{v}$,*
2. *$\zeta\mathrm{M}[\overline{z}] = w[\overline{y} \mapsto \overline{v'}]$ and $\zeta\mathrm{N}[\overline{y}] = w[\overline{z} \mapsto \overline{u'}]$.*

Proof. We define an auxiliary family of terms $\Xi \mid \overline{x} : \overline{\alpha} \vdash w'(l, r) : \tau$ for pairs of terms $\Xi \mid \overline{x} : \overline{\alpha}, \overline{z} : \overline{\gamma} \vdash l : \tau$ and $\Xi \mid \overline{x} : \overline{\alpha}, \overline{y} : \overline{\beta} \vdash r : \tau$ such that l is a subterm of $\zeta\mathrm{M}[\overline{z}]$ and r is a subterm of $\zeta\mathrm{N}[\overline{y}]$ satisfying $l[\overline{z} \mapsto \zeta\overline{u}] \equiv_E r[\overline{y} \mapsto \zeta\overline{v}]$. We define $w'(l, r)$ inductively on the structure of l and r, maintaining $l \equiv_E w'(l, r)[\overline{y} \mapsto \overline{v'}]$ and $r \equiv_E w'(l, r)[\overline{z} \mapsto \overline{u'}]$:

1. if $l = x_i$ or $r = x_i$, then $l = r$ and $w'(l, r) = x_i$;
2. if $l = z_k$ then $w'(l, r) = z_k$ and $u'_k = r$;
3. if $r = y_k$ then $w'(l, r) = y_k$ and $v'_k = l$;
4. if $l = \mathsf{F}(\overline{\overline{a}.p})$ and $r = \mathsf{F}(\overline{\overline{a}.q})$ then $w'(l, r) = \mathsf{F}(\overline{\overline{a}.w'(p, q)})$; note that $w'(p_i, q_i)$ is well-defined for all i, since $l[\overline{z} \mapsto \zeta\overline{u}] \equiv_E r[\overline{y} \mapsto \zeta\overline{v}]$ implies component-wise equality $p_i[\overline{z} \mapsto \zeta\overline{u}] \equiv_E q_i[\overline{y} \mapsto \zeta\overline{v}]$ for all i. This is true, since otherwise we are rewriting (at root) both l and r, but $l[\overline{z} \mapsto \zeta\overline{u}]$ nor $r[\overline{y} \mapsto \zeta\overline{v}]$ corresponds to a parameter occurrence ζu_j or ζv_j in terms ζs or ζt correspondingly.
5. if $l = \mathrm{M}[\overline{p}]$ and $r = \mathrm{M}[\overline{q}]$ then $w'(l, r) = \mathrm{M}[\overline{w'(p, q)}]$; here, $w'(p_i, q_i)$ is well-defined for all i, similarly to the previous case.

If u_k (or v_k) has not been defined for some k, it means that a corresponding parameter is not essential and can be eliminated. We set[7] such u_k to be a fresh metavariable $\mathrm{U}_k[]$. We set $w = w'(\zeta\mathrm{M}[\overline{z}], \zeta\mathrm{N}[\overline{y}])$. By construction, $\zeta\mathrm{M}[\overline{z}] = w[\overline{y} \mapsto \overline{v'}]$ and $\zeta\mathrm{N}[\overline{y}] = w[\overline{z} \mapsto \overline{u'}]$. ◀

▶ **Corollary 43.** *Let $s = \mathrm{M}[\overline{u}]$ and $t = \mathrm{M}[\overline{v}]$ such that $\zeta s \overset{*}{\longleftrightarrow}_E \zeta t$. Let $s_1, \ldots, s_n$ be the subterms of ζs, $t_1, \ldots, t_n$ the subterms of ζt such that the rewriting sequence $\zeta s \overset{*}{\longleftrightarrow}_E \zeta t$ corresponds to the union of independent rewritings $s_i \overset{*}{\longleftrightarrow}_E t_i$ for all $i \in \{1, \ldots, n\}$. If for all i we have either that s_i is a subterm of an occurrence of ζu_{j_i} or that t_i is a subterm of an occurrence of ζv_{j_i}, then, there exists a sequence $1 \leq j_1 < \ldots < j_k \leq$ such that and $FV(\zeta\mathrm{M}[\overline{z}]) = \{z_{j_1}, \ldots, z_{j_k}\}$ and $u_{j_i} \equiv_E v_{j_i}$ for all i.*

The following lemma will help us generalize solutions in Item 2(e)iii of the proof below.

▶ **Lemma 44.** *Let $\Xi \mid \overline{x} : \overline{\alpha} \vdash w : \sigma$ be a subterm of $\Xi \mid \overline{x} : \overline{\alpha} \vdash t : \tau$. If t does not contain mixed operators, then there exists a substitution $\zeta_{w,t} = [\mathrm{H}[z, \overline{y}] \mapsto h]$ and a collection of terms $\Xi \mid \overline{x} : \overline{\alpha} \vdash \overline{s} : \overline{\beta}$, such that each s_i is a subterm of t and $\zeta_{w,t}\mathrm{H}[w, \overline{s}] = t$.*

Proof. Note that w and $\overline{s}$ are subterms of t and are not under binders (since they have the same variable context). Then, by induction on the structure of t:

1. if $t = w$, then $\zeta_{w,t} = [\mathrm{H}[z] \mapsto z]$;
2. if $t = \mathsf{F}(\overline{x_1}.t_1, \ldots, \overline{x_n}.t_n)$ then, since t does not contain mixed operators, $\overline{x_i}$ is empty for all i. Now, if w is a subterm of t_i and $\zeta_{w,t_i} = [\mathrm{H}_{t_i}[z, y'_1, \ldots, y'_k] \mapsto h_{t_i}]$ then $\zeta = [\mathrm{H}[z, y'_1, \ldots, y'_k, y_1, \ldots, y_{n-1}] \mapsto \mathsf{F}(y_1, \ldots, y_{i-1}, h_{t_i}, y_i, \ldots, y_{i-1})]$.
3. if $t = \mathrm{N}[t_1, \ldots, t_n]$ such that w is a subterm of t_i and $\zeta_{w,t_i} = [\mathrm{H}_{t_i}[z, y'_1, \ldots, y'_k] \mapsto h_{t_i}]$ then $\zeta = [\mathrm{H}[z, y'_1, \ldots, y'_k, y_1, \ldots, y_{n-1}] \mapsto \mathrm{N}[y_1, \ldots, y_{i-1}, h_{t_i}, y_i, \ldots, y_{i-1}]]$.

Note that case of $t = x$ is impossible unless $t = w$ (case 1). ◀

We are now ready to prove Theorem 38.

Proof of Theorem 38. Let $S_0 = S$ and $\theta_0 = \rho \circ \theta$, where ρ is some renaming substitution such that every metavariable occurring in $\theta_0 S_0$ does not occur in S_0. Note that θ_0 is an E-unifier of S, since θ is by assumption.

We now inductively define S_i, ξ_i, and θ_i until we reach some i such that $S_i = \varnothing$. We ensure that $\mathsf{ord}(S_i, \theta_i)$ decreases with every step, so that such sequence of steps would always terminate. We maintain the following invariants for each step:

1. $\langle S_i, \theta_i \rangle \xrightarrow{\xi_i} \langle S_{i+1}, \theta_{i+1} \rangle$ where $S_i \xrightarrow{\xi_i} S_{i+1}$ by some rule of the unification procedure;
2. $\mathsf{ord}(S_{i+1}, \theta_{i+1}) < \mathsf{ord}(S_i, \theta_i)$;
3. $\theta_i \in U_E(S_i)$;
4. $\theta_0 \equiv_E \theta_i \circ \xi_{i-1} \circ \ldots \circ \xi_0$;
5. every free variable occurring in $\theta_i S_i$ does not occur in S_i;

If $S_i \neq \varnothing$, then let $\forall \overline{x} : \overline{\sigma}. s \overset{?}{=} t : \tau$ be a constraint in S_i. We consider two major cases with respect to the rewriting sequence $\Theta_i \mid \overline{x} : \overline{\sigma} \vdash \theta_i s \overset{*}{\longleftrightarrow}_E \theta_i t : \tau$:

1. **The rewriting sequence contains a root rewrite.** More precisely, there exists a sequence $\theta_i s = u_0 \longleftrightarrow_E \ldots \longleftrightarrow_E u_n = \theta_i t$ and some term u_j such that $u_j \longleftrightarrow_E u_{j+1}$ is a direct application of a rewrite using an axiom. This means that s and t can be unified by a direct use of an axiom. More specifically, there exists an instantiation ζ of an axiom $\Xi \mid \cdots \vdash l \equiv r : \tau$ from E such that $\zeta l = u_j$, $u_{j+1} = \zeta r$, and θ_i unifies both

[7] alternatively, we could have adjusted the statement to only mention subsets of variables $\overline{z}$ and $\overline{y}$ that are used at least once

$s \overset{?}{=} u_j$ and $u_{j+1} \overset{?}{=} t$. Thus, we can apply **(mutate)** rule. The measure decreases since the rewrite sequence $s \overset{*}{\longleftrightarrow}_E t$ is now split into two sequences $s \overset{*}{\longleftrightarrow} u_j$ and $u_{j+1} \overset{*}{\longleftrightarrow}_E t$ such that sum of lengths of new sequences is exactly one less than the length of the original sequence.

2. **Rewriting sequence is empty or does not contain a root rewrite.** This means that rewrites may only happen in subterms.
   a. If $s = x$ and $t = y$ where x and y are variables, then $x = y$ and we can apply **(delete)** rule, with $\xi_i = \mathsf{id}$ and $\theta_{i+1} = \theta_i$. The measure is reduced since the total size of constraints is reduced, while the rewriting sequences and the remaining substitution remain the same.
   b. If $s = \mathsf{F}(\overline{\overline{z}.u})$ and $t = \mathsf{F}(\overline{\overline{z}.v})$, then, $\theta_i s = \mathsf{F}(\overline{\overline{z}.\theta_i u})$ and $\theta_i t = \mathsf{F}(\overline{\overline{z}.\theta_i v})$. Since there are no root rewrites, θ_i unifies each pair $u_j \overset{?}{=} v_j$ in corresponding extended contexts, so we can apply **(decompose)** rule with $\xi_i = \mathsf{id}$ and $\theta_{i+1} = \theta_i$. Note that the chain of rewrites may be split into several chains, but the total sum of lengths remains the same. Second component of the measure also remains unchanged. We reduce the third component of the measure, since the total size of terms in the unification problem decreases, the sum of chains of rewrites is unchanged.
   c. If $s = \mathrm{M}[\overline{u}]$ and M is projective at j relative to θ_i then we can apply **(project)** rule with $\xi_i = [\mathrm{M}[\overline{z}] \mapsto z_j]$. Note that the chain of rewrites remains unchanged and ξ_i does not take any operators away from θ_{i+1} (which is a restriction of θ_i to metavariables other than M). We reduce the measure by reducing the total size of terms in the unification problem.
   d. If $s = \mathrm{M}[\overline{u}]$ where M is not projective relative to θ_i and $\theta_i s = \mathsf{F}(\overline{\overline{z}.u})$ and $t = \mathsf{F}(\overline{\overline{z}.v})$, then θ_i unifies each pair $u_i \overset{?}{=} \zeta v_i$ in corresponding extended contexts and we can apply **(imitate)** rule with $\xi_i = [\mathrm{M}[\overline{z}] \mapsto \mathsf{F}(\overline{\overline{x}.\mathrm{T}[\overline{z}, \overline{x}]})]$. Let $\theta_i \mathrm{M}[\overline{z}] = \mathsf{F}(\overline{\overline{x}.w})$, then θ_{i+1} is constructed from θ_i by removing mapping for M and adding mappings $[\mathrm{T}_j[\overline{z}, \overline{x_j}] \mapsto w_j]$ for all j. The chain of rewrites is unchanged and the measure decreases since we reduce the number of operators in θ_{i+1}.
   e. If $s = \mathrm{M}[\overline{u}]$ where M is not projective relative to θ_i and $\theta_i s \overset{*}{\longleftrightarrow}_E \theta_i t$ contains a rewrite of a subterm w in $\theta_i s$ that is not a subterm of an occurrence of $\theta_i u_i$ for some i, then
      i. If w is under a binder in $\theta_i s$, we take the outermost operator F that binds a variable captured by w (that is, $\theta_i s = \ldots \mathsf{F}(\overline{y_1}.s_1, \overline{y_j}.\ldots w \ldots, \overline{y_n}.s_n))$ and apply **(iterate)** rule with $\xi_i = [\mathrm{M}[\overline{z}] \mapsto \mathrm{M}'[\overline{z}, \mathsf{F}(\overline{y_1}.\mathrm{M}_1[\overline{z}, \overline{y_1}], \ldots, \overline{y_n}.\mathrm{M}_n[\overline{z}, \overline{y_n}])]$. Let $\theta_i \mathrm{M}[\overline{z}] = \mathsf{F}(\overline{\overline{y}.s'})$, then θ_{i+1} is defined as θ_i with mapping of M removed and added mappings $[\mathrm{M}_i[\overline{z}, \overline{y_i}] \mapsto s_i]^{\{i \in \{1,\ldots,n\}\}}$. The chain of rewrites remains unchanged and the number of operators in θ_{i+1} decreases by one, so the measure decreases.
      ii. If $w = \mathsf{F}(\ldots)$ and is not under a binder, then we apply **(iterate)** rule with $\xi_i = [\mathrm{M}[\overline{z}] \mapsto \mathrm{M}'[\overline{z}, \mathsf{F}(\overline{y_1}.\mathrm{M}_1[\overline{z}, \overline{y_1}], \ldots, \overline{y_n}.\mathrm{M}_n[\overline{z}, \overline{y_n}])]$. We define θ_{i+1} and show that the measure decreases analogously to the previous case.
      iii. If $w = \mathrm{W}[\overline{v}]$ and is not under a binder, then $\theta_i \mathrm{M}[\overline{z}]$ contains $w' = \mathrm{W}[\overline{v'}]$ as a subterm and $v'_i[\overline{z} \mapsto \overline{u}] = v_i$ for all i (this is because w is not a subterm of any of the $\theta_i u_j$). Since w' is also not under binder, then by Lemma 44 and assumption of no mixed operators we have that there exist terms h, $\overline{s}$, and a substitution $\zeta = [\mathrm{H}[\overline{z}, \overline{y}] \mapsto h]$ such that $\zeta h[\overline{v'}, \overline{s}] = \theta_i \mathrm{M}[\overline{z}]$. Set $\theta'_i = [\mathrm{M}[\overline{z}] \mapsto h[\overline{y} \mapsto \overline{s}]]$. We have $\theta_i = \zeta \circ \theta'_i$, that is θ'_i is more general that θ_i modulo E. The rewriting sequence remains unchanged. If $\theta_i s$ has an operator at root, then θ'_i has fewer operators which decreases the measure. If $\theta_i s$ has a metavariable at root, then θ'_i has fewer metavariables which decreases the measure.

f. If $s = \mathrm{M}[\overline{u}]$ and $t = \mathrm{N}[\overline{v}]$ where $\mathrm{M} \neq \mathrm{N}$, both M and N are not projective relative to θ_i and $\theta_i s \overset{*}{\longleftrightarrow}_E \theta_i t$ corresponds to the union of independent rewritings $s_i \overset{*}{\longleftrightarrow}_E t_i$ for all $i \in \{1, \ldots, n\}$ such that for all i we have either that s_i is a subterm of an occurrence of ζu_{j_i} or that t_i is a subterm of an occurrence of ζv_{j_i}, then by Lemma 42 there exist terms w, $\overline{u'}$, $\overline{v'}$ such that $\theta_i \mathrm{M}[\overline{z}] = w[\overline{y} \mapsto \overline{v'}]$ and $\theta_i \mathrm{N}[\overline{y}] = w[\overline{z} \mapsto \overline{u'}]$. We now can apply **(identify)** rule with $\xi_i = [\mathrm{M}[\overline{z}] \mapsto \mathrm{W}[\overline{y} \mapsto \overline{\mathrm{Y}[\overline{z}]}]\mathrm{N}[\overline{y}] \mapsto \mathrm{W}[\overline{z} \mapsto \overline{\mathrm{Z}[\overline{y}]}]]$. We define θ_{i+1} to be defined as θ_i without mappings for M and N and with added mappings $[\mathrm{W}[\overline{z}, \overline{y}] \mapsto w, \overline{\mathrm{Y}[\overline{z}]} \mapsto \overline{v'}, \overline{\mathrm{Z}[\overline{y}]} \mapsto \overline{u'}]$. The rewriting sequence remains unchanged, but by Lemma 42 the term w is not a variable, so there is an operator or a metavariable that was mentioned twice in θ_i (once for M and once for N) and is now mentioned once in θ_{i+1} (for W), so the number of operators or metavariables in θ_{i+1} decreases by at least 1. Thus, the measure decreases.

g. If $s = \mathrm{M}[\overline{u}]$ and $t = \mathrm{M}[\overline{v}]$ where M is not projective relative to θ_i and $\theta_i s \overset{*}{\longleftrightarrow}_E \theta_i t$ corresponds to the union of independent rewritings $s_i \overset{*}{\longleftrightarrow}_E t_i$ for all $i \in \{1, \ldots, n\}$ such that for all i we have either that s_i is a subterm of an occurrence of ζu_{j_i} or that t_i is a subterm of an occurrence of ζv_{j_i}, then by Corollary 43 we have $\overline{z'} = FV(\theta_i \mathrm{M}[\overline{z}]) \subseteq \overline{z}$ such that for each $z_k \in \overline{z'}$ we have $\theta_i u_k \equiv_E \theta_i v_k$. Consider two subcases:

   i. If $\overline{z'} = \overline{z}$ we apply **(decompose)** rule with $\xi_i = \mathsf{id}$ and $\theta_{i+1} = \theta_i$. The chain of rewrites remains, the remaining substitution is unchanged, but the total size of constraints is reduced, so the measure decreases.

   ii. If $\overline{z'} \subset \overline{z}$ we apply **(eliminate)** rule with $\xi_i = [\mathrm{M}[\overline{z}] \mapsto \mathrm{E}[\overline{z'}]]$ and θ_{i+1} defined as a version of θ_i with removed mapping for M and added mapping $[\mathrm{E}[\overline{z'}] \mapsto \theta_i s]$. Note that the chain of rewrites remains unchanged and ξ_i does not take any operators away from θ_{i+1}. We reduce the measure by reducing the total size of terms in the unification problem (as at least one parameter is removed from at least one metavariable M).

We now have a sequence $\langle S_0, \theta_0 \rangle \xrightarrow{\xi_0} \langle S_1, \theta_1 \rangle \xrightarrow{\xi_1} \ldots$. The sequence is finite since the measure $\mathsf{ord}(S_i, \theta_i)$ strictly decreases with every step. Therefore, $\langle S, \theta_0 \rangle \xrightarrow{\xi_0} \ldots \xrightarrow{\xi_n} \langle \varnothing, \mathsf{id} \rangle$ and $\theta \equiv_E \rho^{-1} \circ \theta_i \circ \xi_{i-1} \circ \ldots \xi_0 \equiv_E \rho^{-1} \circ \xi_n \circ \ldots \circ \xi_0 \preccurlyeq_E \xi_n \circ \ldots \circ \xi_0$, completing the proof. ◀

6 Discussion

A pragmatic implementation of our procedure may enjoy the following changes. We find that these help make a reasonable compromise between completeness and performance:

1. remove **(iterate)** rule; this rule sacrifices completeness, but helps significantly reduce non-determinism; the solutions lost are also often highly non-trivial and might be unwanted in certain applications such as type inference;
2. implement **(eliminate*)** rule;
3. split axioms $E = B \uplus R$ such that R constitutes a confluent and terminating term rewriting system, and introduce **(normalize)** rule to normalize terms (lazily) before applying any other rules except **(delete)** and **(eliminate*)**;
4. introduce a limit on a number of applications of **(mutate)** rule;
5. introduce a limit on a number of bindings that do not decrease problem size;
6. introduce a limit on total number of bindings.

When adapting ideas from classical E-unification and HOU, some technical difficulties arise from having binders lacking (in general) the nice syntactic properties of λ-calculus. These difficulties affect both the design of our unification procedure, leading to some simplifications, and the completeness proof, requiring us to find a different approach to define the measure and consider cases that do not have analogues.

In the procedure, we had to simplify whenever those ideas relied on normalisation, η-expansion, or specific syntax of λ-terms. Many HOU algorithms look at syntactic properties of terms to determine which rules to apply. In particular, HOU algorithms often distinguish *flex* and *rigid* terms [15, 23]. Jensen and Pietrzykowski introduce a notion of *ω-simple* terms [16]. Vukmirovic, Bentkamp, and Nummelin [34] introduce notions of *base-simple* and *solid* terms. These properties crucially depend on specific normalisation properties of λ-calculus, which might be inaccessible in an arbitrary second-order equational theory. Thus, our procedure contains more non-determinism than might be necessary.

One notable example of such simplication is in the imitation and projection bindings. In HOU algorithms, it is common to have substitutions of the form

$$[\mathrm{M} \mapsto \lambda x_1, \ldots, x_n.f\ (\mathrm{H}_1\ x_1\ \ldots\ x_n)\ \ldots\ (\mathrm{H}_k\ x_1\ \ldots\ x_n)]$$

where f can be a bound variable (one of $x_1, \ldots, x_n$) or a constant of type $\sigma_1 \Rightarrow \ldots \Rightarrow \sigma_k \Rightarrow \tau$. These are called Huet-style projection or imitation bindings [16, 34] or partial bindings [15, 23]. Huet-style projections (and conditions prompting their use) are non-trivial to generalise well to arbitrary second-order abstract syntax, so we skipped them in this paper, opting out for simpler rules but larger search space.

In the completeness proof for HOU algorithms, the syntactic properties of λ-calculus are heavily exploited. Their inaccessibility in a general second-order equational theory has contributed to some difficulties when developing the proof of completeness in Theorem 38. Perhaps, the most challenging of all was handling of the Item 2(e)iii of the proof which requires the assumption of no mixed operators and Lemma 44. These do not appear to have an analogue in completeness proofs for HOU or first-order E-unification.

7 Conclusion and Future Work

We have formulated the equational unification problem for second-order abstract syntax, allowing to reason naturally about unification of terms in languages with binding constructions. Such languages include, but are not limited to higher-order systems such as λ-calculus, which expands potential applications to more languages. We also presented a procedure to solve such problems and our main result shows completeness of this procedure.

In future work, we will focus on optimisations and recognition of decidable fragments of E-unification over second-order equations.

One notable optimisation is splitting E into two sets $R \uplus B$, where R is a set of directed equations, forming a confluent second-order term rewriting system, and B is a set of undirected equations (such as associativity and commutativity axioms).

Another potential optimisation stems from a generalisation of Huet-style binding (also known as *partial binding*), which can lead to more informed decisions on which rule to apply in the procedure, introduce Huet-style version of **(project)** and improve **(iterate)** rule, significantly reducing the search space. A version of such an optimisation has been implemented in a form of a heuristic to combine **(imitate)** and **(project)** rules by Kudasov [20].

There are several well-studied fragments both for E-unification and higher-order unification. For example, unification in monoidal theories is essentially solving linear equations over semirings [26]. In higher-order unification, there are several well-known decidable fragments such as pattern unification [23]. Vukmirovic, Bentkamp, and Nummelin have identified some of the practically important decidable fragments as well as a new one in their recent work on efficient full higher-order unification [34]. It is interesting to see if these fragments could be generalised to second-order abstract syntax and used as oracles, possibly yielding an efficient E-unification for second-order abstract syntax as a strict generalisation of their procedure.

References

1 Franco Barbanera, Maribel Fernández, and Herman Geuvers. Modularity of strong normalization in the algebraic-λ-cube. *J. Funct. Program.*, 7(6):613–660, November 1997. doi:10.1017/S095679689700289X.

2 Adam Chlipala. Parametric higher-order abstract syntax for mechanized semantics. In *Proceedings of the 13th ACM SIGPLAN International Conference on Functional Programming*, ICFP '08, pages 143–156, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1411204.1411226.

3 Jacek Chrząszcz and Daria Walukiewicz-Chrząszcz. *Towards Rewriting in Coq*, pages 113–131. Springer-Verlag, Berlin, Heidelberg, 2007.

4 Jesper Cockx. Type Theory Unchained: Extending Agda with User-Defined Rewrite Rules. In Marc Bezem and Assia Mahboubi, editors, *25th International Conference on Types for Proofs and Programs (TYPES 2019)*, volume 175 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:27, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.TYPES.2019.2.

5 Jesper Cockx. 1001 representations of syntax with binding. https://jesper.sikanda.be/posts/1001-syntax-representations.html, November 2021. Accessed: 2023-01-21.

6 Jesper Cockx, Nicolas Tabareau, and Théo Winterhalter. The Taming of the Rew: A Type Theory with Computational Assumptions. *Proc. ACM Program. Lang.*, 5(POPL), January 2021. doi:10.1145/3434341.

7 Denis Cousineau and Gilles Dowek. Embedding Pure Type Systems in the Lambda-Pi-Calculus Modulo. In Simona Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4583 of *Lecture Notes in Computer Science*, pages 102–117. Springer, 2007. doi:10.1007/978-3-540-73228-0_9.

8 Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Higher Order Unification via Explicit Substitutions. *Inf. Comput.*, 157(1-2):183–235, 2000. doi:10.1006/inco.1999.2837.

9 Marcelo Fiore and Dmitrij Szamozvancev. Formal Metatheory of Second-Order Abstract Syntax. *Proc. ACM Program. Lang.*, 6(POPL), January 2022. doi:10.1145/3498715.

10 Marcelo P. Fiore and Chung-Kil Hur. Second-Order Equational Logic (Extended Abstract). In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2010. doi:10.1007/978-3-642-15205-4_26.

11 Jean H. Gallier and Wayne Snyder. Complete sets of transformations for general E-unification. *Theor. Comput. Sci.*, 67(2&3):203–260, 1989. doi:10.1016/0304-3975(89)90004-2.

12 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.

13 Adam Michael Gundry. *Type inference, Haskell and dependent types.* PhD thesis, University of Strathclyde, Glasgow, UK, 2013. URL: http://oleg.lib.strath.ac.uk/R/?func=dbin-jump-full&object_id=22728.

14 Michael Hoche and Peter Szabó. Essential unifiers. *J. Appl. Log.*, 4(1):1–25, 2006. doi:10.1016/j.jal.2004.12.001.

15 Gérard P. Huet. A unification algorithm for typed lambda-calculus. *Theor. Comput. Sci.*, 1(1):27–57, 1975. doi:10.1016/0304-3975(75)90011-0.

16 D. C. Jensen and Tomasz Pietrzykowski. Mechanizing ω-order type theory through unification. *Theor. Comput. Sci.*, 3(2):123–171, 1976. doi:10.1016/0304-3975(76)90021-9.

17 Claude Kirchner and Christophe Ringeissen. Higher-Order Equational Unification via Explicit Substitutions. In Michael Hanus, Jan Heering, and Karl Meinke, editors, *Algebraic and Logic Programming, 6th International Joint Conference, ALP '97 - HOA '97, Southampton, UK, Spetember 3-5, 1997, Proceedings*, volume 1298 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 1997. doi:10.1007/BFb0027003.

18 Edward Kmett. Rotten bananas. http://comonad.com/reader/2008/rotten-bananas/, March 2008. Accessed: 2023-01-21.

19 Edward Kmett. Bound. https://www.schoolofhaskell.com/user/edwardk/bound, December 2015. Accessed: 2023-01-21.

20 Nikolai Kudasov. Functional Pearl: Dependent type inference via free higher-order unification, 2022. arXiv:2204.05653.

21 Tomer Libal and Dale Miller. Functions-as-Constructors Higher-Order Unification. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, volume 52 of *LIPIcs*, pages 26:1–26:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.FSCD.2016.26.

22 Francesco Mazzoli and Andreas Abel. Type checking through unification. *CoRR*, abs/1609.09709, 2016. arXiv:1609.09709.

23 Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Log. Comput.*, 1(4):497–536, 1991. doi:10.1093/logcom/1.4.497.

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

25 Tobias Nipkow and Christian Prehofer. Higher-order rewriting and equational reasoning. In W. Bibel and P. Schmitt, editors, *Automated Deduction — A Basis for Applications. Volume I: Foundations*, volume 8 of *Applied Logic Series*, pages 399–430. Kluwer, 1998.

26 Werner Nutt. Unification in monoidal theories is solving linear equations over semirings. In *SciDok - Der Wissenschaftsserver der Universität des Saarlandes*, volume 92-01 of *Research report / Deutsches Forschungszentrum für Künstliche Intelligenz*. SE - DFKI Deutsches Forschungszentrum für Künstliche Intelligenz, 1992. doi:10.22028/D291-24991.

27 Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In Richard L. Wexelblat, editor, *Proceedings of the ACM SIGPLAN'88 Conference on Programming Language Design and Implementation (PLDI), Atlanta, Georgia, USA, June 22-24, 1988*, pages 199–208. ACM, 1988. doi:10.1145/53990.54010.

28 Wayne Snyder. Higher order E-unification. In Mark E. Stickel, editor, *10th International Conference on Automated Deduction, Kaiserslautern, FRG, July 24-27, 1990, Proceedings*, volume 449 of *Lecture Notes in Computer Science*, pages 573–587. Springer, 1990. doi:10.1007/3-540-52885-7_115.

29 Mark-Oliver Stehr. The open calculus of constructions (part I): an equational type theory with dependent types for programming, specification, and interactive theorem proving. *Fundam. Informaticae*, 68(1-2):131–174, 2005. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi68-1-2-05.

30 Mark-Oliver Stehr. The open calculus of constructions (part II): an equational type theory with dependent types for programming, specification, and interactive theorem proving. *Fundam. Informaticae*, 68(3):249–288, 2005. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi68-3-04.

31 Peter Szabó and Jörg H. Siekmann. E-Unification based on Generalized Embedding. *Math. Struct. Comput. Sci.*, 31(8):898–917, 2021. doi:10.1017/S0960129522000019.

32 Peter Szabó, Jörg H. Siekmann, and Michael Hoche. What Is Essential Unification? In Eugenio G. Omodeo and Alberto Policriti, editors, *Martin Davis on Computability, Computational Logic, and Mathematical Foundations*, volume 10 of *Outstanding Contributions to Logic*, pages 285–314. Springer, 2016. doi:10.1007/978-3-319-41842-1_11.

33 Val Tannen. Combining algebra and higher-order types. In *Proceedings of the Third Annual Symposium on Logic in Computer Science (LICS '88), Edinburgh, Scotland, UK, July 5-8, 1988*, pages 82–90. IEEE Computer Society, 1988. doi:10.1109/LICS.1988.5103.

34 Petar Vukmirovic, Alexander Bentkamp, and Visa Nummelin. Efficient Full Higher-Order Unification. *Log. Methods Comput. Sci.*, 17(4), 2021. doi:10.46298/lmcs-17(4:18)2021.

35 Daria Walukiewicz-Chrząszcz. Termination of rewriting in the calculus of constructions. *J. Funct. Program.*, 13(2):339–414, March 2003. doi:10.1017/S0956796802004641.

36 Geoffrey Washburn and Stephanie Weirich. Boxes go bananas: Encoding higher-order abstract syntax with parametric polymorphism. *J. Funct. Program.*, 18(1):87–140, 2008. doi:10.1017/S0956796807006557.

37 Beta Ziliani and Matthieu Sozeau. A unification algorithm for Coq featuring universe polymorphism and overloading. In Kathleen Fisher and John H. Reppy, editors, *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada, September 1-3, 2015*, pages 179–191. ACM, 2015. doi:10.1145/2784731.2784751.

Two Decreasing Measures for Simply Typed λ-Terms

Pablo Barenbaum ✉
ICC, Universidad de Buenos Aires, Argentina
Universidad Nacional de Quilmes (CONICET), Buenos Aires, Argentina

Cristian Sottile ✉
ICC, Universidad de Buenos Aires (CONICET), Argentina
Universidad Nacional de Quilmes, Buenos Aires, Argentina

Abstract

This paper defines two decreasing measures for terms of the simply typed λ-calculus, called the $\mathcal{W}$-measure and the $\mathcal{T}^{\mathbf{m}}$-measure. A decreasing measure is a function that maps each typable λ-term to an element of a well-founded ordering, in such a way that contracting any β-redex decreases the value of the function, entailing strong normalization. Both measures are defined constructively, relying on an auxiliary calculus, a non-erasing variant of the λ-calculus. In this system, dubbed the $\lambda^{\mathbf{m}}$-calculus, each β-step creates a "wrapper" containing a copy of the argument that cannot be erased and cannot interact with the context in any other way. Both measures rely crucially on the observation, known to Turing and Prawitz, that contracting a redex cannot create redexes of higher degree, where the degree of a redex is defined as the height of the type of its λ-abstraction. The $\mathcal{W}$-measure maps each λ-term to a natural number, and it is obtained by evaluating the term in the $\lambda^{\mathbf{m}}$-calculus and counting the number of remaining wrappers. The $\mathcal{T}^{\mathbf{m}}$-measure maps each λ-term to a structure of nested multisets, where the nesting depth is proportional to the maximum redex degree.

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

Keywords and phrases Lambda Calculus, Rewriting, Termination, Strong Normalization, Simple Types

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.11

Funding This work was partially supported by project grants PICT-2021-I-A-00090, PICT-2021-I-INVI-00602, PIP 11220200100368CO, PICT 2019-1272, PUNQ 2218/22, and PUNQ 2219/22.

Acknowledgements To Giulio Manzonetto for fruitful discussions that led to the development of this work. To Eduardo Bonelli and the anonymous reviewers for feedback on earlier versions of this paper. The second author would like to thank his advisors Alejandro Díaz-Caro and Pablo E. Martínez López.

1 Introduction

In this paper we revisit a fundamental question, that of **strong normalization** of the simply typed λ-calculus (STLC). We begin by recalling that a reduction relation is *weakly normalizing* (WN) if every term can be reduced to normal form in a finite number of steps, whereas it is *strongly normalizing* (SN) if there are no infinite reduction sequences ($a_1 \to a_2 \to a_3 \to \ldots$). Let us review three proof techniques for proving strong normalization of the STLC.

One of the better known ways to prove that the STLC is SN is through arguments based on **reducibility models**. The idea is to interpret each type A as a set $[\![A]\!]$ of strongly normalizing terms, and to prove that each term M of type A is an element of $[\![A]\!]$.

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 11; pp. 11:1–11:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Many variants of these ideas can be found in the literature, including Girard's reducibility candidates [17] and Tait's saturated sets [30]. These techniques provide relatively succint proofs and they generalize well to extensions of the STLC, *e.g.* to dependent type theory [6] or classical calculi [13]. On the other hand, the abstract nature of reducibility arguments does not provide a "tangible" insight on why a β-reduction step brings a term closer to normal form. More specifically, reducibility arguments do not construct explicit **decreasing measures**. By decreasing measure we mean a function "#" mapping each λ-term to a well-founded ordering $(X, >)$ such that $M \to_\beta N$ implies $\#(M) > \#(N)$.

Another way to prove strong normalization is based on **redex degrees**. A *redex* in the STLC is an applied abstraction, *i.e.* a term of the form $(\lambda x.\, M)\, N$. The *degree* of a redex is defined as the *height* of the type of its abstraction. A crucial observation, that can be attributed to an unpublished note of Turing (as reported by Gandy [15]; see also [4]), is that *contracting a redex cannot create a redex of higher or equal degree.* Recall that a redex S is *created* by the contraction of a redex R if S has no *ancestor* before R. Indeed, as shown by Lévy [22], in the λ-calculus, redexes can be created in exactly one of the three ways below:

$$\begin{array}{lrcl}
\mathbf{1} & (\underline{\lambda}x.\,x)\,(\lambda y.\,M)\,N & \to_\beta & (\underline{\lambda}y.\,M)\,N \\
\mathbf{2} & (\underline{\lambda}x.\,\lambda y.\,M)\,N\,P & \to_\beta & (\underline{\lambda}y.\,M[x := N])\,P \\
\mathbf{3} & (\underline{\lambda}x.\,\ldots x\,M \ldots)\,(\lambda y.\,N) & \to_\beta & \ldots(\underline{\lambda}y.\,N)\,M[x := \lambda y.\,N]\ldots
\end{array}$$

where we underline the λ of the contracted redex on the left, and the λ of the created redex on the right. In each of these cases, it can be seen that the degree of the created redex is strictly lower than the degree of the contracted redex. For instance, in creation case **1**, the type of the contracted redex is of the form $(A \to B) \to (A \to B)$, while the type of the created redex is $A \to B$, so the height strictly decreases.

With this fact in mind, for each term M one can define what we call **Turing's measure**, *i.e.* the multiset $\mathcal{T}(M)$ of the degrees of all the redexes of M. One may hope that any reduction step $M \to_\beta N$ decreases the measure, *i.e.* $\mathcal{T}(M) \succ \mathcal{T}(N)$, where "$\succ$" is the usual well-founded multiset ordering induced by the ordering $(\mathbb{N}, >)$ of its elements [12]. Unfortunately, this is not the case: even though contracting a redex can only create redexes of strictly lower degree, it can still make an arbitrary number of *copies* of redexes of arbitrarily large degrees.

In his notes, Turing observed that one can follow a reduction strategy that always selects the *rightmost* redex of highest degree. This strategy ensures that the contracted redex does not copy redexes of higher or equal degree, which makes the $\mathcal{T}(-)$ measure strictly decrease, thus proving that the λ-calculus is WN. An even simpler measure that also decreases, if one follows this strategy, is $\mathcal{T}'(M) = (D, n)$, where D is the maximum degree of the redexes in M and n is the number of redexes of degree D in M. Similar ideas were exploited by Prawitz [28] and Gentzen (as reported by von Plato [27]) to normalize proofs in natural deduction. After WN has been established, an indirect proof of SN can be obtained by translating each typable λ-term M to a typable term M' of the *λI-calculus*; see for instance [29, Section 3.5].

In summary, redex degrees can be used to define concrete measures such as $\mathcal{T}(M)$ and $\mathcal{T}'(M)$, that are computable in linear time and decrease when following a particular reduction strategy. As already mentioned, these measures do not necessarily decrease when contracting arbitrary β-redexes.

A third way to prove SN relies on an interpretation that maps terms to **increasing functionals**. This approach was pioneered by Gandy [16] and refined by de Vrijer [10]. Each type A is mapped to a partially ordered set $[\![A]\!]$. Specifically, base types are mapped to $(\mathbb{N}, \leq)$, and $[\![A \to B]\!]$ is defined as the set of strictly increasing functions $[\![A]\!] \to [\![B]\!]$, partially ordered by the point-wise order. Each term M of type A is interpreted as an element $[M] \in [\![A]\!]$. Moreover, an element $f \in [\![A]\!]$ can be projected to a natural number $f\star \in \mathbb{N}$ in such a way that $M \to_\beta N$ implies $[M]\star > [N]\star$. This indeed provides a decreasing measure. One of the downsides of this measure is that computing $[M]\star$ is essentially as difficult as evaluating M, because $[M]$ is defined as a higher-order functional with a similar structure as the λ-term M itself.

In this work we propose **two decreasing measures for the STLC**, that we dub the $\mathcal{W}$-measure and the $\mathcal{T}^{\mathbf{m}}$-measure, and we prove that they are decreasing. An ideal decreasing measure should fulfill multiple (partly subjective) requirements: **1.** the measure should be easy to calculate, in terms of computational complexity; **2.** its codomain (a well-founded ordering) should be simple, in terms of its ordinal type; **3.** it should give us insight on why β-reduction terminates; **4.** it should be easy to prove that the measure is decreasing. A measure that excels simultaneously at all these requirements is elusive, and perhaps unattainable. The proposed measures have different strengths and weaknesses.

Contributions and structure of this document. The $\mathcal{W}$-measure and the $\mathcal{T}^{\mathbf{m}}$-measure are defined by means of on an auxiliary calculus that we dub the $\lambda^{\mathbf{m}}$-calculus. The remainder of the paper is structured as follows.

In **Section 2** we **define the $\lambda^{\mathbf{m}}$-calculus**. It is an extension of the STLC with terms[1] of the form $t\{s\}$, called *wrappers*. A wrapper $t\{s\}$ should be understood as essentially the term t in which s is a *memorized term*, that is, leftover garbage that can be reduced but cannot interact with the context in any way. The type of $t\{s\}$ is the same as the type of t, disregarding the type of s.

The β-reduction rule is modified so that contracting a redex $(\lambda x.\, t)\, s$, besides substituting the free occurrences of x by s in t, produces a wrapper that contains a copy of the argument s. The reduction rule is $(\lambda x.\, t)\{u_1\}\ldots\{u_n\}\, s \;\to_{\mathbf{m}}\; t[x := s]\{s\}\{u_1\}\ldots\{u_n\}$. Note that we allow the presence of an arbitrary number of memorized terms mediating between the abstraction and the application. This is to avoid memorized terms *blocking* redexes. For example, if $I = \lambda x.\, x$:

$$(\lambda x.\, x(xy))I \to_{\mathbf{m}} (I(Iy))\{I\} \to_{\mathbf{m}} (Iy)\{Iy\}\{I\} \to_{\mathbf{m}} (Iy)\{y\{y\}\}\{I\} \to_{\mathbf{m}} y\{y\}\{y\{y\}\}\{I\}$$

Then we study some syntactic properties of $\lambda^{\mathbf{m}}$. In particular, we define a relation $t \rhd s$ of *forgetful reduction*, meaning that s is obtained from t by erasing one memorized subterm. For example, $x\,\{x\{y\}\}\{y\{z\}\} \rhd x\,\{y\{z\}\}$. Forgetful reduction is used as a technical tool to prove that the measures are decreasing in the following sections.

In **Section 3**, we **propose the $\mathcal{W}$-measure** (Def. 12), and we prove that it is decreasing. To define the $\mathcal{W}$-measure, we resort to an operation $\mathtt{S}_d(t)$ that simultaneously contracts all the redexes of degree d in a term of the $\lambda^{\mathbf{m}}$-calculus, that is, the result of the *complete development* of all the redexes of degree d. The degree of a redex $(\lambda x.\, t)\{u_1\}\ldots\{u_n\}\, s$ is defined similarly as for the STLC, as the height of the type of the abstraction. To calculate the $\mathcal{W}$-measure of a λ-term M, let D be the maximum degree of the redexes in M, and define $\mathcal{W}(M)$ as the number of wrappers in $\mathtt{S}_1(\mathtt{S}_2(\ldots\mathtt{S}_D(M)))$. For example, if $M = (\lambda x.\, x\,(x\,y))\,(\lambda z.\, w)$, it turns out that $\mathtt{S}_1(\mathtt{S}_2(M)) = w\{w\{y\}\}\{\lambda z.\, w\}$ which has three

[1] Note that terms of the $\lambda^{\mathbf{m}}$-calculus are ranged over by $t, s, \ldots$ (rather than $M, N, \ldots$).

wrappers, so $\mathcal{W}(M) = 3$. The $\mathcal{W}$-measure maps each typable λ-term to a natural number. The main result of Section 3 is Thm. 15, stating that $\mathcal{W}$ **is decreasing**, *i.e.* that $M \to_\beta N$ implies $\mathcal{W}(M) > \mathcal{W}(N)$.

In **Section 4** we study **reduction by degrees**, a restricted notion of reduction in the $\lambda^{\mathbf{m}}$-calculus, written $t \xrightarrow{d}_{\mathbf{m}} s$, meaning that t reduces to s by contracting a redex of degree d. This section contains technical commutation, termination, and postponement results.

In **Section 5**, we **propose the $\mathcal{T}^{\mathbf{m}}$-measure**, and we prove that it is decreasing. To define the $\mathcal{T}^{\mathbf{m}}$-measure, we define two auxiliary measures $\mathcal{T}^{\mathbf{m}}_{\leq D}(t)$ and $\mathcal{R}^{\mathbf{m}}_{D}(t)$, indexed by a natural number $D \in \mathbb{N}_0$, mutually recursively:

- $\mathcal{T}^{\mathbf{m}}_{\leq D}(t)$ is the multiset of pairs $(d, \mathcal{R}^{\mathbf{m}}_{d}(t))$, for each redex occurrence of degree $d \leq D$ in t;
- $\mathcal{R}^{\mathbf{m}}_{D}(t)$ is the multiset of elements $\mathcal{T}^{\mathbf{m}}_{\leq D-1}(t')$, for each reduction sequence $t \xrightarrow{D}{}^{*}_{\mathbf{m}} t'$.

The measure $\mathcal{T}^{\mathbf{m}}_{\leq D}(t)$ is defined for every $D \geq 0$, while $\mathcal{R}^{\mathbf{m}}_{D}(t)$ is defined only for $D \geq 1$. Multisets are ordered according to the usual multiset ordering, and pairs according to the lexicographic ordering. To calculate the $\mathcal{T}^{\mathbf{m}}$-measure of a λ-term M, let D be the maximum degree of the redexes in M, and define $\mathcal{T}^{\mathbf{m}}_{\leq}(M) \overset{\text{def}}{=} \mathcal{T}^{\mathbf{m}}_{\leq D}(M)$. The measure $\mathcal{T}^{\mathbf{m}}_{\leq}(M)$ yields a structure of nested multisets of nesting depth at most $2D$. The main theorem of Section 3 is Thm. 32, stating that $\mathcal{T}^{\mathbf{m}}$ **is decreasing**, *i.e.* that $M \to_\beta N$ implies $\mathcal{T}^{\mathbf{m}}_{\leq}(M) > \mathcal{T}^{\mathbf{m}}_{\leq}(N)$.

Finally, in **Section 6**, we conclude.

2 The $\lambda^{\mathbf{m}}$-calculus

As mentioned in the introduction, the $\lambda^{\mathbf{m}}$-calculus is an extension of the STLC in which the β-reduction rule keeps an extra memorized copy of the argument in a “wrapper” $t\{\!\!\{s\}\!\!\}$, in such a way that contracting a redex like $(\lambda x.\, t)\, s$ does not erase s, even if x does not occur free in t. In this section we define the $\lambda^{\mathbf{m}}$-calculus and we prove some of the properties that are needed in the following sections to prove that the $\mathcal{W}$-measure and the $\mathcal{T}^{\mathbf{m}}$-measure are decreasing. In particular, we discuss *subject reduction* (Prop. 3) and *confluence* (Prop. 4); we define an operation of **simplification** (Def. 5) which turns out to calculate the normal form of a term (Prop. 7); and we define the relation called **forgetful reduction** (Def. 8), which is shown to commute with reduction (Prop. 10).

First we fix the notation and nomenclature. *Types* of the STLC are either base types $(\alpha, \beta, \ldots)$ or arrow types $(A \to B)$. *Terms* are either variables $(x^A, y^A, \ldots)$, abstractions $(\lambda x^A.\, M)$, or applications $(M\, N)$, with the usual typing rules. Terms are defined up to α-renaming of bound variables. We adopt an *à la* Church presentation of the STLC, but we omit most type decorations on variables as long as there is little danger of confusion. The β-reduction rule is $(\lambda x.\, M)\, N \to_\beta M[x := N]$ where $M[x := N]$ is the capture-avoiding substitution of the free occurrences of x in M by N.

The $\lambda^{\mathbf{m}}$-calculus: syntax and reduction. The set of $\lambda^{\mathbf{m}}$-*terms* – or just *terms* – is given by $t, s, \ldots ::= x^A \mid \lambda x^A.\, t \mid t\, s \mid t\{\!\!\{s\}\!\!\}$. The four kinds of terms are respectively called *variables*, *abstractions*, *applications*, and *wrappers*. In a wrapper $t\{\!\!\{s\}\!\!\}$, the subterm t is called the *body* and s is called the *memorized term*. As in the STLC, we usually omit type annotations and terms are regarded up to α-renaming. A *context* is a term $\mathtt{C}$ with a single free occurrence of a distinguished variable $\Box$, and $\mathtt{C}[t]$ is the variable-capturing substitution of the occurrence of $\Box$ in $\mathtt{C}$ by t.

Typing judgments are of the form $\Gamma \vdash t : A$ where Γ is a partial function mapping variables to types. Derivable typing judgments are defined by the following rules:

$$\frac{}{\Gamma, x : A \vdash x^A : A} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x^A.t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash s : A}{\Gamma \vdash t\, s : B} \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash s : B}{\Gamma \vdash t\{s\} : A}$$

A term t is *typable* if $\Gamma \vdash t : A$ holds for some Γ and some A. Unless otherwise specified, when we speak of "terms" we mean "typable terms". It is straightforward to show that a typable term has a unique type. We write $\mathsf{type}(t)$ for the type of t.

A *memory*, written $\mathtt{L}$, is a list of memorized terms, given by the grammar $\mathtt{L} ::= \Box \mid \mathtt{L}\{t\}$. If t is a term and $\mathtt{L}$ is a memory, we write $t\mathtt{L}$ for the term that results from appending all the memorized terms in $\mathtt{L}$ to t, that is, $(t)(\Box\{s_1\}\ldots\{s_n\}) = t\{s_1\}\ldots\{s_n\}$. We write $t[x := s]$ for the operation of capture-avoiding substitution of the free occurrences of x in t by s. The $\lambda^{\mathbf{m}}$-calculus is the rewriting system whose objects are typable $\lambda^{\mathbf{m}}$-terms, endowed with the following notion of reduction, closed by compatibility under arbitrary contexts:

▶ **Definition 1** (Reduction in the $\lambda^{\mathbf{m}}$-calculus). $(\lambda x.\, t)\mathtt{L}\, s \;\to_{\mathbf{m}}\; t[x := s]\{s\}\mathtt{L}$

Abstractions followed by a memory, *i.e.* terms of the form $(\lambda x.\, t)\mathtt{L}$, are called $\mathbf{m}$-*abstractions*. Note that all abstractions are also $\mathbf{m}$-abstractions, as $\mathtt{L}$ may be empty. A *redex* is an expression matching the left-hand side of the $\to_{\mathbf{m}}$-reduction rule, which must be an *applied* $\mathbf{m}$-*abstraction*, *i.e.* a term of the form $(\lambda x.\, t)\mathtt{L}\, s$. The *height* of a type is given by $\mathsf{h}(\alpha) \overset{\text{def}}{=} 0$ and $\mathsf{h}(A \to B) \overset{\text{def}}{=} 1 + \max(\mathsf{h}(A), \mathsf{h}(B))$. The *degree* of a $\mathbf{m}$-abstraction $(\lambda x.\, t)\mathtt{L}$ is defined as the height of its type; note that this number is always strictly positive, since the type must be of the form $A \to B$. Moreover, this type is unique, so the operation is well-defined. The *degree* of a redex $(\lambda x.\, t)\mathtt{L}\, s$ is defined as the degree of the $\mathbf{m}$-abstraction $(\lambda x.\, t)\mathtt{L}$. The *max-degree* of a term t is written $\mathsf{maxdeg}(t)$ and it is defined as the maximum degree of the redexes in t, or 0 if t has no redexes. The *weight* $\mathsf{w}(t)$ of a $\lambda^{\mathbf{m}}$-term t is the number of wrappers in t.

▶ **Example 2.** Let 0 be a base type and let $t := (\lambda x^{0\to 0}.\, \lambda y^0.\, y^0\{x^{0\to 0}\,(x^{0\to 0}\, z^0)\})\, I\, w^0$, where $I := \lambda x^0.\, x^0$. One possible way to reduce t is:

$$\begin{aligned} (\lambda x.\, \lambda y.\, y\{x\,(x\, z)\})\, I\, w \;&\to_{\mathbf{m}}\; (\lambda y.\, y\{I\,(I\, z)\})\{I\}\, w \;\to_{\mathbf{m}}\; w\{I\,(I\, z)\}\{w\}\{I\} \\ &\to_{\mathbf{m}}\; w\{I\,(z\{z\})\}\,\{w\}\,\{I\} \;\to_{\mathbf{m}}\; w\{z\{z\}\{z\{z\}\}\}\{w\}\{I\} = s \end{aligned}$$

The degrees of the redexes contracted in each step are 2, 1, 1, and 1, in that order. Note that $\mathsf{maxdeg}(t) = 2$ and that the weight of the resulting term is $\mathsf{w}(s) = 6$.

Two basic properties of the $\lambda^{\mathbf{m}}$-calculus are subject reduction and confluence. These are immediate consequences of the fact that the $\lambda^{\mathbf{m}}$-calculus can be understood as an *orthogonal HRS* in the sense of Nipkow [26], *i.e.* a left-linear higher-order rewriting system without critical pairs.

▶ **Proposition 3** (Subject reduction). *Let $\Gamma \vdash t : A$ and $t \to_{\mathbf{m}} s$. Then $\Gamma \vdash s : A$.*

▶ **Proposition 4** (Confluence). *If $t_1 \to_{\mathbf{m}}^* t_2$ and $t_1 \to_{\mathbf{m}}^* t_3$, there exists a term t_4 such that $t_2 \to_{\mathbf{m}}^* t_4$ and $t_3 \to_{\mathbf{m}}^* t_4$.*

Full simplification. Next, we define an operation written $\mathtt{S}_*(t)$ and called *full simplification*.

Let $d \geq 1$ be a natural number. The *simplification of degree d*, written $\mathtt{S}_d(t)$, is the result of simultaneously contracting all the redexes of degree d in t, that is, the result of the *complete development* of all redexes of degree d. Formally, for each $\lambda^{\mathbf{m}}$-term t we define $\mathtt{S}_d(t)$, and, for each memory $\mathtt{L}$, we define $\mathtt{S}_d(\mathtt{L})$ as follows:

▶ **Definition 5** (Simplification).

$$
\begin{aligned}
\mathtt{S}_d(x) &\overset{\text{def}}{=} x \\
\mathtt{S}_d(\lambda x.\, t) &\overset{\text{def}}{=} \lambda x.\, \mathtt{S}_d(t) \\
\mathtt{S}_d(t\, s) &\overset{\text{def}}{=} \begin{cases} \mathtt{S}_d(t')[x := \mathtt{S}_d(s)]\{\mathtt{S}_d(s)\}\mathtt{S}_d(\mathtt{L}) & \text{if } t = (\lambda x.\, t')\mathtt{L} \text{ and it is of degree } d \\ \mathtt{S}_d(t)\, \mathtt{S}_d(s) & \text{otherwise} \end{cases} \\
\mathtt{S}_d(t\{s\}) &\overset{\text{def}}{=} \mathtt{S}_d(t)\{\mathtt{S}_d(s)\}
\end{aligned}
$$

where if $\mathtt{L}$ is a memory, $\mathtt{S}_d(\mathtt{L})$ is defined by $\mathtt{S}_d(\Box) \overset{\text{def}}{=} \Box$ and $\mathtt{S}_d(\mathtt{L}\{t\}) \overset{\text{def}}{=} \mathtt{S}_d(\mathtt{L})\{\mathtt{S}_d(t)\}$. Furthermore, if t is a $\lambda^{\mathbf{m}}$-term of max-degree D, we define the *full simplification* of t as the term that results from iteratively taking the simplification of degree i from D down to 1. More precisely, $\mathtt{S}_*(t) \overset{\text{def}}{=} \mathtt{S}_1(\ldots \mathtt{S}_{D-1}(\mathtt{S}_D(t)))$.

▶ **Example 6.** Consider the λ-term $M = (\lambda x^{0\to 0}.\, x^{0\to 0}(x^{0\to 0}\, y^0))(\lambda z^0.\, w^0)$. It can be regarded also as a $\lambda^{\mathbf{m}}$-term, and we have:

$$
\begin{aligned}
\mathtt{S}_2(M) &= ((\lambda z^0.\, w^0)\, ((\lambda z^0.\, w^0)\, y^0))\{\lambda z^0.\, w^0\} \\
\mathtt{S}_*(M) = \mathtt{S}_1(\mathtt{S}_2(M)) &= w^0\{w^0\{y^0\}\}\{\lambda z^0.\, w^0\}
\end{aligned}
$$

Note that M has only one redex, whose abstraction is of type $(0 \to 0) \to 0$ and hence of degree 2, and that $\mathtt{S}_2(M)$ has two redexes, whose abstractions are of type $0 \to 0$ and hence of degree 1. Moreover, consider the λ-term $N = (\lambda z^0.\, w^0)\, ((\lambda z^0.\, w^0)\, y^0)$. Then $\mathtt{S}_*(N) = \mathtt{S}_1(N) = w\{w\{y\}\}$. Note that N has two redexes whose abstraction is of type $0 \to 0$ and hence of degree 1. As an additional note, in the λ-calculus there is a reduction step $M \to_\beta N$, and we have that $\mathsf{w}(\mathtt{S}_*(M)) = 3 > 2 = \mathsf{w}(\mathtt{S}_*(N))$. So this example illustrates that the $\mathcal{W}$-measure (as defined in Def. 12) is decreasing (as we will show in Thm. 15).

As it turns out, **full simplification corresponds to reduction to normal form**. More precisely, we have the following result, which entails in particular that the $\lambda^{\mathbf{m}}$-calculus is weakly normalizing:

▶ **Proposition 7.** *$t \to_{\mathbf{m}}^* \mathtt{S}_*(t)$, and moreover $\mathtt{S}_*(t)$ is a $\to_{\mathbf{m}}$-normal form.*

Proof. To show that $t \to_{\mathbf{m}}^* \mathtt{S}_*(t)$, it suffices to prove a lemma stating that $t \to_{\mathbf{m}}^* \mathtt{S}_d(t)$ for all $d \geq 1$. This implies that $t \to_{\mathbf{m}}^* \mathtt{S}_D(t) \to_{\mathbf{m}}^* \mathtt{S}_{D-1}(\mathtt{S}_D(t)) \ldots \to_{\mathbf{m}}^* \mathtt{S}_1(\ldots \mathtt{S}_{D-1}(\mathtt{S}_D(t))) = \mathtt{S}_*(t)$, where D is the max-degree of t. The lemma itself is straightforward by induction on t.

To show that $\mathtt{S}_*(t)$ is a $\to_{\mathbf{m}}$-normal form, the key property is that, after performing a simplification of order d, no redexes of order d remain. The reason is that contracting a redex of order d can only create redexes of lower degree. More precisely, we prove a key lemma stating that if $d \geq 1$ and $\mathsf{maxdeg}(t) \leq d$, then $\mathsf{maxdeg}(\mathtt{S}_d(t)) < d$. If we let $\mathsf{maxdeg}(t) \leq D$, we can iterate this lemma, to obtain that $\mathsf{maxdeg}(\mathtt{S}_D(t)) < D$, and $\mathsf{maxdeg}(\mathtt{S}_{D-1}(\mathtt{S}_D(t))) < D - 1$, …, and finally $\mathsf{maxdeg}(\mathtt{S}_1(\ldots \mathtt{S}_{D-1}(\mathtt{S}_D(t)))) < 1$. This means that $\mathtt{S}_*(t) = \mathtt{S}_1(\ldots \mathtt{S}_{D-1}(\mathtt{S}_D(t)))$ does not contain redexes, since there are no redexes of degree 0, so $\mathtt{S}_*(t)$ must be a $\to_{\mathbf{m}}$-normal form. ◀

Forgetful reduction. To conclude this section, we introduce the relation of *forgetful reduction* $t \rhd^+ s$, and we prove that it commutes with reduction.

▶ **Definition 8.** *A $\lambda^{\mathbf{m}}$-term t* reduces via a forgetful step *to s, written $t \rhd s$, according to the following axiom, closed by compatibility under arbitrary contexts:*

$$t\{s\} \rhd t$$

We say that t reduces via forgetful reduction *to s if and only if $t \rhd^+ s$, where $\rhd^+$ denotes the transitive closure of $\rhd$.*

▶ **Example 9.** $(\lambda x. x\{y\{y\}\})\{z\{z\}\} \rhd (\lambda x. x\{y\{y\}\})\{z\} \rhd (\lambda x. x)\{z\} \rhd \lambda x. x$.

▶ **Proposition 10** (Forgetful reduction commutes with reduction)**.** *If $t \rhd^+ s$ and $t \to_{\mathbf{m}}^* t'$, there exists a term s' such that $t' \rhd^+ s'$ and $s \to_{\mathbf{m}}^* s'$. Furthermore, if $t \rhd^+ s$ and t is a $\to_{\mathbf{m}}$-normal form, then s is also a normal form.*

Proof. The result can be reduced to a local commutation result, stating that if $t \rhd s$ and $t \to_{\mathbf{m}} t'$, there exists a term s' such that $t' \rhd^+ s'$ and $s \to_{\mathbf{m}}^= s'$, where $\to_{\mathbf{m}}^=$ is the reflexive closure of $\to_{\mathbf{m}}$. Local commutation can be proved by case analysis. The interesting cases are when a shrinking step $s \rhd s'$ lies inside the argument of a redex, and when a reduction step $r \to_{\mathbf{m}} r'$ is inside erased garbage:

$$\begin{tikzcd}
(\lambda x. t)\mathtt{L}\, s \arrow[r, phantom, "\rhd"] \arrow[d] & (\lambda x. t)\mathtt{L}\, s' \arrow[d] \\
t[x := s]\{s\}\mathtt{L} \arrow[r, phantom, "\rhd^+"] & t[x := s']\{s'\}\mathtt{L}
\end{tikzcd}
\qquad
\begin{tikzcd}
u\{r\} \arrow[r, phantom, "\rhd"] \arrow[d] & u \arrow[d, equal] \\
u\{r'\} \arrow[r, phantom, "\rhd^+"] & u
\end{tikzcd}$$

For the last part of the statement, it suffices to show that if $t \rhd s$ in one step and t is a $\to_{\mathbf{m}}$-normal form, then s is also a normal form, which is straightforward by induction on t. ◀

Each step in the STLC has a *corresponding* step in the $\lambda^{\mathbf{m}}$-calculus, that contracts the redex in the same position. For instance the step $(\lambda x. x\, y)\, I \to_\beta I\, y$ in the STLC has a corresponding step $(\lambda x. x\, y)\, I \to_{\mathbf{m}} (I\, y)\{I\}$ in the $\lambda^{\mathbf{m}}$-calculus. In this example, $(I\, y)\{I\} \rhd I\, y$. The following easy lemma confirms that this is a general fact:

▶ **Lemma 11** (Reduce/forget lemma)**.** *Let $M \to_\beta N$ be a β-step, and let $M \to_{\mathbf{m}} s$ be the corresponding step in $\lambda^{\mathbf{m}}$. Then $s \rhd N$.*

3 The $\mathcal{W}$-measure

In this section, we **define the $\mathcal{W}$-measure** (Def. 12) and we **prove that it is decreasing** (Thm. 15). Let us try to convey some ideas that led to the definition of the $\mathcal{W}$-measure. Recall that an abstract rewriting system $(A, \to)$ is *weakly Church–Rosser* (WCR) if $\leftarrow\to \subseteq \to^*\leftarrow^*$, *Church–Rosser* (CR) if $\leftarrow^*\to^* \subseteq \to^*\leftarrow^*$, and *increasing* (Inc) if there exists a function $|\cdot| : A \to \mathbb{N}$ such that $a \to b$ implies $|a| < |b|$. Let us also recall *Klop–Nederpelt's lemma* [31, Theorem 1.2.3 (iii)], which states that $\text{Inc} \wedge \text{WCR} \wedge \text{WN} \implies \text{SN} \wedge \text{CR}$.

Let $(A, \to)$ be increasing and WCR. Given a reduction $a \to^* b$, where b is a normal form, we can find a *decreasing* measure for the set of objects reachable from a, that is, the set $\{c \in A \mid a \to^* c\}$. In fact, by Klop–Nederpelt's lemma, we know that for every $c \in A$ such that $a \to^* c$ we have that $c \to^* b$, which implies that $|c| \leq |b|$, and hence we can define $\#(c) := |b| - |c|$. It is easy to see that $\#(-)$ is a decreasing measure, since $c \to c'$ implies that $|c| < |c'|$ so $\#(c) := |b| - |c| > |b| - |c'| = \#(c')$. Furthermore, the value of $\#(c)$ does not depend on the choice of a, by uniqueness of normal forms.

The idea behind the $\mathcal{W}$-measure is that the construction of a *decreasing* measure can be based on an *increasing* measure, according to the previous observation. It is not possible to build an increasing measure directly for the STLC; *e.g.* the following infinite sequence of expansions $t \leftarrow I\, t \leftarrow I\, (I\, t) \leftarrow \ldots$ would induce an infinite decreasing chain of natural numbers $|t| > |I\, t| > |I\, (I\, t)| > \ldots$.

One could try to define an increasing measure in a variant of the STLC such as Endrullis *et al.*'s clocked λ-calculus [14], in which the β-rule becomes $(\lambda x.\, t)\, s \to \tau(t[x := s])$, that is, contracting a β-redex produces a counter "τ" that keeps track of the number of contracted redexes. One could then count the number of τ's: for example, in the reduction sequence $(\lambda x.\, x\,(x\,y))\, I \to \tau(I\,(I\,y)) \to \tau\tau(I\,y) \to \tau\tau\tau y$ the number of counters strictly increases with each step. Unfortunately, this does not define an increasing measure, due to *erasure*. For example, $(\lambda x.\, y)\, t \to \tau y$ erases all the counters in t.

This is the motivation behind the definition of the $\lambda^{\mathbf{m}}$-calculus, which avoids erasure by always keeping an extra copy of the argument in a wrapper. The $\lambda^{\mathbf{m}}$-calculus is indeed increasing: in a step $t \to_{\mathbf{m}} s$ one has that $\mathsf{w}(t) < \mathsf{w}(s)$, where we recall that $\mathsf{w}(t)$ denotes the *weight*, *i.e.* the number of wrappers in t. For example, the step $(\lambda x.\, y)\,(z\{z\}) \to_{\mathbf{m}} y\{z\{z\}\}$ increases the number of wrappers. The decreasing measure $\mathcal{W}(M)$ is defined essentially by reducing M to normal form in the $\lambda^{\mathbf{m}}$-calculus and counting the number of wrappers in the result:

▶ **Definition 12** (The $\mathcal{W}$-measure)**.** *For each typable λ-term M, define* $\mathcal{W}(M) \overset{\text{def}}{=} \mathsf{w}(\mathtt{S}_*(M))$.

As we show below, $\mathtt{S}_*(M)$ turns out to be exactly the normal form of M in the $\lambda^{\mathbf{m}}$-calculus. We insist in writing $\mathtt{S}_*(M)$ to emphasize that the *definition* of the $\mathcal{W}$-measure does not require to prove that the $\lambda^{\mathbf{m}}$-calculus is weakly normalizing. Indeed, the simplification $\mathtt{S}_d(t)$ can be defined by structural induction on t, and the full simplification $\mathtt{S}_*(t) = \mathtt{S}_1(\mathtt{S}_2(\ldots \mathtt{S}_D(t)))$ can be calculated in exactly D iterations. On the other hand, the *proof* that the $\mathcal{W}$-measure is decreasing does rely on the fact that $\mathtt{S}_*(M)$ is the normal form of M.

In the remainder of this section, we prove that the $\mathcal{W}$-measure is indeed decreasing. The following lemma states that forgetful reduction decreases weight, and it is straightforward to prove:

▶ **Lemma 13.** *If $t \rhd^+ s$ then* $\mathsf{w}(t) > \mathsf{w}(s)$.

The proof that the $\mathcal{W}$-measure decreases relies on the two following properties that relate full simplification $\mathtt{S}_*(-)$ respectively with reduction ($\to_{\mathbf{m}}$) and forgetful reduction ($\rhd^+$):

▶ **Lemma 14.** 1. *If $t \to_{\mathbf{m}} s$ then $\mathtt{S}_*(t) = \mathtt{S}_*(s)$.* 2. *If $t \rhd^+ s$ then $\mathtt{S}_*(t) \rhd^+ \mathtt{S}_*(s)$.*

Proof. For the first item, note that by Prop. 7, we know that $t \to_{\mathbf{m}}^* \mathtt{S}_*(t)$ and that $t \to_{\mathbf{m}} s \to_{\mathbf{m}}^* \mathtt{S}_*(s)$, where moreover $\mathtt{S}_*(t)$ and $\mathtt{S}_*(s)$ are $\to_{\mathbf{m}}$-normal forms. By confluence (Prop. 4), this means that $\mathtt{S}_*(t) = \mathtt{S}_*(s)$.

For the second item, note that by Prop. 7, we know that $t \to_{\mathbf{m}}^* \mathtt{S}_*(t)$. Since we also know $t \rhd^+ s$ by hypothesis, and since forgetful reduction commutes with reduction (Prop. 10), there exists a term u such that $s \to_{\mathbf{m}}^* u$ and $\mathtt{S}_*(t) \rhd^+ u$. By Prop. 7 we know that $\mathtt{S}_*(t)$ is in normal form, so by Prop. 10 u must also be a normal form. On the other hand, by Prop. 7 we know that $s \to_{\mathbf{m}}^* \mathtt{S}_*(s)$, where $\mathtt{S}_*(s)$ must also be a normal form. In summary, we have that $s \to_{\mathbf{m}}^* u$ and $s \to_{\mathbf{m}}^* \mathtt{S}_*(s)$, where both u and $\mathtt{S}_*(s)$ are normal forms. By confluence (Prop. 4) $u = \mathtt{S}_*(s)$, and from this we obtain that $\mathtt{S}_*(t) \rhd^+ u = \mathtt{S}_*(s)$, as required. ◀

▶ **Theorem 15.** *Let M, N be typable λ-terms such that $M \to_\beta N$. Then $\mathcal{W}(M) > \mathcal{W}(N)$.*

Proof. Given the step $M \to_\beta N$, consider the corresponding step $M \to_{\mathbf{m}} s$, and note that $s \rhd^+ N$ by the reduce/forget lemma (Lem. 11). Since $M \to_{\mathbf{m}} s \rhd^+ N$, by Lem. 14, we have that $\mathtt{S}_*(M) = \mathtt{S}_*(s) \rhd^+ \mathtt{S}_*(N)$. Finally, by Lem. 13, $\mathcal{W}(M) = \mathsf{w}(\mathtt{S}_*(M)) > \mathsf{w}(\mathtt{S}_*(N)) = \mathcal{W}(N)$. ◀

The following is one example that the $\mathcal{W}$-measure decreases (see Ex. 6 for another example):

▶ **Example 16.** Let $M = (\lambda x^{0}.\, y^{0\to 0\to 0}\, x^{0}\, x^{0})\, ((\lambda x^{0\to 0}.\, x^{0\to 0}\, z^{0})\, f^{0\to 0})$, consider the step $M = (\lambda x.\, y\, x\, x)((\lambda x.\, x\, z)\, f) \to_{\beta} (\lambda x.\, y\, x\, x)\,(f\, z) = N$, and note that $\mathcal{W}(M) = \mathtt{w}(\mathtt{S}_{*}(M)) = 4 > 1 = \mathcal{W}(N)$, since:

$$\mathtt{S}_{*}(M) = (y\,(f\,z)\boldsymbol{\{}f\boldsymbol{\}}\,(f\,z)\boldsymbol{\{}f\boldsymbol{\}})\boldsymbol{\{}(f\,z)\boldsymbol{\{}f\boldsymbol{\}}\boldsymbol{\}} \qquad \mathtt{S}_{*}(N) = (y\,(f\,z)\,(f\,z))\boldsymbol{\{}f\,z\boldsymbol{\}}$$

4 Reduction by degrees

This section is of purely technical nature. The aim is to develop tools that we use in the following section to reason about the $\mathcal{T}^{\mathbf{m}}$-measure. To do so, we need to introduce witnesses of steps and reduction sequences, treating the $\lambda^{\mathbf{m}}$-calculus as an *abstract rewriting system* in the sense of [31, Def. 8.2.2] or as a *transition system* in the sense of [24, Def. 1]. *Objects* are $\lambda^{\mathbf{m}}$-terms, *steps* are 5-uples $R = (\mathtt{C}, x, t, \mathtt{L}, s)$ witnessing the reduction step $\mathtt{C}[(\lambda x.\, t)\mathtt{L}\, s] \to_{\mathbf{m}} \mathtt{C}[t[x := s]\boldsymbol{\{}s\boldsymbol{\}}\mathtt{L}]$ under a context $\mathtt{C}$, and *reductions* $(\rho, \sigma, \ldots)$ are sequences of composable steps. Similarly, *forgetful steps* are triples $R = (\mathtt{C}, t, s)$ witnessing the forgetful reduction $\mathtt{C}[t\boldsymbol{\{}s\boldsymbol{\}}] \triangleright \mathtt{C}[t]$, and *forgetful reductions* (also written $\rho, \sigma, \ldots$) are sequences of composable forgetful steps. We write ρ^{src} and ρ^{tgt} respectively for the source and target terms of ρ.

For each $d \in \mathbb{N}_0$, we define **reduction of degree** d as follows:

▶ **Definition 17.** *$t \xrightarrow{d}_{\mathbf{m}} s$ if and only if $t \to_{\mathbf{m}} s$ by contracting a redex of degree d.*

We write $R : t \xrightarrow{d}_{\mathbf{m}} s$ if R is a step witnessing a reduction step of degree d, and $\rho : t \xrightarrow{d}{}^{*}_{\mathbf{m}} s$ if ρ is a reduction witnessing a sequence of reduction steps of degree d.

The following results require to explicitly manipulate steps and reductions.

▶ **Proposition 18** (Commutation of reduction by degrees)**.** *For any two reductions $\rho : t_1 \xrightarrow{d}{}^{*}_{\mathbf{m}} t_2$ and $\sigma : t_1 \xrightarrow{D}{}^{*}_{\mathbf{m}} t_3$, there exists a term t_4 and one can construct reductions $\sigma/\rho : t_2 \xrightarrow{D}{}^{*}_{\mathbf{m}} t_4$ and $\rho/\sigma : t_3 \xrightarrow{d}{}^{*}_{\mathbf{m}} t_4$ such that, furthermore, if $d \neq D$, then* **1.** *ρ/σ contains at least as many steps as ρ; and* **2.** *ρ/σ determines ρ, that is, $\rho_1/\sigma = \rho_2/\sigma$ implies $\rho_1 = \rho_2$.*

Proof. This is reduced to the fact that the $\lambda^{\mathbf{m}}$-calculus can be understood as an orthogonal higher-order rewriting system in the sense of Nipkow [26]. Indeed, ρ/σ and σ/ρ can be taken to be the standard notion of projection based on residuals for orthogonal HRSs. Note that item **1.** holds because the $\lambda^{\mathbf{m}}$-calculus is non-erasing while item **2.** is a consequence of the *unique ancestor* property, *i.e.* each redex *descends* from at most one redex. ◀

▶ **Corollary 19** (Termination of reduction by degrees)**.** *The relation $\xrightarrow{d}_{\mathbf{m}}$ is strongly normalizing.*

Proof. This is a consequence of the fact that HRSs enjoy the Finite Developments property [31, Theorem 11.5.11], observing that reduction of degree d does not create redexes of degree d. Alternatively, it can be easily shown that $t \xrightarrow{d}{}^{*}_{\mathbf{m}} \mathtt{S}_{d}(t)$ and $\mathtt{S}_{d}(t)$ is in $\xrightarrow{d}_{\mathbf{m}}$-normal form, so $\xrightarrow{d}_{\mathbf{m}}$ is WN. Moreover, one can observe that $\xrightarrow{d}_{\mathbf{m}}$ is *uniformly normalizing* [19], given that there is no erasure, which entails that $\xrightarrow{d}_{\mathbf{m}}$ is SN. ◀

▶ **Proposition 20** (Lifting property for lower steps)**.** *Let $d < D$ and $t \xrightarrow{d}_{\mathbf{m}} s \xrightarrow{D}{}^{*}_{\mathbf{m}} s'$. Then there exist terms t', s'' such that $t \xrightarrow{D}{}^{*}_{\mathbf{m}} t'$ and $s' \xrightarrow{D}{}^{*}_{\mathbf{m}} s''$ and $t' \xrightarrow{d}{}^{+}_{\mathbf{m}} s''$.*

Proof. Note that $t \overset{D}{\rightarrow}{}^{*}_{\mathbf{m}} \mathtt{S}_D(t)$. By Prop. 18, there exists a term u such that $s \overset{D}{\rightarrow}{}^{*}_{\mathbf{m}} u$ and $\mathtt{S}_D(t) \overset{d}{\rightarrow}{}^{+}_{\mathbf{m}} u$. Again by Prop. 18, there exists s'' such that $u \overset{D}{\rightarrow}{}^{*}_{\mathbf{m}} s''$ and $s' \overset{D}{\rightarrow}{}^{*}_{\mathbf{m}} s''$. Moreover, $\mathtt{S}_D(t)$ is in $\overset{D}{\rightarrow}_{\mathbf{m}}$-normal form. Since $\mathtt{S}_D(t) \overset{d}{\rightarrow}{}^{*}_{\mathbf{m}} u$ with $d < D$ and reduction does not create redexes of higher degree, u is also in $\overset{D}{\rightarrow}_{\mathbf{m}}$-normal form, so $u = s''$, and we are done. ◀

▶ **Proposition 21** (Postponement of forgetful reduction)**.** *For any two reductions $\rho : t \rhd^{*} t'$ and $\sigma : t' \overset{d}{\rightarrow}{}^{*}_{\mathbf{m}} s'$, there exists a term s and reductions $\rho^{\frown}\sigma : s \rhd^{*} s'$ and $\sigma^{\frown}\rho : t \overset{d}{\rightarrow}{}^{*}_{\mathbf{m}} s$. Furthermore, $\sigma^{\frown}\rho$ determines σ, that is, $\sigma_1^{\frown}\rho = \sigma_2^{\frown}\rho$ implies $\sigma_1 = \sigma_2$.*

Proof. This can be reduced to an analysis of the critical pairs between the rewriting rules defining $\rhd^{-1}$ and $\rightarrow_{\mathbf{m}}$. Critical pairs are of the form $(\lambda x.\,t)\mathtt{L}_1\{s\}\mathtt{L}_2\,u \rhd (\lambda x.\,t)\mathtt{L}_1\mathtt{L}_2\,u \rightarrow_{\mathbf{m}} t[x := u]\{u\}\mathtt{L}_1\mathtt{L}_2$ and can be closed by $(\lambda x.\,t)\mathtt{L}_1\{s\}\mathtt{L}_2\,u \rightarrow_{\mathbf{m}} t[x := u]\{u\}\mathtt{L}_1\{s\}\mathtt{L}_2 \rhd t[x := u]\{u\}\mathtt{L}_1\mathtt{L}_2$. ◀

The following diagrams depict the statements of the three preceding propositions:

$$\begin{tikzcd}
t_1 \arrow[r, two heads, "d", "\rho"'] \arrow[d, "\sigma"', "D"] \arrow[dr, phantom, "\text{Prop. 18}"] & t_2 \arrow[d, dotted, two heads, "\sigma/\rho"', "D"] \\
t_3 \arrow[r, dotted, two heads, "d", "\rho/\sigma"'] & t_4
\end{tikzcd}
\qquad
\begin{tikzcd}
t \arrow[rr, dotted, two heads, "D"] \arrow[d, two heads, "d"] & \text{Prop. 20} & t' \arrow[d, dotted, "d", "+"'] \\
s \arrow[r, two heads, "D"] & s' \arrow[r, dotted, two heads, "D"] & s''
\end{tikzcd}
\qquad
\begin{tikzcd}
t \arrow[r, phantom, "\rhd^{*}_{\rho}"] \arrow[d, dotted, two heads, "\sigma^{\frown}\rho"', "d"] & \text{Prop. 21} & t' \arrow[l, phantom] \arrow[d, "\sigma"', "d", "*"] \\
s \arrow[r, phantom, "\rhd^{*}_{\rho^{\frown}\sigma}"] & & s'
\end{tikzcd}$$

5 The $\mathcal{T}^{\mathrm{m}}$-measure

In this section, we **define the $\mathcal{T}^{\mathrm{m}}$-measure** (Def. 25) and we **prove that it is decreasing** (Thm. 32). We start with some preliminary notions.

A partially ordered set $(X, >)$ is *well-founded* if there are no infinite decreasing chains. $\mathbb{M}(X)$ denotes the set of *finite multisets* over a set X, which are functions $\mathfrak{m} : X \to \mathbb{N}_0$ such that $\mathfrak{m}(x) > 0$ for finitely many values of $x \in X$. We write $\mathfrak{m} + \mathfrak{n}$ for the sum of multisets, and $x \in \mathfrak{m}$ if $\mathfrak{m}(x) > 0$. We write $[x_1, \ldots, x_n]$ for the multiset of elements $x_1, \ldots, x_n$, taking multiplicities into account. If X is a finite set and $f : X \to Y$ is a function, we use the "multiset builder" notation $[f(x) \mid\mid x \in X]$ to denote the multiset $\sum_{x \in X}[f(x)]$. If $(X, >)$ is a partially ordered set, we define a binary relation $\succ^{1}$ on multisets by declaring that $\mathfrak{m} + [x] \succ^{1} \mathfrak{m} + \mathfrak{n}$ if $x > y$ for every $y \in \mathfrak{n}$. The *multiset order* induced by $(X, >)$ is the strict order relation on multisets defined by declaring that $\mathfrak{m} \succ \mathfrak{n}$ if and only if $\mathfrak{m}\ (\succ^{1})^{+}\ \mathfrak{n}$. We recall the following widely known theorem by Dershowitz and Manna [12]:

▶ **Theorem 22.** *If $(X, >)$ is well-founded, then $(\mathbb{M}(X), \succ)$ is well-founded.*

As usual, $\mathfrak{m} \succeq \mathfrak{n}$ stands for $(\mathfrak{m} = \mathfrak{n} \vee \mathfrak{m} \succ \mathfrak{n})$, and $\mathfrak{m} \preceq \mathfrak{n}$ stands for $\mathfrak{n} \succeq \mathfrak{m}$. We define an operation $k \otimes \mathfrak{m}$ by the recursive equations $0 \otimes \mathfrak{m} \overset{\text{def}}{=} []$ and $(1 + k) \otimes \mathfrak{m} \overset{\text{def}}{=} \mathfrak{m} + k \otimes \mathfrak{m}$. The relation $\mathfrak{m} :\succ: \mathfrak{n}$, called the *pointwise multiset order*, is defined to hold if $\mathfrak{m}$ and $\mathfrak{n}$ can be written as of the forms $\mathfrak{m} = [x_1, \ldots, x_n]$ and $\mathfrak{n} = [y_1, \ldots, y_n]$ in such a way that $x_i > y_i$ for all $i \in 1..n$. Observe that if $\mathfrak{m} :\succ: \mathfrak{n}$ then for all $k \in \mathbb{N}_0$ we have that $\mathfrak{m} \succeq k \otimes \mathfrak{n}$. Another easy-to-check property is that if $\mathfrak{m} :\succ: \mathfrak{n}$ and $\mathfrak{m}$ is non-empty then $\mathfrak{m} \succ \mathfrak{n}$.

A first frustrated attempt. As mentioned in the introduction, Turing's measure, given by $\mathcal{T}(M) \overset{\text{def}}{=} [d \mid\mid R \text{ is a redex occurrence of degree } d \text{ in } M]$, decreases when contracting the rightmost redex of highest degree. Our goal is to mend the $\mathcal{T}$-measure in such a way that contracting *any* redex decreases the measure. The difficulty is that a redex of degree d may copy redexes of a higher or equal degree $d' \geq d$. So one can wonder: whenever a redex R of degree d makes n copies of a redex S of degree $d' \geq d$, in what sense can the copies of S be considered "smaller" than S? To address this, we generalize the $\mathcal{T}$-measure to a family of measures $\mathcal{T}_D(M) \stackrel{\text{def}}{=} [(d, \mathcal{T}_{d-1}(M)) \mid\mid R \text{ is a redex occurrence of degree } d \leq D \text{ in } M]$ indexed by a degree $D \in \mathbb{N}_0$. Note that $\mathcal{T}_0(M)$ is the empty multiset because there are no redexes of degree 0.

Let us try to argue that if $d \leq D$ and $M \xrightarrow{d}_\beta N$ then $\mathcal{T}_D(M) \succ \mathcal{T}_D(N)$. Here $M \xrightarrow{d}_\beta N$ means that $M \to_\beta N$ by contracting a redex of degree d. Suppose that the contraction of the redex $R : M \xrightarrow{d}_\beta N$ copies a redex S of degree d', where we assume that $d < d' \leq D$, producing n copies $S_1, \dots, S_n$. Note that the contribution of S to the multiset is $(d', \mathcal{T}_{d'-1}(M))$, and the contribution of each S_i is $(d', \mathcal{T}_{d'-1}(N))$. By induction on D, we could inductively argue that $\mathcal{T}_{d'-1}(M) \succ \mathcal{T}_{d'-1}(N)$, since $d' - 1 < d' \leq D$. So far the property would seem to hold.

The problem with this proposal is that a redex R of degree d may still make copies of redexes of degree *exactly* d, whose contribution does not necessarily decrease[2].

A second frustrated attempt. The difficulty is to deal with the situation in which a redex R of degree d makes n copies of a redex S of the same degree d. A key observation is that a reduction sequence $M \xrightarrow{d}{}^*_\beta N$ must be a *development*[3] of the set of redexes of degree d. This is because contracting a redex of degree d can only create redexes of degree strictly less than d, so any redex of degree d that remains after one $\xrightarrow{d}_\beta$-step must be a *residual* of a preexisting redex. This motivates our second attempt to define a measure, consisting of two families of measures $\mathcal{T}^\beta_{\leq D}(-)$ and $\mathcal{R}^\beta_D(-)$, indexed by $D \in \mathbb{N}_0$ and defined mutually recursively:

$$\mathcal{T}^\beta_{\leq D}(M) \stackrel{\text{def}}{=} [(d, \mathcal{R}^\beta_d(M)) \mid\mid R \text{ is a } \beta\text{-redex occurrence of degree } d \leq D \text{ in } M]$$

$$\mathcal{R}^\beta_D(M) \stackrel{\text{def}}{=} [\mathcal{T}^\beta_{\leq D-1}(M') \mid\mid \rho : M \xrightarrow{D}{}^*_\beta M']$$

Note that there are no redexes of degree 0, so $\mathcal{T}^\beta_{\leq D}(M)$ may not depend on $\mathcal{R}^\beta_0(M)$. In fact, $\mathcal{R}^\beta_D(M)$ is defined only for $D \geq 1$. The recursive definition is well-founded because $\mathcal{T}^\beta_{\leq D}(M)$ may depend on $\mathcal{R}^\beta_1(M), \dots, \mathcal{R}^\beta_D(M)$ which in turn may only depend on $\mathcal{T}^\beta_{\leq d}(M')$ for $d < D$. The multiplicity of $\mathcal{T}^\beta_{\leq D-1}(M')$ in the multiset $\mathcal{R}^\beta_D(M)$ is given by the number of reduction sequences that contract only redexes of degree D, that is, the number of different paths $M \xrightarrow{D}{}^*_{\mathbf{m}} M'$. One important point is that, for the measure $\mathcal{R}^\beta_D(t)$ to be well defined, one needs to argue that the number of paths $M \xrightarrow{D}{}^*_{\mathbf{m}} M'$ is finite. Since $M \xrightarrow{D}{}^*_{\mathbf{m}} M'$ is a development, this is a consequence of the *finite developments* (FD) property for orthogonal HRSs [31, Theorem 11.5.11].[4]

[2] For example, in $M = (\lambda x^0.\, y^{0\to0\to0}\, x^0\, x^0)\,((\lambda z^0.\, z^0)\, w^0) \xrightarrow{1}_\beta y^{0\to0\to0}\,((\lambda z^0.\, z^0)\, w^0)\,((\lambda z^0.\, z^0)\, w^0) = N$ the measure does not decrease, as $\mathcal{T}_1(M) = [(1, []), (1, [])] = \mathcal{T}_1(N)$.

[3] Recall that a development of a set of redexes X is a reduction sequence $M \to^*_\beta N$ in which each step contracts a *residual* of a redex in X. The residuals of a redex $S : t \to_\beta s$ after the contraction of a redex $R : t \to_\beta t'$ are, informally speaking, the "copies" left of S in t'. For formal definitions see [3, Section 11.2].

[4] Note that FD only ensures that developments are finite. To see that the set $\{\rho \mid M \xrightarrow{D}{}^*_{\mathbf{m}} M'\}$ is finite, one should resort to König's lemma, together with the fact that the STLC is finitely branching. For a constructive proof, one can use a computable decreasing measure, such as in de Vrijer's proof of FD [9].

Let us try to argue that if $d \leq D$ and $M \xrightarrow{d}_{\beta} N$ then $\mathcal{T}^{\beta}_{\leq D}(M) \succ \mathcal{T}^{\beta}_{\leq D}(N)$. On the first hand, if a redex $R : M \xrightarrow{d}_{\beta} N$ of degree d copies a redex S of *exactly* the same degree d making n copies $S_1, \ldots, S_n$, the contribution of S to the multiset is $(d, \mathcal{R}^{\beta}_{d}(M))$, whereas each S_i contributes $(d, \mathcal{R}^{\beta}_{d}(N))$, and we can argue that $\mathcal{R}^{\beta}_{d}(M) \succ \mathcal{R}^{\beta}_{d}(N)$, because we can injectively map each reduction sequence $\rho : N \xrightarrow{d}^{*}_{\beta} N'$ to the reduction sequence $R\rho : M \xrightarrow{d}_{\beta} N \xrightarrow{d}^{*}_{\beta} N'$, where $R\rho$ denotes the composition of R and ρ. Furthermore, there is an empty reduction sequence $M \xrightarrow{d}^{*}_{\beta} M$ contributing an element $\mathcal{T}^{\beta}_{\leq d-1}(M)$ to $\mathcal{R}^{\beta}_{d}(M)$ but not to $\mathcal{R}^{\beta}_{d}(N)$.

On the other hand, if the contraction of a redex $R : M \xrightarrow{d}_{\beta} N$ of degree d copies a redex S of *strictly* greater degree $d' > d$ making n copies $S_1, \ldots, S_n$, the weight of S is $(d', \mathcal{R}^{\beta}_{d'}(M))$ and the weight of each S_i is $(d', \mathcal{R}^{\beta}_{d'}(N))$, and we would need to show that $\mathcal{R}^{\beta}_{d'}(M) \succ \mathcal{R}^{\beta}_{d'}(N)$. One way to do so would be to map each reduction sequence $\rho : N \xrightarrow{d}^{*}_{\beta} N'$ to a reduction sequence $\sigma : M \xrightarrow{d}^{*}_{\beta} M'$ such that $\mathcal{T}^{\beta}_{\leq d'-1}(M') \succ \mathcal{T}^{\beta}_{\leq d'-1}(N')$. However, there does not seem to be a way to rule out the possibility that σ might erase R and that $M' = N'$, which would yield $\mathcal{T}^{\beta}_{\leq d'-1}(M') = \mathcal{T}^{\beta}_{\leq d'-1}(N')$, rather than a strict inequality. The root of the problem seems again to be *erasure*.

Definition of the $\mathcal{T}^{\mathbf{m}}$-measure. The $\mathcal{T}^{\mathbf{m}}$-measure is based on the ideas described above, but considering reduction in the $\lambda^{\mathbf{m}}$-calculus rather than in the STLC, to ensure that there is no erasure. Informally, the $\mathcal{T}^{\mathbf{m}}$-measure is defined by means of the two following equations. These equations are exactly as the ones defining $\mathcal{T}^{\beta}_{\leq D}(-)$ and $\mathcal{R}^{\beta}_{D}(-)$ above, with the only difference that they deal with $\lambda^{\mathbf{m}}$-terms and $\to_{\mathbf{m}}$-reduction rather than with pure λ-terms and $\to_{\beta}$-reduction:

$$\mathcal{T}^{\mathbf{m}}_{\leq D}(t) \overset{\text{def}}{=} [(d, \mathcal{R}^{\mathbf{m}}_{d}(t)) \mid\mid R \text{ is a } \mathbf{m}\text{-redex occurrence of degree } d \leq D \text{ in } t]$$

$$\mathcal{R}^{\mathbf{m}}_{D}(t) \overset{\text{def}}{=} [\mathcal{T}^{\mathbf{m}}_{\leq D-1}(t') \mid\mid \rho : t \xrightarrow{D}^{*}_{\mathbf{m}} t']$$

To be able to reason about these measures inductively, it will be convenient to define an auxiliary measure $\mathcal{T}^{\mathbf{m}}_{d}(t_0, t)$ as the multiset of elements of the form $(d, \mathcal{R}^{\mathbf{m}}_{d}(t_0))$ for each $\mathbf{m}$-redex occurrence of degree *exactly* d in t. This auxiliary measure takes two arguments t_0 and t, and it is defined by structural recursion on the second argument (t), while the first argument (t_0) is used to keep track of the original term. Note that, with this auxiliary definition, we can write $\mathcal{T}^{\mathbf{m}}_{\leq D}(t)$ as a sum, namely $\mathcal{T}^{\mathbf{m}}_{\leq D}(t) = \mathcal{T}^{\mathbf{m}}_{1}(t, t) + \ldots + \mathcal{T}^{\mathbf{m}}_{D}(t, t)$.

To define the measure formally, we start by precisely defining its codomain.

▶ **Definition 23** (Codomain of the $\mathcal{T}^{\mathbf{m}}$-measure). *For each $d \geq 0$, we define a set $\mathbb{T}_d$, and for $d \geq 1$ we define a set $\mathbb{R}_d$, mutually recursively:*

$$\mathbb{T}_d \overset{\text{def}}{=} \mathbb{M}(\{(i, b) \mid 1 \leq i \leq d,\ b \in \mathbb{R}_i\}) \qquad \mathbb{R}_d \overset{\text{def}}{=} \mathbb{M}(\mathbb{T}_{d-1})$$

The sets $\mathbb{T}_d$ and $\mathbb{R}_d$ are partially ordered by the induced multiset ordering on their elements. Tuples (i, b) are ordered with the lexicographic order, that is, $(i, b) > (i', b')$ if and only if $i > i' \vee (i = i' \wedge b \succ b')$. Note that $\mathbb{T}_0 = \{[\,]\}$ and that if $d \leq d'$ then $\mathbb{T}_d \subseteq \mathbb{T}_{d'}$ and $\mathbb{R}_d \subseteq \mathbb{R}_{d'}$. Moreover, $(\mathbb{T}_d, \succ)$ and $(\mathbb{R}_d, \succ)$ are well-founded partial orders by Thm. 22.

Given typable $\lambda^{\mathbf{m}}$-terms t_0, t, and $d \in \mathbb{N}_0$, we define $\mathcal{T}^{\mathbf{m}}_{d}(t_0, t) \in \mathbb{T}_d$ and $\mathcal{T}^{\mathbf{m}}_{\leq d}(t) \in \mathbb{T}_d$, and if $d > 0$ we define $\mathcal{R}^{\mathbf{m}}_{d}(t) \in \mathbb{R}_d$, by induction on d as follows. Note that $\mathcal{T}^{\mathbf{m}}_{d}(t_0, t)$ is defined by a nested induction on t, and it is also defined on memories ($\mathcal{T}^{\mathbf{m}}_{d}(t_0, \mathtt{L})$):

▶ **Definition 24** (The measures $\mathcal{T}^{\mathbf{m}}_d(-,-)$, $\mathcal{T}^{\mathbf{m}}_{\leq d}(-)$, and $\mathcal{R}^{\mathbf{m}}_d(-)$)**.**

$$
\begin{aligned}
\mathcal{T}^{\mathbf{m}}_d(t_0, x) &\overset{\text{def}}{=} [\,] \\
\mathcal{T}^{\mathbf{m}}_d(t_0, \lambda x.\, s) &\overset{\text{def}}{=} \mathcal{T}^{\mathbf{m}}_d(t_0, s) \\
\mathcal{T}^{\mathbf{m}}_d(t_0, s\, u) &\overset{\text{def}}{=} \begin{cases} \mathcal{T}^{\mathbf{m}}_d(t_0, s') + \mathcal{T}^{\mathbf{m}}_d(t_0, \mathtt{L}) + \mathcal{T}^{\mathbf{m}}_d(t_0, u) + [(d, \mathcal{R}^{\mathbf{m}}_d(t_0))] \\ \hfill \textit{if } s = (\lambda x.\, s')\mathtt{L} \textit{ and it is of degree } d \\ \mathcal{T}^{\mathbf{m}}_d(t_0, s) + \mathcal{T}^{\mathbf{m}}_d(t_0, u) \hfill \textit{otherwise} \end{cases} \\
\mathcal{T}^{\mathbf{m}}_d(t_0, s\{u\}) &\overset{\text{def}}{=} \mathcal{T}^{\mathbf{m}}_d(t_0, s) + \mathcal{T}^{\mathbf{m}}_d(t_0, u)
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{T}^{\mathbf{m}}_d(t_0, \Box) &\overset{\text{def}}{=} [\,] \\
\mathcal{T}^{\mathbf{m}}_d(t_0, \mathtt{L}\{t\}) &\overset{\text{def}}{=} \mathcal{T}^{\mathbf{m}}_d(t_0, \mathtt{L}) + \mathcal{T}^{\mathbf{m}}_d(t_0, t)
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{T}^{\mathbf{m}}_{\leq d}(t) &\overset{\text{def}}{=} \textstyle\sum_{i=1}^{d} \mathcal{T}^{\mathbf{m}}_i(t, t) \\
\mathcal{R}^{\mathbf{m}}_d(t) &\overset{\text{def}}{=} [\mathcal{T}^{\mathbf{m}}_{\leq d-1}(t') \mid\mid \rho : t \xrightarrow{d}{}^{*}_{\mathbf{m}} t']
\end{aligned}
$$

Moreover, the $\mathcal{T}^{\mathbf{m}}$**-measure** itself is defined for λ-terms as follows:

▶ **Definition 25.** *If M is a typable λ-term, $\mathcal{T}^{\mathbf{m}}(M) \overset{\text{def}}{=} \mathcal{T}^{\mathbf{m}}_{\leq D}(M)$ where $D :=$* $\mathsf{maxdeg}(M)$*.*

When we write $\mathcal{T}^{\mathbf{m}}_{\leq D}(M)$, we implicitly regard M as a $\lambda^{\mathbf{m}}$-term without any memorized terms.

From a higher-level perspective, the $\mathcal{T}^{\mathbf{m}}_d(t_0, t)$ measure defined above is the multiset of pairs of the form $(d, \mathcal{R}^{\mathbf{m}}_d(t_0))$ for each redex of degree d in t. Similarly, $\mathcal{T}^{\mathbf{m}}_{\leq D}(t)$ is the multiset of pairs of the form $(d, \mathcal{R}^{\mathbf{m}}_d(t))$ for each redex of degree $d \leq D$ in t. In particular, $\mathcal{T}^{\mathbf{m}}_0(t_0, t)$ and $\mathcal{T}^{\mathbf{m}}_{\leq 0}(t)$ are empty multisets, because there are no redexes of degree 0. Two easy remarks are that $D \leq D'$ implies $\mathcal{T}^{\mathbf{m}}_{\leq D}(t) \preceq \mathcal{T}^{\mathbf{m}}_{\leq D'}(t)$, and that $\mathcal{T}^{\mathbf{m}}_d(t_0, t\mathtt{L}) = \mathcal{T}^{\mathbf{m}}_d(t_0, t) + \mathcal{T}^{\mathbf{m}}_d(t_0, \mathtt{L})$.

▶ Remark 26. As mentioned in the preceding discussion, one important point is that for $\mathcal{R}^{\mathbf{m}}_d(-)$ to be well-defined we need to argue that the set $\{\rho \mid \exists t'.\ \rho : t \xrightarrow{d}{}^{*}_{\mathbf{m}} t'\}$ is finite. This is a consequence of Coro. 19.

▶ **Example 27.** Let $\Delta := \lambda x^{0\to 0}.\, x^{0\to 0}(x^{0\to 0} z^0)$ and $W := \lambda y^0.\, w^0$ and consider the diagram:

$$\begin{tikzcd}
 & & t_2 = w\{Wz\}\{W\} \arrow[dr, "1"] & \\
t_0 = \Delta\, W \arrow[r, "2"] & t_1 = (W(Wz))\{W\} \arrow[ur, "1"] \arrow[dr, "1"] & & t_4 = w\{w\{z\}\}\{W\} \\
 & & t_3 = (W(w\{z\}))\{W\} \arrow[ur, "1"] &
\end{tikzcd}$$

Then $\mathcal{T}^{\mathbf{m}}_{\leq 0}(t_1) = \mathcal{T}^{\mathbf{m}}_{\leq 0}(t_2) = \mathcal{T}^{\mathbf{m}}_{\leq 0}(t_3) = \mathcal{T}^{\mathbf{m}}_{\leq 0}(t_4) = \mathcal{T}^{\mathbf{m}}_{\leq 1}(t_4) = \mathcal{T}^{\mathbf{m}}_{\leq 2}(t_4) = [\,]$, and:

$$
\begin{array}{ll}
\mathcal{T}^{\mathbf{m}}_{\leq 2}(t_0) = [(2, \mathcal{R}^{\mathbf{m}}_2(t_0))] & \mathcal{R}^{\mathbf{m}}_2(t_0) = [\mathcal{T}^{\mathbf{m}}_{\leq 1}(t_0), \mathcal{T}^{\mathbf{m}}_{\leq 1}(t_1)] \\
\mathcal{T}^{\mathbf{m}}_{\leq 2}(t_1) = \mathcal{T}^{\mathbf{m}}_{\leq 1}(t_1) = [(1, \mathcal{R}^{\mathbf{m}}_1(t_1)), (1, \mathcal{R}^{\mathbf{m}}_1(t_1))] & \mathcal{R}^{\mathbf{m}}_1(t_1) = [\mathcal{T}^{\mathbf{m}}_{\leq 0}(t_1), \mathcal{T}^{\mathbf{m}}_{\leq 0}(t_2), \mathcal{T}^{\mathbf{m}}_{\leq 0}(t_3), \mathcal{T}^{\mathbf{m}}_{\leq 0}(t_4)] \\
\mathcal{T}^{\mathbf{m}}_{\leq 2}(t_2) = \mathcal{T}^{\mathbf{m}}_{\leq 1}(t_2) = [(1, \mathcal{R}^{\mathbf{m}}_1(t_2))] & \mathcal{R}^{\mathbf{m}}_1(t_2) = [\mathcal{T}^{\mathbf{m}}_{\leq 0}(t_2), \mathcal{T}^{\mathbf{m}}_{\leq 0}(t_4)] \\
\mathcal{T}^{\mathbf{m}}_{\leq 2}(t_3) = \mathcal{T}^{\mathbf{m}}_{\leq 1}(t_3) = [(1, \mathcal{R}^{\mathbf{m}}_1(t_3))] & \mathcal{R}^{\mathbf{m}}_1(t_3) = [\mathcal{T}^{\mathbf{m}}_{\leq 0}(t_3), \mathcal{T}^{\mathbf{m}}_{\leq 0}(t_4)]
\end{array}
$$

In particular, $\mathcal{T}^{\mathbf{m}}_{\leq 2}(t_0) \succ \mathcal{T}^{\mathbf{m}}_{\leq 2}(t_1) \succ \mathcal{T}^{\mathbf{m}}_{\leq 2}(t_2) \succ \mathcal{T}^{\mathbf{m}}_{\leq 2}(t_4)$ and $\mathcal{T}^{\mathbf{m}}_{\leq 2}(t_1) \succ \mathcal{T}^{\mathbf{m}}_{\leq 2}(t_3) \succ \mathcal{T}^{\mathbf{m}}_{\leq 2}(t_4)$.

The $\mathcal{T}^{\mathbf{m}}$-measure is decreasing. Lastly, we show the main theorem of this section, stating that if $M \to_\beta N$ then $\mathcal{T}^{\mathbf{m}}(M) \succ \mathcal{T}^{\mathbf{m}}(N)$. This theorem is based on three technical results, that we call *high/increase*, *low/decrease*, and *forget/decrease*:

1. **High/increase** (Prop. 29) establishes – perhaps confusingly – that $\mathcal{T}^{\mathbf{m}}_{\leq d}(-)$ (non-strictly) **increases** if one contracts a redex of higher degree $D > d$. More precisely, if $0 \leq d < D$ and $t \xrightarrow{D}_{\mathbf{m}} t'$ then $\mathcal{T}^{\mathbf{m}}_{\leq d}(t) \preceq \mathcal{T}^{\mathbf{m}}_{\leq d}(t')$. Note that $\mathcal{T}^{\mathbf{m}}_{\leq d}(t)$ only looks at redexes of degree $i \leq d$, and contracting a redex of degree $D > d$ *cannot erase* a redex of any degree $i \leq d$, because the $\lambda^{\mathbf{m}}$-calculus is non-erasing. Contracting a redex of degree D can, at most, replicate redexes of degree i. This property is needed for a technical reason to prove the low/decrease property, and it relies crucially on the commutation result of the previous section (Prop. 18).
2. **Low/decrease** (Prop. 30) establishes that $\mathcal{T}^{\mathbf{m}}_{\leq D}(-)$ **strictly** decreases if one contracts a redex of lower degree $d < D$. More precisely, if $1 \leq d \leq D$ and $t \xrightarrow{d}_{\mathbf{m}} t'$ then $\mathcal{T}^{\mathbf{m}}_{\leq D}(t) \succ \mathcal{T}^{\mathbf{m}}_{\leq D}(t')$. This is the core of the argument, and the most technically difficult part to prove. It relies crucially on the lifting property of the previous section (Prop. 20).
3. **Forget/decrease** (Prop. 31) establishes that forgetful reduction (non-strictly) decreases the measure. More precisely, if $t \rhd t'$ then $\mathcal{T}^{\mathbf{m}}_{\leq d}(t) \succeq \mathcal{T}^{\mathbf{m}}_{\leq d}(t')$. This property is used as a final step in the main theorem, and it relies crucially on postponement of forgetful reduction, as studied in the previous section (Prop. 21).

Below we sketch the proofs of these three properties. Let us first mention a straightforward lemma.

▶ **Lemma 28** (Measure of a substitution)**.** **1.** $\mathcal{T}^{\mathbf{m}}_{d}(t_0, t) \preceq \mathcal{T}^{\mathbf{m}}_{d}(t_0, t[x := s])$. **2.** *If s is not a $\mathbf{m}$-abstraction of degree d, then $\mathcal{T}^{\mathbf{m}}_{d}(t_0, t[x := s]) = \mathcal{T}^{\mathbf{m}}_{d}(t_0, t) + k \otimes \mathcal{T}^{\mathbf{m}}_{d}(t_0, s)$ for some $k \in \mathbb{N}_0$.*

Proof. By induction on t. ◀

▶ **Proposition 29** (High/increase)**.** *Let $D \in \mathbb{N}_0$. Then the following hold:*

1. *If $1 \leq d < D$ and $t \xrightarrow{D}_{\mathbf{m}} t'$ then $\mathcal{R}^{\mathbf{m}}_{d}(t) \preceq \mathcal{R}^{\mathbf{m}}_{d}(t')$.*
2. *If $0 \leq d < D$ and $t_0 \xrightarrow{D}_{\mathbf{m}} t'_0$ then $\mathcal{T}^{\mathbf{m}}_{d}(t_0, t) \preceq \mathcal{T}^{\mathbf{m}}_{d}(t'_0, t)$.*
3. *If $0 \leq d < D$ and $t_0 \xrightarrow{D}_{\mathbf{m}} t'_0$ and $t \xrightarrow{D}_{\mathbf{m}} t'$ then $\mathcal{T}^{\mathbf{m}}_{d}(t_0, t) \preceq \mathcal{T}^{\mathbf{m}}_{d}(t'_0, t')$.*
4. *If $0 \leq d < D$ and $t \xrightarrow{D}_{\mathbf{m}} t'$ then $\mathcal{T}^{\mathbf{m}}_{\leq d}(t) \preceq \mathcal{T}^{\mathbf{m}}_{\leq d}(t')$.*

Proof. The four items are proved simultaneously by induction on d, where item **1** resorts to the IH, and the following items may resort to the previous items without decreasing d. Items **2** and **3** proceed by a nested induction on t. Most cases are straightforward.

One interesting situation occurs in item **3** when $t = (\lambda x.\, s)\mathtt{L}\, u$ is the redex of degree D contracted by the step $t \xrightarrow{D}_{\mathbf{m}} t'$. Then we resort to the first part of Lem. 28.

Another interesting part of the proof is item **1**. Let $1 \leq d < D$ and $t \xrightarrow{D}_{\mathbf{m}} t'$ and let us show that $\mathcal{R}^{\mathbf{m}}_{d}(t) \preceq \mathcal{R}^{\mathbf{m}}_{d}(t')$. Indeed, let $X := \{\rho \mid (\exists s)\ \rho : t \xrightarrow{d}{}^{*}_{\mathbf{m}} s\}$ and $Y := \{\sigma \mid (\exists s')\ \sigma : t' \xrightarrow{d}{}^{*}_{\mathbf{m}} s'\}$, and let $R : t \xrightarrow{D}_{\mathbf{m}} t'$. Using Prop. 18, we can define an injective function $\varphi : X \to Y$ by $\varphi(\rho) := \rho/R$. Note that $\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\rho^{\mathsf{tgt}}) \preceq \mathcal{T}^{\mathbf{m}}_{\leq d-1}(\varphi(\rho)^{\mathsf{tgt}})$ holds for every $\rho \in X$ using item **4** of the IH (noting that $1 \leq d - 1 < D$ holds because $1 \leq d < D$), resorting to the IH as many times as the length of the reduction $s \xrightarrow{D}{}^{*}_{\mathbf{m}} s'_{\rho}$. To conclude the proof, let $Z = Y \setminus \varphi(X)$. Then:

$$\begin{aligned}\mathcal{R}^{\mathbf{m}}_{d}(t) &= [\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\rho^{\mathsf{tgt}}) \mid\mid \rho \in X] \preceq^{(\star)} [\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\varphi(\rho)^{\mathsf{tgt}}) \mid\mid \rho \in X] =^{(\star\star)} [\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\sigma^{\mathsf{tgt}}) \mid\mid \sigma \in \varphi(X)] \\ &\preceq [\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\sigma^{\mathsf{tgt}}) \mid\mid \sigma \in \varphi(X)] + [\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\sigma^{\mathsf{tgt}}) \mid\mid \sigma \in Z] = [\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\sigma^{\mathsf{tgt}}) \mid\mid \sigma \in Y] = \mathcal{R}^{\mathbf{m}}_{d}(t')\end{aligned}$$

To justify the step marked with ($\star$), note that $[\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\rho^{\mathsf{tgt}}) \mid\mid \rho \in X] = \sum_{\rho\in X}[\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\rho^{\mathsf{tgt}})] \preceq \sum_{\rho\in X}[\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\varphi(\rho)^{\mathsf{tgt}})] = [\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\varphi(\rho)^{\mathsf{tgt}}) \mid\mid \rho \in X]$ because $\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\rho^{\mathsf{tgt}}) \preceq \mathcal{T}^{\mathbf{m}}_{\leq d-1}(\varphi(\rho)^{\mathsf{tgt}})$, as we have already claimed. To justify the step marked with ($\star\star$), note that φ is injective. ◀

▶ **Proposition 30** (Low/decrease). *Let $D \in \mathbb{N}_0$. Then the following hold:*

1. *If $1 \leq d \leq j \leq D$ and $t \xrightarrow{d}_{\mathbf{m}} t'$ then $\mathcal{R}^{\mathbf{m}}_j(t) \succ \mathcal{R}^{\mathbf{m}}_j(t')$.*
2. *If $1 \leq d \leq j \leq D$ and $t_0 \xrightarrow{d}_{\mathbf{m}} t'_0$ then $\mathcal{T}^{\mathbf{m}}_j(t_0, t) \mathrel{:\succ:} \mathcal{T}^{\mathbf{m}}_j(t'_0, t)$.*
3. *If $1 \leq d \leq D$ and $t_0 \xrightarrow{d}_{\mathbf{m}} t'_0$ and $t \xrightarrow{d}_{\mathbf{m}} t'$, then for all $\mathfrak{m} \in \mathbb{T}_{d-1}$ we have $\mathcal{T}^{\mathbf{m}}_d(t_0, t) \succ \mathcal{T}^{\mathbf{m}}_d(t'_0, t') + \mathfrak{m}$.*
4. *If $1 \leq d < j \leq D$ and $t_0 \xrightarrow{d}_{\mathbf{m}} t'_0$ and $t \xrightarrow{d}_{\mathbf{m}} t'$ then $\mathcal{T}^{\mathbf{m}}_j(t_0, t) \succeq \mathcal{T}^{\mathbf{m}}_j(t'_0, t')$.*
5. *If $1 \leq d \leq D$ and $t \xrightarrow{d}_{\mathbf{m}} t'$ then $\mathcal{T}^{\mathbf{m}}_{\leq D}(t) \succ \mathcal{T}^{\mathbf{m}}_{\leq D}(t')$.*

Proof. The five items are proved simultaneously by induction on D, where item **1** resorts to the IH, and the following items may resort to the previous items without decreasing d. Items **2**–**4** proceed by a nested induction on t. We mention some of the interesting parts of the proof.

For item **1**, let $1 \leq d \leq j \leq D$ and $t \xrightarrow{d}_{\mathbf{m}} t'$ and let us show that $\mathcal{R}^{\mathbf{m}}_j(t) \succ \mathcal{R}^{\mathbf{m}}_j(t')$. Let $X := \{\rho \mid (\exists s)\ \rho : t \xrightarrow{j}{}^{*}_{\mathbf{m}} s\}$ and $Y := \{\sigma \mid (\exists s')\ \sigma : t' \xrightarrow{j}{}^{*}_{\mathbf{m}} s'\}$, and consider two subcases:

- If $d = j$, let $R : t \xrightarrow{d}_{\mathbf{m}} t'$, define an injective function $\varphi : Y \to X$ by $\varphi(\sigma) = R\sigma$, let $Z = X \setminus \varphi(Y)$, and note that:

$$\begin{aligned}
\mathcal{R}^{\mathbf{m}}_j(t) &= [\mathcal{T}^{\mathbf{m}}_{\leq j-1}(\rho^{\mathsf{tgt}}) \mid\mid \rho \in \varphi(Y)] + [\mathcal{T}^{\mathbf{m}}_{\leq j-1}(\rho^{\mathsf{tgt}}) \mid\mid \rho \in Z] \\
&= [\mathcal{T}^{\mathbf{m}}_{\leq j-1}(R\sigma^{\mathsf{tgt}}) \mid\mid \sigma \in Y] + [\mathcal{T}^{\mathbf{m}}_{\leq j-1}(\rho^{\mathsf{tgt}}) \mid\mid \rho \in Z] \text{ since } \varphi \text{ is injective} \\
&= [\mathcal{T}^{\mathbf{m}}_{\leq j-1}(\sigma^{\mathsf{tgt}}) \mid\mid \sigma \in Y] + [\mathcal{T}^{\mathbf{m}}_{\leq j-1}(\rho^{\mathsf{tgt}}) \mid\mid \rho \in Z] = \mathcal{R}^{\mathbf{m}}_j(t') + [\mathcal{T}^{\mathbf{m}}_{\leq j-1}(\rho^{\mathsf{tgt}}) \mid\mid \rho \in Z]
\end{aligned}$$

  To conclude that $\mathcal{R}^{\mathbf{m}}_j(t) \succ \mathcal{R}^{\mathbf{m}}_j(t')$, note that Z is non-empty because it contains the empty reduction $\epsilon : t \xrightarrow{d}{}^{*}_{\mathbf{m}} t$.
- If $d < j$, we construct a function $\varphi : Y \to X$ as follows. By Prop. 20, for each reduction $\sigma : t' \xrightarrow{j}{}^{*}_{\mathbf{m}} s'$ there exist s_σ, u_σ, and reductions $\varphi(\sigma) : t \xrightarrow{j}{}^{*}_{\mathbf{m}} s_\sigma$ and $s' \xrightarrow{j}{}^{*}_{\mathbf{m}} u_\sigma$ and $s_\sigma \xrightarrow{d}{}^{+}_{\mathbf{m}} u_\sigma$. Note that for every $\sigma \in Y$ we have $\mathcal{T}^{\mathbf{m}}_{\leq j-1}(\varphi(\sigma)^{\mathsf{tgt}}) = \mathcal{T}^{\mathbf{m}}_{\leq j-1}(s_\sigma) \succ^{\dagger} \mathcal{T}^{\mathbf{m}}_{\leq j-1}(u_\sigma) \succeq^{\ddagger} \mathcal{T}^{\mathbf{m}}_{\leq j-1}(s') = \mathcal{T}^{\mathbf{m}}_{\leq j-1}(\sigma^{\mathsf{tgt}})$ where † holds by item 5 of the IH observing that $1 \leq d \leq j-1 < D$ because $d < j \leq D$, and ‡ holds by high/increase (Prop. 29) observing that $0 \leq j-1 < j$. To conclude the proof, let $Z = X \setminus \varphi(Y)$, and note that:

$$\begin{aligned}
\mathcal{R}^{\mathbf{m}}_j(t) &= [\mathcal{T}^{\mathbf{m}}_{\leq j-1}(\rho^{\mathsf{tgt}}) \mid\mid \rho \in \varphi(Y)] + [\mathcal{T}^{\mathbf{m}}_{\leq j-1}(\rho^{\mathsf{tgt}}) \mid\mid \rho \in Z] \\
&= [\mathcal{T}^{\mathbf{m}}_{\leq j-1}(\varphi(\sigma)^{\mathsf{tgt}}) \mid\mid \sigma \in Y] + [\mathcal{T}^{\mathbf{m}}_{\leq j-1}(\rho^{\mathsf{tgt}}) \mid\mid \rho \in Z] \\
&\succeq [\mathcal{T}^{\mathbf{m}}_{\leq j-1}(\varphi(\sigma)^{\mathsf{tgt}}) \mid\mid \sigma \in Y] \succ^{(\star)} [\mathcal{T}^{\mathbf{m}}_{\leq j-1}(\sigma^{\mathsf{tgt}}) \mid\mid \sigma \in Y] = \mathcal{R}^{\mathbf{m}}_j(t')
\end{aligned}$$

  For the step marked with ($\star$), note that $[\mathcal{T}^{\mathbf{m}}_{\leq j-1}(\varphi(\sigma)^{\mathsf{tgt}}) \mid\mid \sigma \in Y] \mathrel{:\succ:} [\mathcal{T}^{\mathbf{m}}_{\leq j-1}(\sigma^{\mathsf{tgt}}) \mid\mid \sigma \in Y]$ because $\mathcal{T}^{\mathbf{m}}_{\leq j-1}(\varphi(\sigma)^{\mathsf{tgt}}) \succ \mathcal{T}^{\mathbf{m}}_{\leq j-1}(\sigma^{\mathsf{tgt}})$ holds by the claim above where, moreover, Y is non-empty because it contains the empty reduction $\epsilon : t' \xrightarrow{j}{}^{*}_{\mathbf{m}} t'$.

Another interesting situation occurs in item **3**, when $t = (\lambda x.\, s)\mathtt{L}\, u$ is the redex of degree d contracted by the step $t \xrightarrow{d}_{\mathbf{m}} t'$. The step is of the form $t = (\lambda x.\, s)\mathtt{L}\, u \xrightarrow{d}_{\mathbf{m}} s[x := u]\{u\}\mathtt{L} = t'$. Note that u is not an abstraction of degree d, because it is the argument of an abstraction of degree d. So by Lem. 28 there exists $k \in \mathbb{N}_0$ such that $\mathcal{T}^{\mathbf{m}}_d(t'_0, s[x := u]) = \mathcal{T}^{\mathbf{m}}_d(t'_0, s) + k \otimes \mathcal{T}^{\mathbf{m}}_d(t'_0, u)$. The crucial observation is that $\mathcal{T}^{\mathbf{m}}_d(t_0, u) \succeq (1 + k) \otimes \mathcal{T}^{\mathbf{m}}_d(t'_0, u)$, which is because by item 2 we have that $\mathcal{T}^{\mathbf{m}}_d(t_0, u) \mathrel{:\succ:} \mathcal{T}^{\mathbf{m}}_d(t'_0, u)$.

Finally, for item **5**, let $1 \leq d \leq D$ and $t \overset{d}{\to}_{\mathbf{m}} t'$ and let us show that $\mathcal{T}^{\mathbf{m}}_{\leq D}(t) \succ \mathcal{T}^{\mathbf{m}}_{\leq D}(t')$. Indeed:

$$\begin{aligned}
\mathcal{T}^{\mathbf{m}}_{\leq D}(t) &= \sum_{i=1}^{D} \mathcal{T}^{\mathbf{m}}_{i}(t,t) \succeq \mathcal{T}^{\mathbf{m}}_{d}(t,t) + \sum_{j=d+1}^{D} \mathcal{T}^{\mathbf{m}}_{j}(t,t) \\
&\succ \mathcal{T}^{\mathbf{m}}_{\leq d-1}(t') + \mathcal{T}^{\mathbf{m}}_{d}(t',t') + \sum_{j=d+1}^{D} \mathcal{T}^{\mathbf{m}}_{j}(t,t) \text{ by item } \mathbf{3}\text{, taking } \mathfrak{m} := \mathcal{T}^{\mathbf{m}}_{\leq d-1}(t') \\
&\succeq \mathcal{T}^{\mathbf{m}}_{\leq d-1}(t') + \mathcal{T}^{\mathbf{m}}_{d}(t',t') + \sum_{j=d+1}^{D} \mathcal{T}^{\mathbf{m}}_{j}(t',t') = \mathcal{T}^{\mathbf{m}}_{\leq D}(t') \text{ by item } \mathbf{4}.
\end{aligned}$$ ◀

▶ **Proposition 31** (Forget/decrease). *Let $d \in \mathbb{N}_0$. Then the following hold:*

1. *If $t \rhd t'$ then $\mathcal{R}^{\mathbf{m}}_{d}(t) \succeq \mathcal{R}^{\mathbf{m}}_{d}(t')$.*
2. *If $t_0 \rhd t'_0$ then $\mathcal{T}^{\mathbf{m}}_{d}(t_0,t) \succeq \mathcal{T}^{\mathbf{m}}_{d}(t'_0,t)$.*
3. *If $t_0 \rhd t'_0$ and $t \rhd t'$ then $\mathcal{T}^{\mathbf{m}}_{d}(t_0,t) \succeq \mathcal{T}^{\mathbf{m}}_{d}(t'_0,t')$.*
4. *If $t \rhd t'$ then $\mathcal{T}^{\mathbf{m}}_{\leq d}(t) \succeq \mathcal{T}^{\mathbf{m}}_{\leq d}(t')$.*

Proof. The four items are proved simultaneously by induction on D, where item **1** resorts to the IH, and the following items may resort to the previous items without decreasing d. Items **2** and **3** proceed by a nested induction on t.

The interesting part is item **1**, so let $t \rhd t'$ and let us show that $\mathcal{R}^{\mathbf{m}}_{d}(t) \succeq \mathcal{R}^{\mathbf{m}}_{d}(t')$. Let $X := \{\rho \mid (\exists s)\ \rho : t \overset{d}{\to}^{*}_{\mathbf{m}} s\}$ and $Y := \{\sigma \mid (\exists s')\ \sigma : t' \overset{d}{\to}^{*}_{\mathbf{m}} s'\}$. Define an injective function $\varphi : Y \to X$ by $\varphi(\sigma) := \sigma^{\frown} R$, resorting to Prop. 21, where $\sigma^{\frown} R : t \overset{d}{\to}^{*}_{\mathbf{m}} s_{\sigma}$. and $s_{\sigma} \rhd^{*} s'$. Note that for every $\sigma \in Y$ we have $\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\varphi(\sigma)^{\mathsf{tgt}}) = \mathcal{T}^{\mathbf{m}}_{\leq d-1}(s_{\sigma}) \succeq^{\dagger} \mathcal{T}^{\mathbf{m}}_{\leq d-1}(s') = \mathcal{T}^{\mathbf{m}}_{\leq d-1}(\sigma^{\mathsf{tgt}})$, where † holds by item **4** of the IH, observing that $d-1 < d$. To conclude the proof, let $Z = X \setminus \varphi(Y)$, and note that:

$$\begin{aligned}
\mathcal{R}^{\mathbf{m}}_{d}(t) &= [\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\rho^{\mathsf{tgt}}) \mid\mid \rho \in \varphi(Y)] + [\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\rho^{\mathsf{tgt}}) \mid\mid \rho \in Z] \succeq [\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\rho^{\mathsf{tgt}}) \mid\mid \rho \in \varphi(Y)] \\
&=^{(\star)} [\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\varphi(\sigma)^{\mathsf{tgt}}) \mid\mid \sigma \in Y] \succeq^{(\star\star)} [\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\sigma^{\mathsf{tgt}}) \mid\mid \sigma \in Y] = \mathcal{R}^{\mathbf{m}}_{d}(t')
\end{aligned}$$

For the step marked with $(\star)$, note that φ is injective. For the step marked with $(\star\star)$, note that $[\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\varphi(\sigma)^{\mathsf{tgt}}) \mid\mid \sigma \in Y] = \sum_{\sigma \in Y} [\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\varphi(\sigma)^{\mathsf{tgt}})] \succeq \sum_{\sigma \in Y} [\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\sigma^{\mathsf{tgt}})] = [\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\sigma^{\mathsf{tgt}}) \mid\mid \sigma \in Y]$ because $\mathcal{T}^{\mathbf{m}}_{\leq d-1}(\varphi(\sigma)^{\mathsf{tgt}}) \succeq \mathcal{T}^{\mathbf{m}}_{\leq d-1}(\sigma^{\mathsf{tgt}})$, as we have already justified. ◀

Finally, we prove the main theorem in this section:

▶ **Theorem 32.** *Let M, N be typable λ-terms such that $M \to_{\beta} N$. Then $\mathcal{T}^{\mathbf{m}}(M) > \mathcal{T}^{\mathbf{m}}(N)$.*

Proof. Let $D = \mathsf{maxdeg}(M)$ and $D' = \mathsf{maxdeg}(N)$. Let $M \to_{\mathbf{m}} s$ be the step corresponding to $M \to_{\beta} N$. By Lem. 11 note that $s \rhd N$. Then:

$$\mathcal{T}^{\mathbf{m}}(M) = \mathcal{T}^{\mathbf{m}}_{\leq D}(M) \succ^{\text{Prop. 30}} \mathcal{T}^{\mathbf{m}}_{\leq D}(s) \succeq^{\text{Prop. 31}} \mathcal{T}^{\mathbf{m}}_{\leq D}(N) \succeq \mathcal{T}^{\mathbf{m}}_{\leq D'}(N) = \mathcal{T}^{\mathbf{m}}(N)$$

The last inequality holds because $D \geq D'$ since, as is well-known, contraction of a β-redex in the simply typed λ-calculus cannot create a redex of higher degree. ◀

6 Conclusion

We have defined two decreasing measures for the STLC, the $\mathcal{W}$-measure (Def. 12) and the $\mathcal{T}^{\mathbf{m}}$-measure (Def. 25). These measures are decreasing (Thm. 15 and Thm. 32 respectively) and, to the best of our knowledge, they provide two new proofs of strong normalization for the STLC. Both measures are defined constructively and by purely syntactic methods, using the $\lambda^{\mathbf{m}}$-calculus as an auxiliary tool.

The problem of finding a "straightforward" decreasing measure for β-reduction in the simply typed λ-calculus is posed as Problem #26 in the TLCA list of open problems [5], and as Problem #19 in the RTA list of open problems [11].

One strength of the $\mathcal{W}$-measure is that its codomain is simple: each term is mapped to a natural number. One weakness is that the definition of the $\mathcal{W}$-measure relies on reduction in the $\lambda^{\mathbf{m}}$-calculus, and computing the $\mathcal{W}$-measure is at least as costly as evaluating the λ-term itself. Measures based on Gandy's [16, 10] have similar characteristics. One question is whether the values of the $\mathcal{W}$-measure and measures based on Gandy's can be related. It is not immediate to establish a precise correspondence.

On the other hand, one strength of the $\mathcal{T}^{\mathbf{m}}$-measure is that it shows how to extend Turing's measure $\mathcal{T}(-)$ so that it decreases when contracting *any* redex. The proof is based on a delicate analysis of how contracting a redex of degree d may create and copy redexes of degree d', depending on whether $d < d'$, or $d = d'$, or $d > d'$. We hope that this may provide novel insights on why the STLC is SN. The codomain of the $\mathcal{T}^{\mathbf{m}}$-measure is not so simple, as the $\mathcal{T}^{\mathbf{m}}$-measure maps each term to a structure of nested multisets. Yet, it is "reasonably simple": the fact that the partial orders $\mathbb{T}_d$ and $\mathbb{R}_d$ are well-founded only relies on the ordinary multiset and lexicographic orderings. The $\mathcal{T}^{\mathbf{m}}$-measure is costly to compute; in particular $\mathcal{R}^{\mathbf{m}}_d(t)$ is defined as a sum over all reductions $\rho : t \xrightarrow{d}^{*}_{\mathbf{m}} t'$, which may produce a combinatorial explosion. Another weakness is that our proofs make use of relatively heavy rewriting machinery, as we have to keep explicit track of witnesses (*e.g.* in Section 4).

Besides the techniques mentioned in the introduction, other proofs of SN of the STLC can be found in the literature. For example, David [7] gives a purely syntactic proof of SN relying on the standardization theorem; Loader [23], as well as Joachimski and Matthes [18], give combinatorial proofs of SN based on inductive predicates characterizing strongly normalizing terms. As far as we know, the only proofs that explicitly construct decreasing measures are those based on Gandy's.

The idea of keeping "leftover garbage" can be traced back to at least the works of Nederpelt [21] and Klop [20], who studied non-erasing variants of (possibly) erasing rewriting systems, in order to relate weak and strong normalization. Many variations of these ideas have been explored in the past, such as in de Groote's notion of β_S reduction [8] or Neergaard and Sørensen calculus with memory [25]. Instead of using the $\lambda^{\mathbf{m}}$-calculus, it is possible that other non-erasing systems may be used. For instance, Gandy [16] translates λ-terms to the terms of λI-calculus to avoid erasing arguments.

The definition of reduction in the $\lambda^{\mathbf{m}}$-calculus, which allows arbitrary memory in between the abstraction and the application, is inspired by Accattoli and Kesner's work on calculi with explicit substitutions "at a distance" [1]. This mechanism can be traced back, again, to at least the work of Nederpelt [21].

The definition of the $\lambda^{\mathbf{m}}$-calculus as a means to obtain an *increasing* measure was inspired by the fact that, in explicit substitution calculi without erasure, labeled reduction (in the sense of Lévy labels [22]) increases the sum of the sizes of all the labels in the term [2].

References

1 Beniamino Accattoli and Delia Kesner. The structural *lambda*-calculus. In *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, pages 381–395, 2010.

2 Pablo Barenbaum and Eduardo Bonelli. Optimality and the linear substitution calculus. In *2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, September 3-9, 2017, Oxford, UK*, pages 9:1–9:16, 2017. doi:10.4230/LIPIcs.FSCD.2017.9.

3 Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103. Elsevier, 1984.

4 Henk P. Barendregt and Giulio Manzonetto. Turing's contributions to lambda calculus. In B. Cooper and J. van Leeuwen, editors, *Alan Turing - His Work and Impact*, pages 139–143. Elsevier, 2013.

5 TCLA Editorial Board. TLCA list of open problems. http://tlca.di.unito.it/opltlca/, 2006.

6 Thierry Coquand. Canonicity and normalization for dependent type theory. *Theor. Comput. Sci.*, 777:184–191, 2019. doi:10.1016/j.tcs.2019.01.015.

7 René David. Normalization without reducibility. *Ann. Pure Appl. Log.*, 107(1-3):121–130, 2001. doi:10.1016/S0168-0072(00)00030-0.

8 Philippe de Groote. The conservation theorem revisited. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*, volume 664 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 1993. doi:10.1007/BFb0037105.

9 Roel de Vrijer. A direct proof of the finite developments theorem. *The Journal of symbolic logic*, 50(2):339–343, 1985.

10 Roel de Vrijer. Exactly estimating functionals and strong normalization. In *Indagationes Mathematicae (Proceedings)*, volume 90, pages 479–493. North-Holland, 1987.

11 Nachum Dershowitz, Jean-Pierre Jouannaud, and Jan Willem Klop. Open problems in rewriting. In *International Conference on Rewriting Techniques and Applications*, pages 445–456. Springer, 1991.

12 Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.

13 Paul Downen, Philip Johnson-Freyd, and Zena M. Ariola. Abstracting models of strong normalization for classical calculi. *J. Log. Algebraic Methods Program.*, 111:100512, 2020. doi:10.1016/j.jlamp.2019.100512.

14 Jörg Endrullis, Dimitri Hendriks, Jan Willem Klop, and Andrew Polonsky. Clocked lambda calculus. *Math. Struct. Comput. Sci.*, 27(5):782–806, 2017. doi:10.1017/S0960129515000389.

15 Robin O. Gandy. An early proof of normalization by A.M. Turing. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 453–455. Academic Press, 1980.

16 Robin O. Gandy. Proofs of strong normalization. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 457–477. Academic Press, 1980.

17 Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur.* PhD thesis, Université Paris 7, 1972.

18 Felix Joachimski and Ralph Matthes. Short proofs of normalization for the simply- typed lambda-calculus, permutative conversions and Gödel's T. *Arch. Math. Log.*, 42(1):59–87, 2003. doi:10.1007/s00153-002-0156-9.

19 Zurab Khasidashvili, Mizuhito Ogawa, and Vincent van Oostrom. Uniform normalisation beyond orthogonality. In Aart Middeldorp, editor, *Rewriting Techniques and Applications, 12th International Conference, RTA 2001, Utrecht, The Netherlands, May 22-24, 2001, Proceedings*, volume 2051 of *Lecture Notes in Computer Science*, pages 122–136. Springer, 2001. doi:10.1007/3-540-45127-7_11.

20 Jan Willem Klop. *Combinatory Reduction Systems.* PhD thesis, Utrecht University, 1980.

21 Robert Pieter Nederpelt Lazarom. *Strong normalization in a typed lambda calculus with lambda structured types.* PhD thesis, TU Eindhoven, 1973.

22 Jean-Jacques Lévy. *Réductions correctes et optimales dans le lambda-calcul.* PhD thesis, Université de Paris 7, 1978.

23 Ralph Loader. Notes on simply typed lambda calculus. Technical Report ECS-LFCS-98-381, University of Edinburgh, 1998.

24 Paul-André Melliès. Axiomatic rewriting theory I: A diagrammatic standardization theorem. In Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel C. de Vrijer, editors, *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*, volume 3838 of *Lecture Notes in Computer Science*, pages 554–638. Springer, 2005. doi:10.1007/11601548_23.

25 Peter Møller Neergaard and Morten Heine Sørensen. Conservation and uniform normalization in lambda calculi with erasing reductions. *Inf. Comput.*, 178(1):149–179, 2002. doi:10.1006/inco.2002.3153.

26 Tobias Nipkow. Higher-order critical pairs. In *Proceedings 1991 Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 342–343. IEEE Computer Society, 1991.

27 Jan von Plato. Gentzen's proof of normalization for natural deduction. *Bulletin of Symbolic Logic*, 14(2):240–257, 2008.

28 Dag Prawitz. *Natural deduction: a proof-theoretical study.* PhD thesis, Almqvist & Wiksell, 1965.

29 Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard isomorphism*, volume 149. Elsevier, 2006.

30 William W. Tait. A realizability interpretation of the theory of species. In Rohit Parikh, editor, *Logic Colloquium*, pages 240–251, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg.

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

Hydra Battles and AC Termination

Nao Hirokawa
School of Information Science, JAIST, Ishikawa, Japan

Aart Middeldorp
Department of Computer Science, Universität Innsbruck, Austria

Abstract

We present a new encoding of the Battle of Hercules and Hydra as a rewrite system with AC symbols. Unlike earlier term rewriting encodings, it faithfully models any strategy of Hercules to beat Hydra. To prove the termination of our encoding, we employ type introduction in connection with many-sorted semantic labeling for AC rewriting and AC-RPO.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Theory of computation → Rewrite systems; Theory of computation → Computability

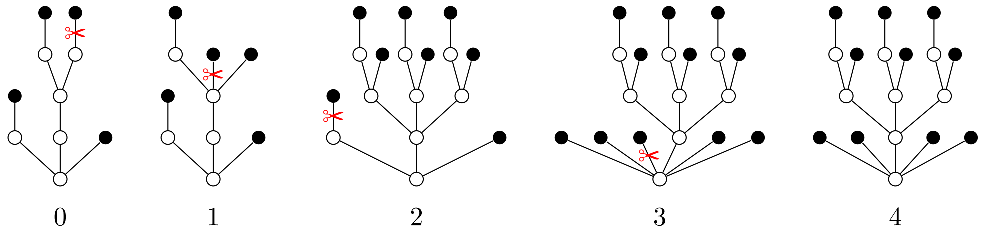
Keywords and phrases battle of Hercules and Hydra, term rewriting, termination

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.12

Funding *Nao Hirokawa*: JSPS KAKENHI Grant Number JP22K11900.
Aart Middeldorp: Part of this work was performed when the second author was employed at the Future Value Creation Research Center of Nagoya University, Japan.

1 Introduction

The mythological monster Hydra is a dragon-like creature with multiple heads. Whenever Hercules in his fight chops off a head, more and more new heads can grow instead, since the beast gets increasingly angry. Here we model a Hydra as an unordered tree. If Hercules cuts off a leaf corresponding to a head, the tree is modified in the following way: If the cut-off node h has a grandparent n, then the branch from n to the parent of h gets multiplied, where the number of copies depends on the number of decapitations so far. Hydra dies if there are no heads left, in that case Hercules wins. The following sequence shows an example fight:



Though the number of heads can grow considerably in one step, it turns out that the fight always terminates, and Hercules will win independent of his strategy. Proving termination of the Battle is challenging since Kirby and Paris proved in their landmark paper [11] that termination for an arbitrary (computable) strategy is independent of Peano arithmetic. In [11] a termination argument based on ordinals is used.

Starting with [4, p. 271], several TRS encodings of the Battle of Hercules and Hydra have been proposed and studied [3, 5, 7, 17, 21]. Touzet [21] was the first to give a rigorous termination proof and in [24] the automation of ordinal interpretations is discussed. In this paper we present yet another encoding. In contrast to earlier TRS encodings that model a specific strategy, it uses AC matching to represent *arbitrary* battles. To prove its termination, we apply and extend existing termination methods for AC rewriting.

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 12; pp. 12:1–12:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The remainder of the paper is organized as follows. After recalling some basic definitions in Section 2, we present our new encoding of the Battle in Section 3. We give a rigorous proof that our encoding faithfully represents the Battle. In Section 4 we present many-sorted semantic labeling for AC rewriting and apply it to our encoding. This results in an infinite AC rewrite system, which is subjected to AC-RPO [20] in Section 5. Related work is discussed in Section 6. In particular, we comment on earlier encodings of the Battle. We conclude in Section 7 with suggestions for future research.

2 Preliminaries

Let $\mathcal{S}$ be a set of *sorts*. An $\mathcal{S}$*-sorted signature* $\mathcal{F}$ consists of function symbols f having a sort declaration $S_1 \times \cdots \times S_n \to S$. Here $S_1, \dots, S_n$ and S are sorts in $\mathcal{S}$ and n is the *arity* of f. By $f^{(n)}$ we indicate that f has arity n. Let $\mathcal{V}$ be a countably infinite set of variables, where every variable has its own sort. We assume the existence of infinitely many variables of each sort. *Terms* of sort S are inductively defined as usual: Every variable of sort S is a term of sort S and if f has sort declaration $S_1 \times \cdots \times S_n \to S$ and t_i is a term of sort S_i for all $1 \leqslant i \leqslant n$ then $f(t_1, \dots, t_n)$ is a term of sort S. *Ground terms* are terms without variables. By $\mathcal{T}(\{f_1, \dots, f_m\})$ we denote the set of all ground terms over $\{f_1, \dots, f_m\}$. The *root symbol* $\mathsf{root}(t)$ of a term t is t if it is a variable, and f if $t = f(t_1, \dots, t_n)$. For every sort S we introduce a fresh constant $\square_S$, called the *hole*. A term over $\mathcal{F} \uplus \{\square_S \mid S \in \mathcal{S}\}$ is a *context* over $\mathcal{F}$ if it contains exactly one hole. Given a context C and a term t, we write $C[t]$ for the term resulting from replacing the hole in C by t. A mapping σ that associates each variable to a term of the same sort is a *substitution* if its domain $\{x \in \mathcal{V} \mid \sigma(x) \neq x\}$ is finite. The application $t\sigma$ of σ to a term t is defined as $\sigma(t)$ if t is a variable and $f(t_1\sigma, \dots, t_n\sigma)$ if $t = f(t_1, \dots, t_n)$. A binary relation $\to$ on terms is *closed under substitutions* if $s\sigma \to t\sigma$ whenever $s \to t$, for all substitutions σ. It is *closed under contexts* if $C[s] \to C[t]$ whenever $s \to t$, for all contexts C. Moreover, the relation $\to$ is said to be a *rewrite relation* if it is closed under contexts and substitutions.

A *rewrite rule* $\ell \to r$ consists of two terms ℓ and r of the same sort such that all variables in r occur in ℓ. A (many-sorted) *term rewrite system* (TRS) is a set of rewrite rules. We denote by $\to_{\mathcal{R}}$ the smallest rewrite relation that contains the pairs of the TRS $\mathcal{R}$. A rule $\ell \to r$ is *non-collapsing* if r is not a variable. A TRS is called *non-collapsing* if all rules are non-collapsing. Let $\mathcal{F}_{\mathsf{AC}}$ be a subset of the binary function symbols in $\mathcal{F}$ that have sort declarations of the form $S \times S \to S$. We denote by AC the set of equations

$$f(f(x,y),z) \approx f(x,f(y,z)) \qquad\qquad f(x,y) \approx f(y,x)$$

expressing the associativity and commutativity of each $f \in \mathcal{F}_{\mathsf{AC}}$. The relation $\to_{\mathsf{AC}}$ is defined as expected and its reflexive, transitive, and symmetric closure is denoted by $=_{\mathsf{AC}}$. Let $\mathcal{R}$ be a TRS. The relation $=_{\mathsf{AC}} \cdot \to_{\mathcal{R}} \cdot =_{\mathsf{AC}}$ is called AC rewriting and abbreviated by $\to_{\mathcal{R}/\mathsf{AC}}$. We say that $\mathcal{R}$ is *AC terminating* if $\to_{\mathcal{R}/\mathsf{AC}}$ is well-founded. A rewrite relation is a *reduction order* if it is a well-founded order. A reduction order $>$ is *AC-compatible* if the inclusion $=_{\mathsf{AC}} \cdot > \cdot =_{\mathsf{AC}} \subseteq >$ holds. AC termination of a TRS $\mathcal{R}$ can be shown by finding an AC-compatible reduction order such that $\mathcal{R} \subseteq >$ holds.

The above definitions specialize to the usual unsorted setting when the set of sorts is a singleton set.

Finally, we recall two order extensions. Let $>$ be a strict order on a set A. The *lexicographic extension* $>^{\mathsf{lex}}$ of $>$ is defined on tuples over A as follows: $(a_1, \dots, a_m) >^{\mathsf{lex}} (b_1, \dots, b_n)$ if $n = m$ and there exists an index $1 \leqslant k \leqslant n$ such that $a_k > b_k$ and $a_i = b_i$ for all $i < k$. The *multiset extension* $>^{\mathsf{mul}}$ of $>$ is defined on multisets over A as follows: $M >^{\mathsf{mul}} N$ if there exist multisets X and Y such that $M = (N - X) \uplus Y$, $\varnothing \neq X \subseteq M$, and every $b \in Y$ admits an element $a \in X$ with $a > b$.

3 Encoding

▶ **Definition 1.** *To represent Hydras, we use a signature containing a constant symbol* h *representing a head, a binary symbol* $\mid$ *for siblings, and a unary function symbol* i *representing the internal nodes. We use infix notation for* $\mid$ *and declare it to be an* AC *symbol.*

▶ **Example 2.** The Hydras in the above example fight are represented by the terms

$$\begin{aligned}
H_0 &= \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{i}(\mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{i}(\mathsf{h}))) \mid \mathsf{h})\\
H_1 &= \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{i}(\mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{h} \mid \mathsf{h})) \mid \mathsf{h})\\
H_2 &= \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{i}(\mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{h}) \mid \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{h}) \mid \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{h})) \mid \mathsf{h})\\
H_3 &= \mathsf{i}(\mathsf{h} \mid \mathsf{h} \mid \mathsf{h} \mid \mathsf{i}(\mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{h}) \mid \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{h}) \mid \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{h})) \mid \mathsf{h} \mid \mathsf{h})\\
H_4 &= \mathsf{i}(\mathsf{h} \mid \mathsf{h} \mid \mathsf{i}(\mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{h}) \mid \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{h}) \mid \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{h})) \mid \mathsf{h} \mid \mathsf{h})
\end{aligned}$$

⌟

▶ **Definition 3.** *The* TRS $\mathcal{H}$ *consists of the following 14 rewrite rules:*

$$\begin{array}{rclrcl}
\mathsf{A}(n, \mathsf{i}(\mathsf{h})) & \xrightarrow{1} & \mathsf{A}(\mathsf{s}(n), \mathsf{h}) & \mathsf{D}(n, \mathsf{i}(\mathsf{i}(x))) & \xrightarrow{8} & \mathsf{i}(\mathsf{D}(n, \mathsf{i}(x)))\\
\mathsf{A}(n, \mathsf{i}(\mathsf{h} \mid x)) & \xrightarrow{2} & \mathsf{A}(\mathsf{s}(n), \mathsf{i}(x)) & \mathsf{D}(n, \mathsf{i}(\mathsf{i}(x) \mid y)) & \xrightarrow{9} & \mathsf{i}(\mathsf{D}(n, \mathsf{i}(x)) \mid y)\\
\mathsf{A}(n, \mathsf{i}(x)) & \xrightarrow{3} & \mathsf{B}(n, \mathsf{D}(\mathsf{s}(n), \mathsf{i}(x))) & \mathsf{D}(n, \mathsf{i}(\mathsf{i}(\mathsf{h} \mid x) \mid y)) & \xrightarrow{10} & \mathsf{i}(\mathsf{C}(n, \mathsf{i}(x)) \mid y)\\
\mathsf{C}(0, x) & \xrightarrow{4} & \mathsf{E}(x) & \mathsf{D}(n, \mathsf{i}(\mathsf{i}(\mathsf{h} \mid x))) & \xrightarrow{11} & \mathsf{i}(\mathsf{C}(n, \mathsf{i}(x)))\\
\mathsf{C}(\mathsf{s}(n), x) & \xrightarrow{5} & x \mid \mathsf{C}(n, x) & \mathsf{D}(n, \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid y)) & \xrightarrow{12} & \mathsf{i}(\mathsf{C}(n, \mathsf{h}) \mid y)\\
\mathsf{i}(\mathsf{E}(x) \mid y) & \xrightarrow{6} & \mathsf{E}(\mathsf{i}(x \mid y)) & \mathsf{D}(n, \mathsf{i}(\mathsf{i}(\mathsf{h}))) & \xrightarrow{13} & \mathsf{i}(\mathsf{C}(n, \mathsf{h}))\\
\mathsf{i}(\mathsf{E}(x)) & \xrightarrow{7} & \mathsf{E}(\mathsf{i}(x)) & \mathsf{B}(n, \mathsf{E}(x)) & \xrightarrow{14} & \mathsf{A}(\mathsf{s}(n), x)
\end{array}$$

⌟

The Battle is started with the term $\mathsf{A}(0, t)$ where t is the term representation of the initial Hydra. Rule 1 takes care of the dying Hydra ○—●. Rule 2 cuts a head without grandparent node, and so no copying takes place. Due to the power of AC matching, the removed head need not be the leftmost one. With rule 3, the search for locating a head with grandparent node starts. The search is performed with the auxiliary symbol D and involves rules 8–13. When the head to be cut is located (in rules 10–13), copying begins with the auxiliary symbol C and rules 4 and 5. The end of the copying phase is signaled with E, which travels upwards with rules 6 and 7. Finally, rule 14 creates the next stage of the Battle. Note that we make extensive use of AC matching to simplify the search process.

▶ **Theorem 4.** *If* H *and* H' *are the encodings in* $\mathcal{T}(\{\mathsf{h}, \mathsf{i}, \mid\})$ *of successive Hydras in an arbitrary battle then* $\mathsf{A}(n, H) \rightarrow^{+}_{\mathcal{H}/\mathsf{AC}} \mathsf{A}(\mathsf{s}(n), H')$ *for some* $n \in \mathcal{T}(\{0, \mathsf{s}\})$.

Before presenting the proof, we illustrate how the rewrite rules transform H_0 to H_1 in Example 2.

▶ **Example 5.** The following sequence simulates the first step in the example fight:

$$\begin{aligned}
\mathsf{A}(0, H_0) &\xrightarrow{3} \mathsf{B}(0, \mathsf{D}(\mathsf{s}(0), H_0))\\
&=_{\mathsf{AC}} \cdot \xrightarrow{9} \mathsf{B}(0, \mathsf{i}(\mathsf{D}(\mathsf{s}(0), \mathsf{i}(\mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{i}(\mathsf{h})))) \mid \mathsf{i}(\mathsf{h}) \mid \mathsf{h}))\\
&\xrightarrow{8} \mathsf{B}(0, \mathsf{i}(\mathsf{i}(\mathsf{D}(\mathsf{s}(0), \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{i}(\mathsf{h})))) \mid \mathsf{i}(\mathsf{h}) \mid \mathsf{h}))
\end{aligned}$$

$$\begin{aligned}
&\xrightarrow{12} \mathsf{B}(\mathsf{0}, \mathsf{i}(\mathsf{i}(\mathsf{i}(\mathsf{C}(\mathsf{s}(\mathsf{0}), \mathsf{h}) \mid \mathsf{i}(\mathsf{h}))) \mid \mathsf{i}(\mathsf{h}) \mid \mathsf{h}))\\
&\xrightarrow{5} \mathsf{B}(\mathsf{0}, \mathsf{i}(\mathsf{i}(\mathsf{i}(\mathsf{h} \mid \mathsf{C}(\mathsf{0}, \mathsf{h}) \mid \mathsf{i}(\mathsf{h}))) \mid \mathsf{i}(\mathsf{h}) \mid \mathsf{h}))\\
&\xrightarrow{4} \mathsf{B}(\mathsf{0}, \mathsf{i}(\mathsf{i}(\mathsf{i}(\mathsf{h} \mid \mathsf{E}(\mathsf{h}) \mid \mathsf{i}(\mathsf{h}))) \mid \mathsf{i}(\mathsf{h}) \mid \mathsf{h}))\\
=_{\mathsf{AC}} \cdot &\xrightarrow{6} \mathsf{B}(\mathsf{0}, \mathsf{i}(\mathsf{i}(\mathsf{E}(\mathsf{i}(\mathsf{h} \mid \mathsf{h} \mid \mathsf{i}(\mathsf{h})))) \mid \mathsf{i}(\mathsf{h}) \mid \mathsf{h}))\\
&\xrightarrow{7} \mathsf{B}(\mathsf{0}, \mathsf{i}(\mathsf{E}(\mathsf{i}(\mathsf{i}(\mathsf{h} \mid \mathsf{h} \mid \mathsf{i}(\mathsf{h})))) \mid \mathsf{i}(\mathsf{h}) \mid \mathsf{h}))\\
&\xrightarrow{6} \mathsf{B}(\mathsf{0}, \mathsf{E}(\mathsf{i}(\mathsf{i}(\mathsf{i}(\mathsf{h} \mid \mathsf{h} \mid \mathsf{i}(\mathsf{h}))) \mid \mathsf{i}(\mathsf{h}) \mid \mathsf{h})))\\
&\xrightarrow{14} \mathsf{A}(\mathsf{s}(\mathsf{0}), \mathsf{i}(\mathsf{i}(\mathsf{i}(\mathsf{h} \mid \mathsf{h} \mid \mathsf{i}(\mathsf{h}))) \mid \mathsf{i}(\mathsf{h}) \mid \mathsf{h})) =_{\mathsf{AC}} \mathsf{A}(\mathsf{s}(\mathsf{0}), H_1)
\end{aligned} \quad ⌟$$

It is important to note that the TRS $\mathcal{H}$ defined above is *unsorted* and we establish in this paper the result that it is AC terminating on all terms. When simulating a battle, like in the statement of the Theorem 4, we deal with well-behaved terms adhering to the sort discipline introduced shortly. The restriction to sorted terms is crucial for our termination proof, but entails no loss of generality. This is due to the following result, which is a special case of [15, Corollary 3.9].

▶ **Theorem 6.** *A non-collapsing* TRS *over a many-sorted signature is* AC *terminating if and only if the corresponding* TRS *over the unsorted version of the signature is* AC *terminating.*

The idea of using sorts to simplify termination proof goes back to Zantema [25]. The TRS $\mathcal{H}$ can be seen as a TRS over the many-sorted signature $\mathcal{F}'$:

$$\begin{array}{llll}
\mathsf{h} : \mathsf{O} & \mathsf{i}, \mathsf{E} : \mathsf{O} \to \mathsf{O} & \mid\, : \mathsf{O} \times \mathsf{O} \to \mathsf{O} & \mathsf{A}, \mathsf{B} : \mathsf{N} \times \mathsf{O} \to \mathsf{S}\\
\mathsf{0} : \mathsf{N} & \quad \mathsf{s} : \mathsf{N} \to \mathsf{N} & & \mathsf{C}, \mathsf{D} : \mathsf{N} \times \mathsf{O} \to \mathsf{O}
\end{array}$$

where N, O and S are sort symbols. Since $\mathcal{H}$ is non-collapsing, Theorem 6 guarantees that AC termination of $\mathcal{H}$ follows from AC termination of well-sorted terms over $\mathcal{F}'$.

In the remainder of this section we present a proof of Theorem 4 and its converse.

▶ **Definition 7.** *Let n be a natural number. The* TRS $\mathcal{R}_n$ *operates on encodings of Hydras and consists of the following four rules:*

$$\begin{array}{rlrl}
\mathsf{i}(\mathsf{i}(\mathsf{h})) & \xrightarrow{1} \mathsf{i}(\mathsf{h}^{n+2}) & \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid y) & \xrightarrow{3} \mathsf{i}(\mathsf{h}^{n+2} \mid y)\\
\mathsf{i}(\mathsf{i}(\mathsf{h} \mid x)) & \xrightarrow{2} \mathsf{i}(\mathsf{i}(x)^{n+2}) & \mathsf{i}(\mathsf{i}(\mathsf{h} \mid x) \mid y) & \xrightarrow{4} \mathsf{i}(\mathsf{i}(x)^{n+2} \mid y)
\end{array}$$

Here t^k for $k \geqslant 1$ is defined inductively as follows:

$$t^k = \begin{cases} t & \textit{if } k = 1\\ t^{k-1} \mid t & \textit{if } k > 1 \end{cases}$$

The following lemma relates successive Hydras in a battle to the rules in Definition 7. The easy proof is omitted.

▶ **Lemma 8.** *If H and H' be the encodings of Hydras at stages n and $n+1$ in a battle then*

1. *$H = \mathsf{i}(\mathsf{h})$ and $H' = \mathsf{h}$, or*
2. *$H =_{\mathsf{AC}} \mathsf{i}(\mathsf{h} \mid t)$ and $H' = \mathsf{i}(t)$ for some term t, or*
3. *$H \to_{\mathcal{R}_n/\mathsf{AC}} H'$.*

In the following we write n for $\mathsf{s}^n(\mathsf{0})$. Moreover, $\mathcal{T}_{\mathcal{H}}$ denotes the set of ground terms over $\{\mathsf{h}, \mathsf{i}, \mid\}$, and $\mathcal{C}_{\mathcal{H}}$ denotes the set of ground contexts over $\{\mathsf{h}, \mathsf{i}, \mid\}$.

▶ **Lemma 9.** *If $n > 0$ then $\mathsf{C}(\mathsf{n},t) \to^*_{\mathcal{H}/\mathsf{AC}} t^n \mid \mathsf{E}(t)$ for all terms t.*

Proof. We use induction on n. If $n = 1$ then

$$\mathsf{C}(\mathsf{n},t) \xrightarrow{5} t \mid \mathsf{C}(\mathsf{0},t) \xrightarrow{6} t \mid \mathsf{E}(t) = t^n \mid \mathsf{E}(t)$$

Suppose the result holds for $n \geqslant 1$ and consider $n+1$. The induction hypothesis yields $\mathsf{C}(\mathsf{n},t) \to^*_{\mathcal{H}/\mathsf{AC}} t^n \mid \mathsf{E}(t)$. Hence

$$\mathsf{C}(\mathsf{n{+}1},t) \xrightarrow{5} t \mid \mathsf{C}(\mathsf{n},t) \to^*_{\mathcal{H}/\mathsf{AC}} t \mid (t^n \mid \mathsf{E}(t)) =_{\mathsf{AC}} t^{n+1} \mid \mathsf{E}(t) \qquad ◀$$

▶ **Lemma 10.** *If $n \geqslant 0$ then $\mathsf{D}(\mathsf{n{+}1},H) \to^*_{\mathcal{H}/\mathsf{AC}} \mathsf{E}(H')$.*

Proof. We use structural induction on H and consider the following two cases.

- First suppose $H \to_{\mathcal{R}_n/\mathsf{AC}} H'$ is a root step. If the first rule of $\mathcal{R}_n$ is used then $H = \mathsf{i}(\mathsf{i}(\mathsf{h}))$ and $H' =_{\mathsf{AC}} \mathsf{i}(h^{n+2})$. We have $\mathsf{D}(\mathsf{n{+}1},H) \xrightarrow{13} \mathsf{i}(\mathsf{C}(\mathsf{n{+}1},\mathsf{h}))$. Using Lemma 9 we obtain

  $$\mathsf{i}(\mathsf{C}(\mathsf{n{+}1},\mathsf{h})) \to^*_{\mathcal{H}/\mathsf{AC}} \mathsf{i}(\mathsf{h}^{n+1} \mid \mathsf{E}(\mathsf{h})) =_{\mathsf{AC}} \cdot \xrightarrow{6} \mathsf{E}(\mathsf{i}(\mathsf{h} \mid \mathsf{h}^{n+1})) =_{\mathsf{AC}} \mathsf{E}(H')$$

  If the second rule of $\mathcal{R}_n$ is used then $H =_{\mathsf{AC}} \mathsf{i}(\mathsf{i}(\mathsf{h} \mid t))$ and $H' =_{\mathsf{AC}} \mathsf{i}(\mathsf{i}(t)^{n+2})$ for some term t. We have $\mathsf{D}(\mathsf{n{+}1},H) =_{\mathsf{AC}} \cdot \xrightarrow{11} \mathsf{i}(\mathsf{C}(\mathsf{n{+}1},\mathsf{i}(t)))$. Using Lemma 9 we obtain

  $$\mathsf{i}(\mathsf{C}(\mathsf{n{+}1},\mathsf{i}(t))) \to^*_{\mathcal{H}/\mathsf{AC}} \mathsf{i}(\mathsf{i}(t)^{n+1} \mid \mathsf{E}(\mathsf{i}(t))) =_{\mathsf{AC}} \cdot \xrightarrow{6} \mathsf{E}(\mathsf{i}(\mathsf{i}(t) \mid \mathsf{i}(t)^{n+1})) =_{\mathsf{AC}} \mathsf{E}(H')$$

  If the third rule of $\mathcal{R}_n$ is used then $H =_{\mathsf{AC}} \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid t)$ and $H' =_{\mathsf{AC}} \mathsf{i}(\mathsf{h}^{n+2} \mid t)$ for some term t. We have $\mathsf{D}(\mathsf{n{+}1},H) =_{\mathsf{AC}} \cdot \xrightarrow{12} \mathsf{i}(\mathsf{C}(\mathsf{n{+}1},\mathsf{h}) \mid t)$. The remaining argument is the same as in the preceding cases. If the fourth rule of $\mathcal{R}_n$ is used then $H =_{\mathsf{AC}} \mathsf{i}(\mathsf{i}(\mathsf{h} \mid s) \mid t)$ and $H' =_{\mathsf{AC}} \mathsf{i}(\mathsf{i}(s)^{n+2} \mid t)$ for some terms s and t. Using Lemma 9 we obtain

  $$\begin{aligned}\mathsf{D}(\mathsf{n{+}1},H) &=_{\mathsf{AC}} \cdot \xrightarrow{10} \mathsf{i}(\mathsf{C}(\mathsf{n{+}1},\mathsf{i}(s)) \mid t) \to^*_{\mathcal{H}/\mathsf{AC}} \mathsf{i}((\mathsf{i}(s)^{n+1} \mid \mathsf{E}(\mathsf{i}(s))) \mid t)\\ &=_{\mathsf{AC}} \cdot \xrightarrow{6} \mathsf{E}(\mathsf{i}(\mathsf{i}(s) \mid (\mathsf{i}(s)^{n+1} \mid t))) =_{\mathsf{AC}} \mathsf{E}(H')\end{aligned}$$

- Otherwise, $H =_{\mathsf{AC}} \mathsf{i}(H_1 \mid H_2 \mid \cdots \mid H_m)$ and $H' =_{\mathsf{AC}} \mathsf{i}(H_1' \mid H_2 \mid \cdots \mid H_m)$ for some $m \geqslant 1$ and Hydras $H_1, \dots, H_m, H_1'$ with $H_1 \to_{\mathcal{R}_n/\mathsf{AC}} H_1'$. We obtain $\mathsf{D}(\mathsf{n{+}1},H_1) \to^*_{\mathcal{H}/\mathsf{AC}} \mathsf{E}(H_1')$ from the induction hypothesis. Note that $\mathsf{root}(H_1) = \mathsf{i}$. If $m = 1$ then

  $$\begin{aligned}\mathsf{D}(\mathsf{n{+}1},H) &=_{\mathsf{AC}} \mathsf{D}(\mathsf{n{+}1},\mathsf{i}(H_1)) \xrightarrow{8} \mathsf{i}(\mathsf{D}(\mathsf{n{+}1},H_1)) \to^*_{\mathcal{H}/\mathsf{AC}} \mathsf{i}(\mathsf{E}(H_1')) \xrightarrow{7} \mathsf{E}(\mathsf{i}(H_1'))\\ &=_{\mathsf{AC}} \mathsf{E}(H_1)\end{aligned}$$

  and if $m > 1$ we reach the same conclusion using rules 9 and 6 instead of 8 and 7. ◀

Proof of Theorem 4. Our task is to show

$$\mathsf{A}(\mathsf{n},H) \to^*_{\mathcal{H}/\mathsf{AC}} \mathsf{A}(\mathsf{n{+}1},H')$$

For the first two cases in Lemma 8 the claim is immediate by rules 1 and 2 of $\mathcal{H}$. To verify the third case, assume $H \to_{\mathcal{R}_n/\mathsf{AC}} H'$. This implies $\mathsf{root}(H) = \mathsf{i}$. Using rules 3 and 14 together with Lemma 10 yields

$$\mathsf{A}(\mathsf{n},H) \xrightarrow{3} \mathsf{B}(\mathsf{n},\mathsf{D}(\mathsf{n{+}1},H)) \to^*_{\mathcal{H}/\mathsf{AC}} \mathsf{B}(\mathsf{n},\mathsf{E}(H')) \xrightarrow{14} \mathsf{A}(\mathsf{n{+}1},H') \qquad ◀$$

In the remaining part of this section we prove the converse of Theorem 4.

▶ **Theorem 11.** *Let $H, H' \in \mathcal{T}_{\mathcal{H}}$ be encodings of Hydras and let n be a natural number. If $\mathsf{A}(\mathsf{n}, H) \to^*_{\mathcal{H}/\mathsf{AC}} \mathsf{A}(\mathsf{n{+}1}, H')$ then H and H' are successive Hydras in a battle.*

In order to show the claim we need a few auxiliary lemmata.

▶ **Definition 12.** *We define U as the set consisting of all terms of the forms $\mathsf{A}(\mathsf{n}, t)$, $\mathsf{B}(\mathsf{n}, C[\mathsf{C}(\mathsf{m}, t)])$, $\mathsf{B}(\mathsf{n}, C[\mathsf{D}(\mathsf{n{+}1}, t)])$, and $\mathsf{B}(\mathsf{n}, C[\mathsf{E}(t)])$, where $n, m \in \mathbb{N}$, $t \in \mathcal{T}_{\mathcal{H}}$, and $C \in \mathcal{C}_{\mathcal{H}}$.*

The set U contains all terms reachable from $\mathsf{A}(\mathsf{n}, H)$.

▶ **Lemma 13.** *If $t \in U$ and $t \to^*_{\mathcal{H} \cup \mathsf{AC}} u$ then $u \in U$.*

In order to analyze the rewrite sequence $\mathsf{A}(\mathsf{n}, H) \to^*_{\mathcal{H}/\mathsf{AC}} \mathsf{A}(\mathsf{n{+}1}, H')$ we define three subsets of $\mathcal{H}$: $\mathcal{H}_1 = \{1, 2\}$, $\mathcal{H}_2 = \{3, 4, 5, 6, 7, 8, 9, 14\}$, and $\mathcal{H}_3 = \{10, 11, 12, 13\}$. The rewrite sequence in Example 5 can then be described as follows:

$$\begin{aligned}\mathsf{A}(\mathsf{0}, H_0) &\to^*_{\mathcal{H}_2/\mathsf{AC}} \mathsf{B}(\mathsf{0}, \mathsf{i}(\mathsf{i}(\mathsf{D}(\mathsf{s}(\mathsf{0}), \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{i}(\mathsf{h})))) \mid \mathsf{i}(\mathsf{h}) \mid \mathsf{h}))\\ &\to_{\mathcal{H}_3/\mathsf{AC}} \mathsf{B}(\mathsf{0}, \mathsf{i}(\mathsf{i}(\mathsf{i}(\mathsf{C}(\mathsf{s}(\mathsf{0}), \mathsf{h}) \mid \mathsf{i}(\mathsf{h}))) \mid \mathsf{i}(\mathsf{h}) \mid \mathsf{h}))\\ &\to^*_{\mathcal{H}_2/\mathsf{AC}} \mathsf{A}(\mathsf{1}, H_1)\end{aligned}$$

▶ **Definition 14.** *We define V as the extension of U with $\mathcal{T}_{\mathcal{H}}$ and all terms of the forms $C[\mathsf{C}(\mathsf{n}, t)]$ $C[\mathsf{D}(\mathsf{n}, t)]$, and $C[\mathsf{E}(t)]$ where $n \in \mathbb{N}$, $t \in \mathcal{T}_{\mathcal{H}}$, and $C \in \mathcal{C}_{\mathcal{H}}$. The mapping $\pi \colon V \to \mathcal{T}_{\mathcal{H}}$ is defined as follows:*

$$\pi(t) = \begin{cases} \mathsf{h} & \text{if } t = \mathsf{h}\\ \mathsf{i}(\pi(u)) & \text{if } t = \mathsf{i}(u)\\ \pi(u) \mid \pi(v) & \text{if } t = u \mid v\\ u & \text{if } t = \mathsf{A}(\mathsf{n}, u) \text{ or } t = \mathsf{D}(\mathsf{n}, u) \text{ or } t = \mathsf{E}(u)\\ \pi(u) & \text{if } t = \mathsf{B}(\mathsf{n}, u)\\ u^{n+1} & \text{if } t = \mathsf{C}(\mathsf{n}, u)\end{cases}$$

Taking the role of C into account, the mapping π computes the Hydra in a given term. Applying π to the terms in the above rewrite sequence of $\mathcal{H}_2/\mathsf{AC}$ and $\mathcal{H}_3/\mathsf{AC}$, we obtain

$$\begin{aligned}H_0 = \pi(\mathsf{A}(\mathsf{0}, H_0)) &=_{\mathsf{AC}} \pi(\mathsf{B}(\mathsf{0}, \mathsf{i}(\mathsf{i}(\mathsf{D}(\mathsf{s}(\mathsf{0}), \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid \mathsf{i}(\mathsf{h})))) \mid \mathsf{i}(\mathsf{h}) \mid \mathsf{h})))\\ &\to_{\mathcal{R}_0/\mathsf{AC}} \pi(\mathsf{B}(\mathsf{0}, \mathsf{i}(\mathsf{i}(\mathsf{i}(\mathsf{C}(\mathsf{s}(\mathsf{0}), \mathsf{h}) \mid \mathsf{i}(\mathsf{h}))) \mid \mathsf{i}(\mathsf{h}) \mid \mathsf{h})))\\ &=_{\mathsf{AC}} \pi(\mathsf{A}(\mathsf{1}, H_1)) = H_1\end{aligned}$$

This verifies that H_1 is a successor of H_0.

▶ **Lemma 15.** *The following properties hold.*

1. *$\pi(t) = t$ for all terms $t \in \mathcal{T}_{\mathcal{H}}$,*
2. *$\pi(C[t]) = C[\pi(t)]$ for all terms $t \in V$ and contexts $C \in \mathcal{C}_{\mathcal{H}}$,*
3. *$\pi(C[t]) =_{\mathsf{AC}} \pi(D[u])$ for all terms $t, u \in \mathcal{T}_{\mathcal{H}}$ and contexts $C, D \in \mathcal{C}_{\mathcal{H}}$ with $t =_{\mathsf{AC}} u$ and $C =_{\mathsf{AC}} D$.*

Proof. The first statement is proved by induction on $t \in \mathcal{T}_{\mathcal{H}}$. If $t = \mathsf{h}$ then $\pi(t) = \mathsf{h} = t$. If $t = \mathsf{i}(u)$ with $u \in \mathcal{T}_{\mathcal{H}}$ then $\pi(t) = \mathsf{i}(\pi(u)) = \mathsf{i}(u) = t$. If $t = u \mid v$ with $u, v \in \mathcal{T}_{\mathcal{H}}$ then $\pi(t) = \pi(u) \mid \pi(v) = u \mid v = t$. For the second statement we use induction on the context $C \in \mathcal{C}_{\mathcal{H}}$. If $C = \square$ then $\pi(C[t]) = \pi(t) = C[\pi(t)]$. If $C = \mathsf{i}(D)$ then $\pi(C[t]) = \mathsf{i}(\pi(D[t])) = \mathsf{i}(D[\pi(t)]) = C[\pi(t)]$. If $C = D \mid u$ then $D \in \mathcal{C}_{\mathcal{H}}$ and $u \in \mathcal{T}_{\mathcal{H}}$ and thus $\pi(C[t]) = \pi(D[t]) | \pi(u) = D[\pi(t)] | u = C[\pi(t)]$. If $C = u | D$ then $D \in \mathcal{C}_{\mathcal{H}}$ and $u \in \mathcal{T}_{\mathcal{H}}$ and thus $\pi(C[t]) = \pi(u) \mid \pi(D[t]) = u \mid D[\pi(t)] = C[\pi(t)]$. The third statement follows from statements (1) and (2): $\pi(C[t]) = C[\pi(t)] = C[t] =_{\mathsf{AC}} D[t] =_{\mathsf{AC}} D[u] = D[\pi(u)] = \pi(D[u])$. ◀

The following lemma relates AC rewriting of $\mathcal{H}$ to rewriting of Hydras according to Definition 7.

▶ **Lemma 16.** *The following statements hold for all terms $s \in U$.*

1. *If $s =_{\mathsf{AC}} t$ then $\pi(s) =_{\mathsf{AC}} \pi(t)$.*
2. *If $s \to_{\mathcal{H}_2} t$ then $\pi(s) =_{\mathsf{AC}} \pi(t)$.*
3. *If $s \to_{\mathcal{H}_3} t$ then $\pi(s) \to_{\mathcal{R}_n/\mathsf{AC}} \pi(t)$ with $s = \mathsf{B}(\mathsf{n}, s')$ for some $n \geqslant 0$.*

Proof. Let $s \in U$.

1. If $s = \mathsf{A}(\mathsf{n}, u)$ with $u \in \mathcal{T}_{\mathcal{H}}$ then $t = \mathsf{A}(\mathsf{n}, v)$ for some term $v \in \mathcal{T}_{\mathcal{H}}$ with $u =_{\mathsf{AC}} v$. Since $\pi(s) = u$ and $\pi(t) = v$, $\pi(s) =_{\mathsf{AC}} \pi(t)$ follows. If $s = \mathsf{B}(\mathsf{n}, C[\mathsf{C}(\mathsf{m}, u)])$ with $n, m \in \mathbb{N}$, $C \in \mathcal{C}_{\mathcal{H}}$ and $u \in \mathcal{T}_{\mathcal{H}}$ then $t = \mathsf{B}(\mathsf{n}, D[\mathsf{C}(\mathsf{m}, v)])$ with $C =_{\mathsf{AC}} D$ and $u =_{\mathsf{AC}} v$. Using Lemma 15(1,2) we obtain $\pi(s) = \pi(C[\mathsf{C}(\mathsf{m}, u)]) = C[\pi(\mathsf{C}(\mathsf{m}, u))] = C[u^{m+1}]$ and $\pi(t) = D[v^{m+1}]$. From $u =_{\mathsf{AC}} v$ we infer $u^{m+1} =_{\mathsf{AC}} v^{m+1}$ and thus $\pi(s) =_{\mathsf{AC}} \pi(t)$ by Lemma 15(3). The cases $s = \mathsf{B}(\mathsf{n}, C[\mathsf{D}(\mathsf{n{+}1}, u)])$ and $s = \mathsf{B}(\mathsf{n}, C[\mathsf{E}(u)])$ are treated in the same way.
2. For the second statement we make a case analysis based on the employed rule in $\mathcal{H}_2$.
   - If $s \xrightarrow{3} t$ then $s = \mathsf{A}(\mathsf{n}, \mathsf{i}(u))$ and $t = \mathsf{B}(\mathsf{n}, \mathsf{D}(\mathsf{n{+}1}, \mathsf{i}(u)))$ for some $n \geqslant 0$ and $u \in \mathcal{T}_{\mathcal{H}}$. We have $\pi(s) = \mathsf{i}(u) = \pi(\mathsf{D}(\mathsf{n{+}1}, \mathsf{i}(u))) = \pi(t)$ by the definition of t.
   - If $s \xrightarrow{4} t$ then $s = \mathsf{B}(\mathsf{n}, C[\mathsf{C}(\mathsf{0}, u)])$ and $t = \mathsf{B}(\mathsf{n}, C[\mathsf{E}(u)])$ for some $n \geqslant 0$, $C \in \mathcal{C}_{\mathcal{H}}$ and $u \in \mathcal{T}_{\mathcal{H}}$. We have $\pi(s) = \pi(C[\mathsf{C}(\mathsf{0}, u)]) = C[u^1] = C[u] = \pi(C[u]) = \pi(t)$.
   - If $s \xrightarrow{5} t$ then $s = \mathsf{B}(\mathsf{n}, C[\mathsf{C}(\mathsf{m}, u)])$ and $t = \mathsf{B}(\mathsf{n}, C[u \mid \mathsf{C}(\mathsf{m{-}1}, u)])$ for some $n \geqslant 0$, $m > 0$, $C \in \mathcal{C}_{\mathcal{H}}$ and $u \in \mathcal{T}_{\mathcal{H}}$. We have $\pi(s) = C[u^{m+1}] =_{\mathsf{AC}} C[u \mid u^m] = C[\pi(u \mid u^m)] = \pi(C[u \mid \mathsf{C}(\mathsf{m{-}1}, u)]) = \pi(t)$.
   - If $s \xrightarrow{6} t$ then $s = \mathsf{B}(\mathsf{n}, C[\mathsf{i}(\mathsf{E}(u) | v)])$ and $t = \mathsf{B}(\mathsf{n}, C[\mathsf{E}(\mathsf{i}(u | v))])$ for some $n \geqslant 0$, $C \in \mathcal{C}_{\mathcal{H}}$ and $u, v \in \mathcal{T}_{\mathcal{H}}$. We have $\pi(s) = \pi(C[\mathsf{i}(\mathsf{E}(u) \mid v)]) = C[\mathsf{i}(u \mid v)] = \pi(C[\mathsf{E}(\mathsf{i}(u \mid v))]) = \pi(t)$.
   - If $s \xrightarrow{7} t$ then $s = \mathsf{B}(\mathsf{n}, C[\mathsf{i}(\mathsf{E}(u))])$ and $t = \mathsf{B}(\mathsf{n}, C[\mathsf{E}(\mathsf{i}(u))])$ for some $n \geqslant 0$, $C \in \mathcal{C}_{\mathcal{H}}$ and $u \in \mathcal{T}_{\mathcal{H}}$. We have $\pi(s) = \pi(C[\mathsf{i}(\mathsf{E}(u))]) = C[\mathsf{i}(u)] = \pi(C[\mathsf{E}(\mathsf{i}(u))]) = \pi(t)$.
   - If $s \xrightarrow{8} t$ then $s = \mathsf{B}(\mathsf{n}, C[\mathsf{D}(\mathsf{n{+}1}, \mathsf{i}(\mathsf{i}(u)))])$ and $t = \mathsf{B}(\mathsf{n}, C[\mathsf{i}(\mathsf{D}(\mathsf{n{+}1}, \mathsf{i}(u)))])$ for some $n \geqslant 0$, $C \in \mathcal{C}_{\mathcal{H}}$ and $u \in \mathcal{T}_{\mathcal{H}}$. We have $\pi(s) = C[\mathsf{i}(\mathsf{i}(u))] = \pi(t)$.
   - If $s \xrightarrow{9} t$ then $s = \mathsf{B}(\mathsf{n}, C[\mathsf{D}(\mathsf{n{+}1}, \mathsf{i}(\mathsf{i}(u) \mid v))])$ and $t = \mathsf{B}(\mathsf{n}, C[\mathsf{i}(\mathsf{D}(\mathsf{n{+}1}, \mathsf{i}(u)) \mid v)])$ for some $n \geqslant 0$, $C \in \mathcal{C}_{\mathcal{H}}$ and $u, v \in \mathcal{T}_{\mathcal{H}}$. In this case we obtain $\pi(s) = C[\mathsf{i}(\mathsf{i}(u) \mid v)] = \pi(t)$.
   - If $s \xrightarrow{14} t$ then $s = \mathsf{B}(\mathsf{n}, \mathsf{E}(u))$ and $t = \mathsf{A}(\mathsf{n{+}1}, u)$ for some $n \geqslant 0$ and $u \in \mathcal{T}_{\mathcal{H}}$. In this case we have $\pi(s) = \pi(\mathsf{E}(u)) = u = \pi(t)$.
3. Again we make a case analysis on the applied rewrite rule.
   - If $s \xrightarrow{10} t$ then $s = \mathsf{B}(\mathsf{n}, C[\mathsf{D}(\mathsf{n{+}1}, \mathsf{i}(\mathsf{i}(\mathsf{h} \mid u) \mid v))])$ and $t = \mathsf{B}(\mathsf{n}, C[\mathsf{i}(\mathsf{C}(\mathsf{n{+}1}, \mathsf{i}(u)) \mid v)])$ for some $n \geqslant 0$, $C \in \mathcal{C}_{\mathcal{H}}$ and $u, v \in \mathcal{T}_{\mathcal{H}}$. We obtain $\pi(s) = C[\mathsf{i}(\mathsf{i}(\mathsf{h} \mid u) \mid v)]$ and $\pi(t) = C[\mathsf{i}(\mathsf{i}(u)^{n+2} \mid v)]$. Hence $\pi(s) \to_{\mathcal{R}_n} \pi(t)$ by applying rule 4 of $\mathcal{R}_n$.
   - If $s \xrightarrow{11} t$ then $s = \mathsf{B}(\mathsf{n}, C[\mathsf{D}(\mathsf{n{+}1}, \mathsf{i}(\mathsf{i}(\mathsf{h} \mid u)))])$ and $t = \mathsf{B}(\mathsf{n}, C[\mathsf{i}(\mathsf{C}(\mathsf{n{+}1}, \mathsf{i}(u)))])$ for some $n \geqslant 0$, $C \in \mathcal{C}_{\mathcal{H}}$ and $u, v \in \mathcal{T}_{\mathcal{H}}$. We obtain $\pi(s) = C[\mathsf{i}(\mathsf{i}(\mathsf{h} \mid u))]$ and $\pi(t) = C[\mathsf{i}(\mathsf{i}(u)^{n+2})]$. Hence $\pi(s) \to_{\mathcal{R}_n} \pi(t)$ by applying rule 2 of $\mathcal{R}_n$.
   - If $s \xrightarrow{12} t$ then $s = \mathsf{B}(\mathsf{n}, C[\mathsf{D}(\mathsf{n{+}1}, \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid v))])$ and $t = \mathsf{B}(\mathsf{n}, C[\mathsf{i}(\mathsf{C}(\mathsf{n{+}1}, \mathsf{h}) \mid v)])$ for some $n \geqslant 0$, $C \in \mathcal{C}_{\mathcal{H}}$ and $v \in \mathcal{T}_{\mathcal{H}}$. We obtain $\pi(s) = C[\mathsf{i}(\mathsf{i}(\mathsf{h}) \mid v)]$ and $\pi(t) = C[\mathsf{i}(\mathsf{h}^{n+2} \mid v)]$. Hence $\pi(s) \to_{\mathcal{R}_n} \pi(t)$ by applying rule 3 of $\mathcal{R}_n$.
   - If $s \xrightarrow{13} t$ then $s = \mathsf{B}(\mathsf{n}, C[\mathsf{D}(\mathsf{n{+}1}, \mathsf{i}(\mathsf{i}(\mathsf{h})))])$ and $t = \mathsf{B}(\mathsf{n}, C[\mathsf{i}(\mathsf{C}(\mathsf{n{+}1}, \mathsf{h}))])$ for some $n \geqslant 0$ and $C \in \mathcal{C}_{\mathcal{H}}$. We obtain $\pi(s) = C[\mathsf{i}(\mathsf{i}(\mathsf{h}))]$ and $\pi(t) = C[\mathsf{i}(\mathsf{h}^{n+2})]$. Hence $\pi(s) \to_{\mathcal{R}_n} \pi(t)$ by applying rule 1 of $\mathcal{R}_n$. ◀

So we are ready to prove the main claim.

Proof of Theorem 11. Suppose $s = \mathsf{A}(\mathsf{n}, H) \to^{+}_{\mathcal{H}/\mathsf{AC}} \mathsf{A}(\mathsf{n}{+}1, H') = t$. Inspection of $\mathcal{H}$ reveals that one of the following two cases holds:

(a) $s \to_{\mathcal{H}_1/\mathsf{AC}} t$, or

(b) $s \to^{*}_{\mathcal{H}_2/\mathsf{AC}} \cdot \to_{\mathcal{H}_3/\mathsf{AC}} \cdot \to^{*}_{\mathcal{H}_2/\mathsf{AC}} t$.

We first consider (a). If $s \to_{\mathcal{H}_1/\mathsf{AC}} t$ is a root step using rule 1 then $H = \mathsf{i}(\mathsf{h})$ and $H' = \mathsf{h}$. If $s \to_{\mathcal{H}_1/\mathsf{AC}} t$ is a root step using rule 2 then $H =_{\mathsf{AC}} \mathsf{i}(\mathsf{h} \mid u)$ and $H' =_{\mathsf{AC}} \mathsf{i}(u)$ for some term u. Next we consider (b). We have $s \to^{*}_{\mathcal{H}_2/\mathsf{AC}} s' \to_{\mathcal{H}_3/\mathsf{AC}} t' \to^{*}_{\mathcal{H}_2/\mathsf{AC}} t$ for some s' and t'. From Lemma 13 we obtain $s, s', t', t \in U$. Hence

$$H = \pi(s) =_{\mathsf{AC}} \pi(s') \to_{\mathcal{R}_n/\mathsf{AC}} \pi(t') =_{\mathsf{AC}} \pi(t) = H'$$

is obtained by Lemma 16 and thus also $H \to_{\mathcal{R}_n/\mathsf{AC}} H'$. ◀

4 Many-Sorted Semantic Labeling modulo AC

The mutual dependence between the function symbols A and B in rules 3 and 14 of $\mathcal{H}$ makes proving termination of $\mathcal{H}/\mathsf{AC}$ a non-trivial task. We use the technique of semantic labeling (Zantema [26]) to resolve the dependence by labeling both A and B by the ordinal value of the Hydra encoded in their second arguments. Semantic labeling for rewriting modulo has been investigated in [19]. We need, however, a version for many-sorted rewriting since the distinction between ordinals and natural numbers is essential for the effectiveness of semantic labeling for $\mathcal{H}/\mathsf{AC}$.

Before introducing semantic labeling, we recall some basic semantic definitions. An *algebra* $\mathcal{A}$ for an $\mathcal{S}$-sorted signature $\mathcal{F}$ is a pair $(\{S_{\mathcal{A}}\}_{S \in \mathcal{S}}, \{f_{\mathcal{A}}\}_{f \in \mathcal{F}})$, where each $S_{\mathcal{A}}$ is a non-empty set, called the *carrier of sort* S, and each $f_{\mathcal{A}}$ is a function of type $f : (S_1)_{\mathcal{A}} \times \cdots \times (S_n)_{\mathcal{A}} \to S_{\mathcal{A}}$, called the *interpretation function* of $f : S_1 \times \cdots \times S_n \to S$. A mapping that associates each variable of sort S to an element in $S_{\mathcal{A}}$ is called an *assignment*. We write $\mathcal{A}^{\mathcal{V}}$ for the set of all assignments. Given an assignment $\alpha \in \mathcal{A}^{\mathcal{V}}$, the *interpretation* of a term t is inductively defined as follows:

$$[\alpha]_{\mathcal{A}}(t) = \begin{cases} \alpha(t) & \text{if } t \text{ is a variable} \\ f_{\mathcal{A}}([\alpha]_{\mathcal{A}}(t_1), \ldots, [\alpha]_{\mathcal{A}}(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

Let $\mathcal{A} = (\{S_{\mathcal{A}}\}_{S \in \mathcal{S}}, \{f_{\mathcal{A}}\}_{f \in \mathcal{F}})$ be an $\mathcal{S}$-sorted $\mathcal{F}$-algebra. We assume that each carrier set $S_{\mathcal{A}}$ is equipped with a well-founded order $>_S$ such that the interpretation functions are weakly monotone in all argument positions, and call $(\mathcal{A}, \{>_S\}_{S \in \mathcal{S}})$ a weakly monotone many-sorted algebra. Given terms s and t of sort S, we write $s \geqslant_{\mathcal{A}} t$ ($s =_{\mathcal{A}} t$) if $[\alpha]_{\mathcal{A}}(s) \geqslant_S [\alpha]_{\mathcal{A}}(t)$ ($[\alpha]_{\mathcal{A}}(s) =_S [\alpha]_{\mathcal{A}}(t)$) holds for all $\alpha \in \mathcal{A}^{\mathcal{V}}$.

A labeling L for $\mathcal{F}$ consists of sets of labels $L_f \subseteq S_{\mathcal{A}}$ for every $f : S_1 \times \cdots \times S_n \to S$. The labeled signature $\mathcal{F}_{\mathsf{lab}}$ consists of function symbols $f_a : S_1 \times \cdots \times S_n \to S$ for every function symbol $f : S_1 \times \cdots \times S_n \to S$ in $\mathcal{F}$ and label $a \in L_f$ together with all function symbols $f \in \mathcal{F}$ such that $L_f = \varnothing$. A *labeling* (L, lab) for $(\mathcal{A}, \{>_S\}_{S \in \mathcal{S}})$ consists of a labeling L for the signature $\mathcal{F}$ together with a mapping $\mathsf{lab}_f \colon (S_1)_{\mathcal{A}} \times \cdots \times (S_n)_{\mathcal{A}} \to L_f$ for every function symbol $f : S_1 \times \cdots \times S_n \to S$ in $\mathcal{F}$ with $L_f \neq \varnothing$. We call (L, lab) *weakly monotone* if all its labeling functions lab_f are weakly monotone in all coordinates. The mapping lab_f determines the label of the root symbol f of a term $f(t_1, \ldots, t_n)$, based on the values of its arguments $t_1, \ldots, t_n$. Formally, for every assignment $\alpha \in \mathcal{A}^{\mathcal{V}}$ we define a mapping lab_α inductively as follows:

$$\mathsf{lab}_\alpha(t) = \begin{cases} t & \text{if } t \in \mathcal{V} \\ f(\mathsf{lab}_\alpha(t_1), \dots, \mathsf{lab}_\alpha(t_n)) & \text{if } t = f(t_1, \dots, t_n) \text{ and } L_f = \varnothing \\ f_a(\mathsf{lab}_\alpha(t_1), \dots, \mathsf{lab}_\alpha(t_n)) & \text{if } t = f(t_1, \dots, t_n) \text{ and } L_f \neq \varnothing \end{cases}$$

where a denotes the label $\mathsf{lab}_f([\alpha]_\mathcal{A}(t_1), \dots, [\alpha]_\mathcal{A}(t_n))$. Note that $\mathsf{lab}_\alpha(t)$ and t have the same sort. Given a TRS $\mathcal{R}$ over a (many-sorted) signature $\mathcal{F}$, we define the *labeled* TRS $\mathcal{R}_\mathsf{lab}$ over the signature $\mathcal{F}_\mathsf{lab}$ as follows:

$$\mathcal{R}_\mathsf{lab} = \{\mathsf{lab}_\alpha(\ell) \to \mathsf{lab}_\alpha(r) \mid \ell \to r \in \mathcal{R} \text{ and } \alpha \in A^\mathcal{V}\}$$

Since the AC symbol $|$ in the encoding of the Hydra battle is a constructor, there is no need to label it. Hence we assume for simplicity that $L_f = \varnothing$ for every AC symbol $f \in \mathcal{F}$. The TRS $\mathcal{D}\mathsf{ec}$ consists of all rewrite rules

$$f_a(x_1, \dots, x_n) \to f_b(x_1, \dots, x_n)$$

with $f : S_1 \times \cdots \times S_n \to S$ a function symbol in $\mathcal{F}$, $a, b \in L_f$ such that $a >_S b$, and pairwise different variables $x_1, \dots, x_n$. A weakly monotone algebra $(\mathcal{A}, >)$ is a *quasi-model* of $\mathcal{R}/\mathsf{AC}$ if $\ell \geqslant_\mathcal{A} r$ for all rewrite rules $\ell \to r$ in $\mathcal{R}$ and $\ell =_\mathcal{A} r$ for all equations $\ell \approx r$ in AC.

▶ **Theorem 17.** *Let $\mathcal{R}/\mathsf{AC}$ be a* TRS *over a many-sorted signature $\mathcal{F}$, $(\mathcal{A}, \{>_S\}_{S \in \mathcal{S}})$ a quasi-model of $\mathcal{R}/\mathsf{AC}$ with a weakly monotone labeling (L, lab). If $(\mathcal{R}_\mathsf{lab} \cup \mathcal{D}\mathsf{ec})/\mathsf{AC}$ is terminating then $\mathcal{R}/\mathsf{AC}$ is terminating.*

Proof. We show

1. if $t \to_\mathcal{R} u$ then $\mathsf{lab}_\alpha(t) \to^*_{\mathcal{D}\mathsf{ec}} \cdot \to_{\mathcal{R}_\mathsf{lab}} \mathsf{lab}_\alpha(u)$
2. if $t =_\mathsf{AC} u$ then $\mathsf{lab}_\alpha(t) =_\mathsf{AC} \mathsf{lab}_\alpha(u)$

for all sorts S, terms $t, u \in \mathcal{T}_S(\mathcal{F}, \mathcal{V})$, and assignments $\alpha \in \mathcal{A}^\mathcal{V}$. First suppose $t \to_\mathcal{R} u$ is a root step using the rewrite rule $\ell \to r$. So $t = \ell\sigma$ and $u = r\sigma$ for some substitution σ. Define the assignment $\beta = [\alpha]_\mathcal{A} \circ \sigma$ and the (labeled) substitution $\tau = \mathsf{lab}_\alpha \circ \sigma$. An easy induction proof yields $\mathsf{lab}_\alpha(s\sigma) = \mathsf{lab}_\beta(s)\tau$ for all terms s. By definition $\mathsf{lab}_\beta(\ell) \to \mathsf{lab}_\beta(r) \in \mathcal{R}_\mathsf{lab}$. Hence $\mathsf{lab}_\alpha(t) = \mathsf{lab}_\beta(\ell)\tau \to_{\mathcal{R}_\mathsf{lab}} \mathsf{lab}_\beta(r)\tau = \mathsf{lab}_\alpha(u)$. Next suppose $t \to_\mathcal{R} u$ takes place below the root. So $t = f(t_1, \dots, t_i, \dots t_n)$ and $u = f(t_1, \dots, u_i, \dots t_n)$ with $t_i \to_\mathcal{R} u_i$. Let $S_1 \times \cdots \times S_n \to S$ be the sort declaration of f. The induction hypothesis yields $\mathsf{lab}_\alpha(t_i) \to^*_{\mathcal{D}\mathsf{ec}} \cdot \to_{\mathcal{R}_\mathsf{lab}} \mathsf{lab}_\alpha(u_i)$. We obtain $[\alpha]_\mathcal{A}(t_i) \geqslant_{S_i} [\alpha]_\mathcal{A}(u_i)$ from the quasi-model assumption. If $L_f = \varnothing$ then

$$\begin{aligned} \mathsf{lab}_\alpha(t) = f(\mathsf{lab}_\alpha(t_1), \dots, \mathsf{lab}_\alpha(t_i), \dots, \mathsf{lab}_\alpha(t_n)) &\to^*_{\mathcal{D}\mathsf{ec}} \cdot \to_{\mathcal{R}_\mathsf{lab}} \\ f(\mathsf{lab}_\alpha(t_1), \dots, \mathsf{lab}_\alpha(u_i), \dots, \mathsf{lab}_\alpha(t_n)) &= \mathsf{lab}_\alpha(u) \end{aligned}$$

Suppose $L_f \neq \varnothing$ and let

$$\begin{aligned} a &= \mathsf{lab}_f([\alpha]_\mathcal{A}(t_1), \dots, [\alpha]_\mathcal{A}(t_i), \dots, [\alpha]_\mathcal{A}(t_n)) \\ b &= \mathsf{lab}_f([\alpha]_\mathcal{A}(t_1), \dots, [\alpha]_\mathcal{A}(u_i), \dots, [\alpha]_\mathcal{A}(t_n)) \end{aligned}$$

We obtain $a \geqslant_S b$ from the weak monotonicity of the labeling function lab_f. Therefore, the following rewrite sequence is constructed:

$$\begin{aligned} \mathsf{lab}_\alpha(t) = f_a(\mathsf{lab}_\alpha(t_1), \dots, \mathsf{lab}_\alpha(t_i), \dots, \mathsf{lab}_\alpha(t_n)) &\to^*_{\mathcal{D}\mathsf{ec}} \\ f_b(\mathsf{lab}_\alpha(t_1), \dots, \mathsf{lab}_\alpha(t_i), \dots, \mathsf{lab}_\alpha(t_n)) &\to^*_{\mathcal{D}\mathsf{ec}} \cdot \to_{\mathcal{R}_\mathsf{lab}} \\ f_b(\mathsf{lab}_\alpha(t_1), \dots, \mathsf{lab}_\alpha(u_i), \dots, \mathsf{lab}_\alpha(t_n)) &= \mathsf{lab}_\alpha(u) \end{aligned}$$

This concludes the proof of the first statement. For the second statement we use induction on the number of applications of AC axioms in $t =_{\mathsf{AC}} u$. If this number is one, the conclusion is reached by reasoning as above (with $L_f = \varnothing$ because AC symbols are not labeled and hence the rules of $\mathcal{D}\mathsf{ec}$ do not come into play). ◀

After these preliminaries, we are ready to put many-sorted semantic labeling to the test. Consider the many-sorted algebra $\mathcal{A}$ with carriers $\mathbb{N}$ for sort N and $\mathbb{O}$, the set of ordinal numbers smaller than ϵ_0, for sorts O and S and the following interpretation functions:

$$\begin{array}{rcr} \mathsf{0}_{\mathcal{A}} = \mathsf{h}_{\mathcal{A}} = 1 & \mathsf{s}_{\mathcal{A}}(n) = n + 1 & \mathsf{i}_{\mathcal{A}}(x) = \omega^x \\ x \mid_{\mathcal{A}} y = x \oplus y & \mathsf{E}_{\mathcal{A}}(x) = x + 1 & \mathsf{C}_{\mathcal{A}}(n, x) = x \cdot n + 1 \\ \mathsf{A}_{\mathcal{A}}(n, x) = \mathsf{B}_{\mathcal{A}}(n, x) = \mathsf{D}_{\mathcal{A}}(n, x) = x & & \end{array}$$

Here $\oplus$ denotes natural addition on ordinals, which is strictly monotone in both arguments.

▶ **Lemma 18.** *The algebra* $(\mathcal{A}, \{>_{\mathsf{O}}, >_{\mathsf{N}}\})$ *is a quasi-model of* $\mathcal{H}/\mathsf{AC}$.

Proof. First note that the interpretation functions are weakly monotone. The rewrite rules in $\mathcal{H}$ are oriented by $\geqslant_{\mathsf{O}}$:

$$\mathsf{A}_{\mathcal{A}}(n, \mathsf{i}_{\mathcal{A}}(\mathsf{h}_{\mathcal{A}})) = \omega \;>_{\mathsf{O}}\; 1 = \mathsf{A}_{\mathcal{A}}(\mathsf{s}_{\mathcal{A}}(n), \mathsf{h}_{\mathcal{A}}) \tag{1}$$

$$\mathsf{A}_{\mathcal{A}}(n, \mathsf{i}_{\mathcal{A}}(\mathsf{h}_{\mathcal{A}} \mid_{\mathcal{A}} x)) = \omega^{x+1} \;>_{\mathsf{O}}\; \omega^x = \mathsf{A}_{\mathcal{A}}(\mathsf{s}_{\mathcal{A}}(n), \mathsf{i}_{\mathcal{A}}(x)) \tag{2}$$

$$\mathsf{A}_{\mathcal{A}}(n, \mathsf{i}_{\mathcal{A}}(x)) = \omega^x \;=_{\mathsf{O}}\; \omega^x = \mathsf{B}_{\mathcal{A}}(n, \mathsf{D}_{\mathcal{A}}(\mathsf{s}_{\mathcal{A}}(n), \mathsf{i}_{\mathcal{A}}(x))) \tag{3}$$

$$\mathsf{C}_{\mathcal{A}}(\mathsf{0}_{\mathcal{A}}, x) = x + 1 \;=_{\mathsf{O}}\; x + 1 = \mathsf{E}_{\mathcal{A}}(x) \tag{4}$$

$$\mathsf{C}_{\mathcal{A}}(\mathsf{s}_{\mathcal{A}}(n), x) = x \cdot n + x + 1 \;=_{\mathsf{O}}\; x \cdot n + x + 1 = x \mid_{\mathcal{A}} \mathsf{C}_{\mathcal{A}}(n, x) \tag{5}$$

$$\mathsf{i}_{\mathcal{A}}(\mathsf{E}_{\mathcal{A}}(x) \mid_{\mathcal{A}} y) = \omega^{x \oplus y + 1} \;>_{\mathsf{O}}\; \omega^{x \oplus y} + 1 = \mathsf{E}_{\mathcal{A}}(\mathsf{i}_{\mathcal{A}}(x \mid_{\mathcal{A}} y)) \tag{6}$$

$$\mathsf{i}_{\mathcal{A}}(\mathsf{E}_{\mathcal{A}}(x)) = \omega^{x+1} \;>_{\mathsf{O}}\; \omega^x + 1 = \mathsf{E}_{\mathcal{A}}(\mathsf{i}_{\mathcal{A}}(x)) \tag{7}$$

$$\mathsf{D}_{\mathcal{A}}(n, \mathsf{i}_{\mathcal{A}}(\mathsf{i}_{\mathcal{A}}(x))) = \omega^{\omega^x} \;=_{\mathsf{O}}\; \omega^{\omega^x} = \mathsf{i}_{\mathcal{A}}(\mathsf{D}_{\mathcal{A}}(n, \mathsf{i}_{\mathcal{A}}(x))) \tag{8}$$

$$\mathsf{D}_{\mathcal{A}}(n, \mathsf{i}_{\mathcal{A}}(\mathsf{i}_{\mathcal{A}}(x) \mid_{\mathcal{A}} y)) = \omega^{\omega^x \oplus y} \;=_{\mathsf{O}}\; \omega^{\omega^x \oplus y} = \mathsf{i}_{\mathcal{A}}(\mathsf{D}_{\mathcal{A}}(n, \mathsf{i}_{\mathcal{A}}(x)) \mid_{\mathcal{A}} y) \tag{9}$$

$$\mathsf{D}_{\mathcal{A}}(n, \mathsf{i}_{\mathcal{A}}(\mathsf{i}_{\mathcal{A}}(\mathsf{h}_{\mathcal{A}} \mid_{\mathcal{A}} x) \mid_{\mathcal{A}} y)) = \omega^{\omega^{x+1} \oplus y} \;>_{\mathsf{O}}\; \omega^{\omega^x \cdot n \oplus y + 1} = \mathsf{i}_{\mathcal{A}}(\mathsf{C}_{\mathcal{A}}(n, \mathsf{i}_{\mathcal{A}}(x)) \mid_{\mathcal{A}} y) \tag{10}$$

$$\mathsf{D}_{\mathcal{A}}(n, \mathsf{i}_{\mathcal{A}}(\mathsf{i}_{\mathcal{A}}(\mathsf{h}_{\mathcal{A}} \mid_{\mathcal{A}} x))) = \omega^{\omega^{x+1}} \;>_{\mathsf{O}}\; \omega^{\omega^x \cdot n + 1} = \mathsf{i}_{\mathcal{A}}(\mathsf{C}_{\mathcal{A}}(n, \mathsf{i}_{\mathcal{A}}(x))) \tag{11}$$

$$\mathsf{D}_{\mathcal{A}}(n, \mathsf{i}_{\mathcal{A}}(\mathsf{i}_{\mathcal{A}}(\mathsf{h}_{\mathcal{A}}) \mid_{\mathcal{A}} y)) = \omega^{\omega \oplus y} \;>_{\mathsf{O}}\; \omega^{(n+1) \oplus y} = \mathsf{i}_{\mathcal{A}}(\mathsf{C}_{\mathcal{A}}(n, \mathsf{h}_{\mathcal{A}}) \mid_{\mathcal{A}} y) \tag{12}$$

$$\mathsf{D}_{\mathcal{A}}(n, \mathsf{i}_{\mathcal{A}}(\mathsf{i}_{\mathcal{A}}(\mathsf{h}_{\mathcal{A}}))) = \omega^{\omega} \;>_{\mathsf{O}}\; \omega^{n+1} = \mathsf{i}_{\mathcal{A}}(\mathsf{C}_{\mathcal{A}}(n, \mathsf{h}_{\mathcal{A}})) \tag{13}$$

$$\mathsf{B}_{\mathcal{A}}(n, \mathsf{E}_{\mathcal{A}}(x)) = x + 1 \;>_{\mathsf{O}}\; x = \mathsf{A}_{\mathcal{A}}(\mathsf{s}_{\mathcal{A}}(n), x) \tag{14}$$

Note that inequalities (10) – (13) use the fact that $\omega >_{\mathsf{O}} n$ holds for $n \in \mathbb{N}$. The compatibility of $\mathcal{A}$ with AC follows from the associativity and the commutativity of $\oplus$:

$$\begin{aligned} (x \mid_{\mathcal{A}} y) \mid_{\mathcal{A}} z = (x \oplus y) \oplus z \;&=_{\mathsf{O}}\; x \oplus (y \oplus z) = x \mid_{\mathcal{A}} (y \mid_{\mathcal{A}} z) \\ x \mid_{\mathcal{A}} y = x \oplus y \;&=_{\mathsf{O}}\; y \oplus x = x \mid_{\mathcal{A}} y \end{aligned}$$

Therefore, $\mathcal{A}$ is a quasi-model of $\mathcal{H}/\mathsf{AC}$. ◀

We now label A and B by the value of their second argument. Let $L_{\mathsf{A}} = L_{\mathsf{B}} = \mathbb{O}$ and $L_f = \varnothing$ for the other function symbols f, and define lab as follows:

$$\mathsf{lab}_{\mathsf{A}}(n, x) = \mathsf{lab}_{\mathsf{B}}(n, x) = x$$

The labeling (L, lab) results in the infinite rewrite system $\mathcal{H}_{\mathsf{lab}} \cup \mathcal{D}\mathsf{ec}$ with $\mathcal{H}_{\mathsf{lab}}$ consisting of the rewrite rules

$$\begin{array}{rlrl}
\mathsf{A}_\omega(n, \mathsf{i}(\mathsf{h})) & \xrightarrow{1} \mathsf{A}_1(\mathsf{s}(n), \mathsf{h}) & \mathsf{D}(n, \mathsf{i}(\mathsf{i}(x))) & \xrightarrow{8} \mathsf{i}(\mathsf{D}(n, \mathsf{i}(x))) \\
\mathsf{A}_{\omega^{v+1}}(n, \mathsf{i}(\mathsf{h} \mid x)) & \xrightarrow{2} \mathsf{A}_{\omega^v}(\mathsf{s}(n), \mathsf{i}(x)) & \mathsf{D}(n, \mathsf{i}(\mathsf{i}(x) \mid y)) & \xrightarrow{9} \mathsf{i}(\mathsf{D}(n, \mathsf{i}(x)) \mid y) \\
\mathsf{A}_{\omega^v}(n, \mathsf{i}(x)) & \xrightarrow{3} \mathsf{B}_{\omega^v}(n, \mathsf{D}(\mathsf{s}(n), \mathsf{i}(x))) & \mathsf{D}(n, \mathsf{i}(\mathsf{i}(\mathsf{h} \mid x) \mid y)) & \xrightarrow{10} \mathsf{i}(\mathsf{C}(n, \mathsf{i}(x)) \mid y) \\
\mathsf{C}(0, x) & \xrightarrow{4} \mathsf{E}(x) & \mathsf{D}(n, \mathsf{i}(\mathsf{i}(\mathsf{h} \mid x))) & \xrightarrow{11} \mathsf{i}(\mathsf{C}(n, \mathsf{i}(x))) \\
\mathsf{C}(\mathsf{s}(n), x) & \xrightarrow{5} x \mid \mathsf{C}(n, x) & \mathsf{D}(n, \mathsf{i}(\mathsf{i}(\mathsf{h}) \mid y)) & \xrightarrow{12} \mathsf{i}(\mathsf{C}(n, \mathsf{h}) \mid y) \\
\mathsf{i}(\mathsf{E}(x) \mid y) & \xrightarrow{6} \mathsf{E}(\mathsf{i}(x \mid y)) & \mathsf{D}(n, \mathsf{i}(\mathsf{i}(\mathsf{h}))) & \xrightarrow{13} \mathsf{i}(\mathsf{C}(n, \mathsf{h})) \\
\mathsf{i}(\mathsf{E}(x)) & \xrightarrow{7} \mathsf{E}(\mathsf{i}(x)) & \mathsf{B}_{v+1}(n, \mathsf{E}(x)) & \xrightarrow{14} \mathsf{A}_v(\mathsf{s}(n), x)
\end{array}$$

for all $v \in \mathbb{O}$ and $\mathcal{D}\mathsf{ec}$ consisting of the rewrite rules

$$\mathsf{A}_v(n, x) \to \mathsf{A}_w(n, x) \qquad\qquad \mathsf{B}_v(n, x) \to \mathsf{B}_w(n, x)$$

for all $v, w \in \mathbb{O}$ with $v > w$. According to Theorem 17, the AC termination of $\mathcal{H}$ on many-sorted terms follows from the AC termination of $\mathcal{H}_{\mathsf{lab}} \cup \mathcal{D}\mathsf{ec}$.

▶ **Corollary 19.** *If $\mathcal{H}_{\mathsf{lab}} \cup \mathcal{D}\mathsf{ec}$ is* AC *terminating then $\mathcal{H}$ is* AC *terminating on sorted terms.*

5 AC-RPO

In order to show AC termination of $\mathcal{H}_{\mathsf{lab}} \cup \mathcal{D}\mathsf{ec}$ we use the AC version of recursive path orders (AC-RPO), introduced by Rubio [20]. In this section we first recall the definition of AC-RPO, following the presentation in [23]. AC-RPO is a relation on terms constructed from a strict order $>$ on function symbols, called *precedence*. AC-RPO collects the arguments of successive occurrences of the same AC symbols in a multiset. This operation is captured by top-flattening. Let $\mathcal{F}_{\mathsf{AC}}$ be the set of AC symbols in $\mathcal{F}$. The *top-flattening* of a term t with respect to $f \in \mathcal{F}_{\mathsf{AC}}$ is the multiset $\triangledown_f(t)$ defined inductively as follows:

$$\triangledown_f(t) = \begin{cases} \triangledown_f(t_1) \uplus \triangledown_f(t_2) & \text{if } t = f(t_1, t_2) \\ \{t\} & \text{otherwise} \end{cases}$$

Multisets resulting from top-flattening are first compared by an embedding-like relation. Let t be a term with $\mathsf{root}(t) = f \in \mathcal{F}_{\mathsf{AC}}$ and $\triangledown_f(t) = \{t_1, \dots, t_n\}$. We write $t \rhd^f_{\mathsf{emb}} u$ for all terms u such that $\triangledown_f(u) = \{t_1, \dots, t_{i-1}, s_j, t_{i+1}, \dots, t_n\}$ for some $t_i = g(s_1, \dots, s_m)$ with $g \not> f$ and $1 \leqslant j \leqslant m$. Then terms in the multisets are grouped according to their forms, and compared in a sophisticated way. For this sake we define the following submultisets of a multiset T of terms:

$$T\restriction_{\mathcal{V}} = \{x \in T \mid x \in \mathcal{V}\} \qquad\qquad \begin{aligned} T\restriction^{>}_{f} &= \{g(t_1, \dots, t_n) \in T \setminus \mathcal{V} \mid g > f\} \\ T\restriction^{\not<}_{f} &= \{g(t_1, \dots, t_n) \in T \setminus \mathcal{V} \mid g \not< f\} \end{aligned}$$

Let $T = \{t_1, \dots, t_n\}$. By $\#(T)$ we denote the linear polynomial $\#(t_1) + \cdots + \#(t_n)$. Here $\#(t)$ is defined as follows:

$$\#(t) = \begin{cases} t & \text{if } t \text{ is a variable} \\ 1 & \text{otherwise} \end{cases}$$

Note that $\#(S) > \#(T)$ and $\#(S) \geqslant \#(T)$ denote comparisons of integer polynomials. After these preliminaries, we are ready to present the definition of AC-RPO.

▶ **Definition 20.** *Let $>$ be a precedence and let $\mathcal{F} \setminus \mathcal{F}_{\mathsf{AC}} = \mathcal{F}_{\mathsf{mul}} \uplus \mathcal{F}_{\mathsf{lex}}$. We define $>_{\mathsf{acrpo}}$ inductively as follows: $s >_{\mathsf{acrpo}} t$ if one of the following conditions holds:*

1. *$s = f(s_1, \ldots, s_n)$ and $s_i \geqslant_{\mathsf{acrpo}} t$ for some $1 \leqslant i \leqslant n$,*
2. *$s = f(s_1, \ldots, s_n)$, $t = g(t_1, \ldots, t_m)$, $f > g$, and $s >_{\mathsf{acrpo}} t_j$ for all $1 \leqslant j \leqslant m$,*
3. *$s = f(s_1, \ldots, s_n)$, $t = f(t_1, \ldots, t_n)$, $f \notin \mathcal{F}_{\mathsf{AC}}$, $s >_{\mathsf{acrpo}} t_j$ for all $1 \leqslant j \leqslant n$, and either*
   (a) *$f \in \mathcal{F}_{\mathsf{lex}}$ and $(s_1, \ldots, s_n) >^{\mathsf{lex}}_{\mathsf{acrpo}} (t_1, \ldots, t_n)$, or*
   (b) *$f \in \mathcal{F}_{\mathsf{mul}}$ and $\{s_1, \ldots, s_n\} >^{\mathsf{mul}}_{\mathsf{acrpo}} \{t_1, \ldots, t_n\}$,*
4. *$s = f(s_1, s_2)$, $t = f(t_1, t_2)$, $f \in \mathcal{F}_{\mathsf{AC}}$, and $s' \geqslant_{\mathsf{acrpo}} t$ for some s' such that $s \rhd^{f}_{\mathsf{emb}} s'$,*
5. *$s = f(s_1, s_2)$, $t = f(t_1, t_2)$, $f \in \mathcal{F}_{\mathsf{AC}}$, $s >_{\mathsf{acrpo}} t'$ for all t' such that $t \rhd^{f}_{\mathsf{emb}} t'$, $S{\restriction}^{\not<}_{f} \uplus S{\restriction}_{\mathcal{V}} \geqslant^{\mathsf{mul}}_{\mathsf{acrpo},f} T{\restriction}^{\not<}_{f} \uplus T{\restriction}_{\mathcal{V}}$ for $S = \nabla_f(s)$ and $T = \nabla_f(t)$, and*
   (a) *$S{\restriction}^{>}_{f} >^{\mathsf{mul}}_{\mathsf{acrpo}} T{\restriction}^{>}_{f}$, or*
   (b) *$\#(S) > \#(T)$, or*
   (c) *$\#(S) \geqslant \#(T)$, and $S >^{\mathsf{mul}}_{\mathsf{acrpo}} T$.*

Here $s >_{\mathsf{acrpo},f} t$ means $s >_{\mathsf{acrpo}} t$ and if $\mathsf{root}(s) \not\geqslant f$ then $\mathsf{root}(s) \geqslant \mathsf{root}(t)$. The relation $=_{\mathsf{AC}}$ is used as preorder in $>^{\mathsf{lex}}_{\mathsf{acrpo}}$, $>^{\mathsf{mul}}_{\mathsf{acrpo}}$, and $\geqslant^{\mathsf{mul}}_{\mathsf{acrpo},f}$ as equivalence relation in $\geqslant_{\mathsf{acrpo}}$.

▶ **Theorem 21** ([20], Theorem 4)**.** *The relation $>_{\mathsf{acrpo}}$ is a AC-compatible rewrite order with the subterm property.*

As a consequence, $>_{\mathsf{acrpo}}$ is an AC-compatible reduction order when the underlying signature is finite. As noted in [20, Section 8.2], this also holds for infinite signatures, provided the precedence $>$ is well-founded. This is important because the signature of $\mathcal{H}_{\mathsf{lab}}$ is infinite. Below, we will formally prove the correctness of the extension, by adopting the approach of [16].

A strict order $>$ on a set A is a *partial well-order* if for every infinite sequence $a_0, a_1, \ldots$ of elements in A there exist indices i and j such that $i < j$ and $a_i \leqslant a_j$. Well-founded total orders (*well-orders*) are partial well-orders. Given a partial well-order $>$ on $\mathcal{F}$, the *embedding* TRS $\mathcal{E}\mathsf{mb}(\mathcal{F}, >)$ consists of the rules $f(x_1, \ldots, x_n) \to x_i$ for every n-ary function symbol and $1 \leqslant i \leqslant n$, together with the rules $f(x_1, \ldots, x_n) \to g(x_{i_1}, \ldots, x_{i_m})$ for all function symbols f and g with arities m and n such that $f > g$, and indices $1 \leqslant i_1 < i_2 < \cdots < i_m \leqslant n$. Here $x_1, \ldots, x_n$ are pairwise distinct variables.

▶ **Theorem 22** ([16], Theorem 5.3)**.** *A rewrite order $\succ$ is well-founded if $\mathcal{E}\mathsf{mb}(\mathcal{F}, >) \subseteq \succ$ for some partial well-order $>$.*

▶ **Theorem 23.** *The relation $>_{\mathsf{acrpo}}$ is an AC-compatible reduction order for every well-founded precedence $>$.*

Proof. Let $>$ be a well-founded precedence. We only need to show well-foundedness of $>_{\mathsf{acrpo}}$ because the other properties follow by Theorem 21. Let $\sqsupset$ be an arbitrary well-order that contains $>$. Trivially, $\sqsupset$ is a partial well-order. The inclusion $\mathcal{E}\mathsf{mb}(\mathcal{F}, \sqsupset) \subseteq \sqsupset_{\mathsf{acrpo}}$ is easily verified. Hence the well-foundedness of $\sqsupset_{\mathsf{acrpo}}$ is obtained from Theorem 22. Since $> \subseteq \sqsupset$, the *incrementality* of AC-RPO [20, Lemma 22] yields $>_{\mathsf{acrpo}} \subseteq \sqsupset_{\mathsf{acrpo}}$. It follows that $>_{\mathsf{acrpo}}$ is well-founded. ◀

We show the termination of $\mathcal{H}_{\mathsf{lab}} \cup \mathcal{D}\mathsf{ec}$ by AC-RPO. To this end, we consider the following precedence $>$ on the labeled signature:

$$\begin{aligned}
\mathsf{A}_v &> \mathsf{A}_w && \text{for all } v, w \in \mathbb{O} \text{ with } v > w\\
\mathsf{B}_v &> \mathsf{B}_w && \text{for all } v, w \in \mathbb{O} \text{ with } v > w\\
\mathsf{B}_{v+1} > \mathsf{A}_v &> \mathsf{B}_v && \text{for all } v \in \mathbb{O}\\
\mathsf{B}_0 &> \mathsf{D} > \mathsf{C} > \mathsf{i} > \mathsf{E} > \mathsf{s} > |
\end{aligned}$$

Note that $>$ is well-founded.

▶ **Theorem 24.** $\mathcal{H}_{\mathsf{lab}} \cup \mathcal{D}\mathsf{ec} \subseteq >_{\mathsf{acrpo}}$

Proof. We show the compatibility verification for rules 3 and 6 of $\mathcal{H}_{\mathsf{lab}}$. The other rewrite rules are handled in a similar fashion. For rule 3 we have (the numbers next to the inference steps refer to the cases in Definition 20)

$$\dfrac{\mathsf{A}_v > \mathsf{B}_v \qquad \dfrac{n \geqslant_{\mathsf{acrpo}} n}{\ell_3 >_{\mathsf{acrpo}} n}\;2 \qquad \dfrac{\mathsf{A}_{\omega^v} > \mathsf{D} \qquad \dfrac{\mathsf{A}_{\omega^v} > \mathsf{s} \qquad \dfrac{n \geqslant_{\mathsf{acrpo}} n}{\ell_3 >_{\mathsf{acrpo}} n}\;1}{\ell_3 >_{\mathsf{acrpo}} \mathsf{s}(n)}\;2 \qquad \dfrac{\mathsf{i}(x) \geqslant_{\mathsf{acrpo}} \mathsf{i}(x)}{\ell_3 >_{\mathsf{acrpo}} \mathsf{i}(x)}\;1}{\ell_3 >_{\mathsf{acrpo}} \mathsf{D}(\mathsf{s}(n), \mathsf{i}(x))}\;2}{\ell_3 = \mathsf{A}_{\omega^v}(n, \mathsf{i}(x)) >_{\mathsf{acrpo}} \mathsf{B}_{\omega^v}(n, \mathsf{D}(\mathsf{s}(n), \mathsf{i}(x)))}\;2$$

For rule 6 we have

$$\dfrac{\mathsf{i} > \mathsf{E} \qquad \dfrac{\dfrac{\dfrac{\dfrac{x \geqslant_{\mathsf{acrpo}} x}{\mathsf{E}(x) >_{\mathsf{acrpo}} x}\;1}{\{\mathsf{E}(x), y\} \geqslant^{\mathsf{mul}}_{\mathsf{acrpo}} \{x, y\}} \qquad \{\mathsf{E}(x)\} >^{\mathsf{mul}}_{\mathsf{acrpo}} \varnothing}{\mathsf{E}(x) \mid y >_{\mathsf{acrpo}} x \mid y}\;5(\mathrm{a})}{\mathsf{i}(\mathsf{E}(x) \mid y) >_{\mathsf{acrpo}} \mathsf{i}(x \mid y)}\;3(\mathrm{a})}{\mathsf{i}(\mathsf{E}(x) \mid y) >_{\mathsf{acrpo}} \mathsf{E}(\mathsf{i}(x \mid y))}\;2$$

The multiset comparisons in 5(a) correspond to $S\restriction^{\ngtr}_{f} \uplus S\restriction_{\mathcal{V}} \geqslant^{\mathsf{mul}}_{\mathsf{acrpo},f} T\restriction^{\ngtr}_{f} \uplus T\restriction_{\mathcal{V}}$ and $S\restriction^{>}_{f} >^{\mathsf{mul}}_{\mathsf{acrpo}} T\restriction^{>}_{f}$ for $f = |$, $S = \nabla_f(\mathsf{E}(x) \mid y)$, and $T = \nabla_f(x \mid y)$. These multisets are calculated as follows:

$$\begin{array}{llll}
S = \{\mathsf{E}(x), y\} & S\restriction^{>}_{f} = S\restriction^{\ngtr}_{f} = \{\mathsf{E}(x)\} & S\restriction_{\mathcal{V}} = \{y\} & S\restriction^{\ngtr}_{f} \uplus S\restriction_{\mathcal{V}} = \{\mathsf{E}(x), y\}\\
T = \{x, y\} & T\restriction^{>}_{f} = T\restriction^{\ngtr}_{f} = \varnothing & T\restriction_{\mathcal{V}} = \{x, y\} & T\restriction^{\ngtr}_{f} \uplus T\restriction_{\mathcal{V}} = \{x, y\}
\end{array}$$ ◀

From Theorems 4 and 24 we conclude that Hercules eventually beats Hydra in any battle. Theorems 24 and 6 in connection with Corollary 19 yield the AC termination of $\mathcal{H}$ on arbitrary terms.

6 Related Work

In an influential survey paper, Dershowitz and Jouannaud [4, p. 270] introduced a 5-rule rewrite system to simulate the Hydra Battle. The proposed rewrite system was later shown to be erroneous. A corrected version together with a detailed termination analysis has been given by Dershowitz and Moser [5], see also Moser [17]. Earlier, Touzet [21] presented an 11-rule rewrite system that encodes a specific battle with weakened Hydras (whose height is bounded by 4) and proved total termination by a semantic termination method. It is worth noting that our rewrite system $\mathcal{H}$ is not even simply terminating on unsorted terms. In fact, we have the following cyclic sequence with respect to $\mathcal{H} \cup \mathcal{E}\mathsf{mb}(\mathcal{F}, \varnothing)$:

$$\begin{aligned}\mathsf{A}(\mathsf{E}(\mathsf{i}(x)),\mathsf{i}(x)) &\xrightarrow{3} \mathsf{B}(\mathsf{E}(\mathsf{i}(x)),\mathsf{D}(\mathsf{s}(\mathsf{E}(\mathsf{i}(x))),\mathsf{i}(x))) \to^{*}_{\mathcal{E}\mathsf{mb}(\mathcal{F},\varnothing)} \mathsf{B}(\mathsf{E}(\mathsf{i}(x)),\mathsf{i}(x))\\ &\xrightarrow{14} \mathsf{A}(\mathsf{s}(\mathsf{E}(\mathsf{i}(x))),\mathsf{i}(x)) \to_{\mathcal{E}\mathsf{mb}(\mathcal{F},\varnothing)} \mathsf{A}(\mathsf{E}(\mathsf{i}(x)),\mathsf{i}(x))\end{aligned}$$

So the TRS $\mathcal{H}$ is not simply terminating (see [16, Lemma 4.6]).

The rewrite systems referred to above model the so-called *standard* battle, which corresponds to a specific strategy for Hercules. In this regard it is interesting to quote Kirby and Paris [11], who introduced the battle as an accessible example of an independence result for Peano arithmetic (P):

> A *strategy* is a function which determines for Hercules which head to chop off at each stage of any battle. It is not hard to find a reasonably fast *winning strategy* (i.e. a strategy which ensures that Hercules wins against any hydra). More surprisingly, Hercules cannot help winning:
>
> Theorem 2. (i) *Every strategy is a winning strategy.*
>
> [. . .]
>
> Theorem 2. (ii) *The statement "every recursive strategy is a winning strategy" is not provable from* P.

In a recent paper [6, Section 6], rules are presented to slay Hydras, independent of the strategy. These rules do not constitute a term rewrite system in the usual sense (they operate on terms with *sequence variables*). More importantly, the infinitely many rules do not faithfully represent the battle. Earlier, Ferreira and Zantema [7, Section 10] presented an infinite rewrite system to model the standard strategy and gave a direct ordinal interpretation to conclude its termination. In neither of the latter two papers stages of the battle are modeled.

7 Conclusion

We presented a new TRS encoding of the Battle of Hydra and Hercules. Unlike earlier encodings, it makes use of AC symbols. This allows to faithfully model any strategy of Hercules, as envisaged in the paper by Kirby and Paris [11] in which the Battle was first presented. To prove the termination of the encoding we employed many-sorted rewriting modulo AC and we extended semantic labeling modulo AC to many-sorted TRSs. The infinite TRS produced by semantic labelling was proved terminating by suitably instantiating AC-RPO.

The finite TRS $\mathcal{H}$ poses an interesting challenge for automatic termination tools. None of the tools (AProVE [8], muterm [1], NaTT [22]) competing in the "TRS Equational" category of last year's Termination Competition[1] succeeds on $\mathcal{H}/\mathsf{AC}$. This is not really surprising since most methods implemented in termination tool come with a multiple recursive upper bound on the derivation height (e.g. [10, 13, 18]). The tools even fail to prove termination of $\mathcal{H}$ without AC. The tool $\mathsf{T_TT_2}$ [12] has support for ordinal interpretations [24] but also fails on $\mathcal{H}$.

Formalizing the techniques used in this paper in a proof assistant is an important task to ensure the correctness of the results. Interestingly, the informal paper [9] in which we announced our encoding also presents a termination proof, essentially extending a semantic

[1] https://termcomp.github.io/Y2022/

method of Touzet [21] and Zantema [27] to AC rewriting. Although we believe the non-trivial extension to be correct, its use in proving the AC termination of $\mathcal{H}$ has a critical mistake, which we recently discovered.

Another topic for future research is to investigate the scope of many-sorted semantic labeling. Can the termination of earlier encodings of the battle be established with many-sorted semantic labeling followed by some standard simplification order? Variants of the battle by Buchholz [2] and Lepper [14] are also of interest here.

References

1 Beatriz Alarcón, Raúl Gutiérrez, Salvador Lucas, and Rafael Navarro-Marset. Proving termination properties with MU-TERM. In *Proc. 13th Algebraic Methodology and Software Technology*, volume 6486 of *Lecture Notes in Computer Science*, pages 201–208, 2011. doi:10.1007/978-3-642-17796-5_12.

2 Wilfried Buchholz. An independence result for $(\pi^1_1 - \mathrm{CA}) + \mathrm{BI}$. *Annals of Pure and Applied Logic*, 33:131–155, 1987. doi:10.1016/0168-0072(87)90078-9.

3 Wilfried Buchholz. Another rewrite system for the standard Hydra battle. In *Proc. Mini-Workshop: Logic, Combinatorics and Independence Results*, volume 3(4) of *Oberwolfach Reports*, pages 3099–3102. European Mathematical Society, 2006.

4 Nachum Dershowitz and Jean-Pierre Jouannaud. *Rewrite Systems. Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 243–320. Elsevier, 1990.

5 Nachum Dershowitz and Georg Moser. The Hydra battle revisited. In *Rewriting, Computation and Proof, Essays Dedicated to Jean-Pierre Jouannaud on the Occasion of his 60th Birthday*, volume 4600 of *Lecture Notes in Computer Science*, pages 1–27, 2007. doi:10.1007/978-3-540-73147-4_1.

6 Jörg Endrullis, Jan Willem Klop, and Roy Overbeek. Star games and Hydras. *Logical Methods in Computer Science*, 17(2):20:1–20:32, 2021. doi:10.23638/LMCS-17(2:20)2021.

7 Maria C. F. Ferreira and Hans Zantema. Total termination of term rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 7(2):133–162, 1996. doi:10.1007/BF01191381.

8 Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann. Analyzing program termination and complexity automatically with AProVE. *Journal of Automated Reasoning*, 58(1):3–31, 2017. doi:10.1007/s10817-016-9388-y.

9 Nao Hirokawa and Aart Middeldorp. Hydra battles and AC termination. In *Proc. 18th International Workshop on Termination*, pages 21–25, 2022.

10 Dieter Hofbauer. Termination proofs by multiset path orderings imply primitive recursive derivation lengths. *Theoretical Computer Science*, 105(1):129–140, 1992. doi:10.1016/0304-3975(92)90289-R.

11 Laurence Kirby and Jeff Paris. Accessible independency results for Peano arithmetic. *Bulletin of the London Mathematical Society*, 14:285–325, 1982. doi:10.1112/blms/14.4.285.

12 Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean Termination Tool 2. In *Proc. 20th International Conference on Rewriting Techniques and Applications*, volume 5595 of *Lecture Notes in Computer Science*, pages 295–304, 2009. doi:10.1007/978-3-642-02348-4_21.

13 Ingo Lepper. Derivation lengths and order types of Knuth–Bendix orders. *Theoretical Computer Science*, 269(1-2):433–450, 2001. doi:10.1016/S0304-3975(01)00015-9.

14 Ingo Lepper. Simply terminating rewrite systems with long derivations. *Archive for Mathematical Logic*, 43(1):1–18, 2004. doi:10.1007/s00153-003-0190-2.

15 Aart Middeldorp and Hitoshi Ohsaki. Type introduction for equational rewriting. *Acta Informatica*, 36(12):1007–1029, 2000. doi:10.1007/PL00013300.

16 Aart Middeldorp and Hans Zantema. Simple termination of rewrite systems. *Theoretical Computer Science*, 175(1):127–158, 1997. doi:10.1016/S0304-3975(96)00172-7.

17 Georg Moser. The Hydra battle and Cichon's principle. *Applicable Algebra in Engineering, Communication and Computing*, 20(2):133–158, 2009. doi:10.1007/s00200-009-0094-4.

18 Georg Moser and Andreas Schnabl. The derivational complexity induced by the dependency pair method. *Logical Methods in Computer Science*, 7(3), 2011. doi:10.2168/LMCS-7(3:1)2011.

19 Hitoshi Ohsaki, Aart Middeldorp, and Jürgen Giesl. Equational termination by semantic labelling. In *Proc. 14th EACSL Annual Conference on Computer Science Logic*, volume 1862 of *Lecture Notes in Computer Science*, pages 457–471, 2000. doi:10.1007/3-540-44622-2_31.

20 Albert Rubio. A fully syntactic AC-RPO. *Information and Computation*, 178(2):515–533, 2002. doi:10.1006/inco.2002.3158.

21 Hélène Touzet. Encoding the Hydra battle as a rewrite system. In *Proc. 23rd International Symposium on Mathematical Foundations of Computer Science*, volume 1450 of *Lecture Notes in Computer Science*, pages 267–276, 1998. doi:10.1007/BFb0055776.

22 Akihisa Yamada, Keiichirou Kusakari, and Toshiki Sakabe. Nagoya termination tool. In *Proc. 25th International Conference on Rewriting Techniques and Applications and 12th International Conference on Typed Lambda Calculi and Applications*, volume 8560 of *Lecture Notes in Computer Science*, pages 466–475, 2014. doi:10.1007/978-3-319-08918-8_32.

23 Akihisa Yamada, Sarah Winkler, Nao Hirokawa, and Aart Middeldorp. AC-KBO revisited. *Theory and Practice of Logic Programming*, 16(2):163–188, 2016. doi:10.1017/S1471068415000083.

24 Harald Zankl, Sarah Winkler, and Aart Middeldorp. Beyond polynomials and Peano arithmetic—Automation of elementary and ordinal interpretations. *Journal of Symbolic Computation*, 69:129–158, 2015. doi:10.1016/j.jsc.2014.09.033.

25 Hans Zantema. Termination of term rewriting: Interpretation and type elimination. *Journal of Symbolic Computation*, 17(1):23–50, 1994. doi:10.1006/jsco.1994.1003.

26 Hans Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995. doi:10.3233/FI-1995-24124.

27 Hans Zantema. The termination hierarchy for term rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 12(1-2):3–19, 2001. doi:10.1007/s002000100061.

Strategies as Resource Terms, and Their Categorical Semantics

Lison Blondeau-Patissier
Aix Marseille Univ, CNRS, I2M and LIS, Marseille, France

Pierre Clairambault
Aix Marseille Univ, CNRS, LIS, Marseille, France

Lionel Vaux Auclair
Aix Marseille Univ, CNRS, I2M, Marseille, France

Abstract

As shown by Tsukada and Ong, simply-typed, normal and η-long resource terms correspond to plays in Hyland-Ong games, quotiented by Melliès' homotopy equivalence. Though inspiring, their proof is indirect, relying on the injectivity of the relational model w.r.t. both sides of the correspondence – in particular, the dynamics of the resource calculus is taken into account only *via* the compatibility of the relational model with the composition of normal terms defined by normalization.

In the present paper, we revisit and extend these results. Our first contribution is to restate the correspondence by considering causal structures we call *augmentations*, which are canonical representatives of Hyland-Ong plays up to homotopy. This allows us to give a direct and explicit account of the connection with normal resource terms. As a second contribution, we extend this account to the *reduction* of resource terms: building on a notion of strategies as weighted sums of augmentations, we provide a denotational model of the resource calculus, invariant under reduction. A key step – and our third contribution – is a categorical model we call a *resource category*, which is to the resource calculus what differential categories are to the differential λ-calculus.

2012 ACM Subject Classification Theory of computation → Denotational semantics; Theory of computation → Categorical semantics

Keywords and phrases Resource calculus, Game semantics, Categorical semantics

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.13

Related Version *Full Version*: https://arxiv.org/abs/2302.04685

Funding *Pierre Clairambault*: Supported by the ANR projects DyVerSe (ANR-19-CE48-0010-01) and by the Labex MiLyon (ANR-10-LABX-0070) of Université de Lyon.
Lionel Vaux Auclair: Supported by the ANR projects PPS (ANR-19-CE48-0014), LambdaComb (ANR-21-CE48-0017) and Reciprog (ANR-21-CE48-0019).

1 Introduction

The *Taylor expansion* of programs translates programs with possibly infinite behaviour to infinite formal sums of terms of a language with a strongly finitary behaviour called the *resource calculus*. Its discovery dates back to Ehrhard and Regnier's *differential λ-calculus* [13], reifying syntactically features of certain vectorial models of linear logic. Since its inception [15], Taylor expansion was intended as a quantitative alternative to order-based approximation techniques, such as Scott continuity and Böhm trees. For instance, Barbarossa and Manzonetto leveraged it to get simpler proofs of known results in pure λ-calculus [2].

Game semantics is another well-established line of work, also representing programs as collections of finite behaviours. It is particularly well known for its many full abstraction results [16, 1]. How different is the Taylor expansion of the λ-calculus from its game semantics?

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 13; pp. 13:1–13:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Not very different, suggest Tsukada and Ong [23], who show that certain normal and η-long resource terms correspond bijectively to plays in the sense of Hyland-Ong game semantics [16], up to Opponent's scheduling of the independent explorations of separate branches of the term, as formalized by Melliès' homotopy equivalence on plays [19].

The account of this insightful result by Tsukada and Ong is inspiring, but it also comes with limitations. Their focus is on normal resource terms, and the dynamics is treated only in the form of the composition of terms, *i.e.* substitution followed by normalization. The correspondence is also very indirect, relying on the injectivity of the relational model w.r.t. both normal resource terms and plays up to homotopy. In [23], after laying out the intuitions supporting the correspondence, Tsukada and Ong motivate this indirect route, writing: "The idea will now be intuitively clear. However the definition based on the above argument, which heavily depends on graphical operations, does not seem so easy to handle."

In the present paper, we handle this very task. We rely on a representation of plays quotienting out Opponent's scheduling, recently introduced by the first two authors [4]. This was inspired by concurrent games [11] – similar causal structures existed before, first suggested in [17], and fleshed out more in [22]. In [4], plays are replaced by so-called *augmentations*, which *augment* valid states of the game with causal constraints imposed by the program. Our first contribution is an explicit description of the bijection between normal resource terms and isomorphism classes of augmentations (called *isogmentations*, for the sake of brevity), in a style similar to traditional finite definability arguments: see Section 3.2.

We moreover strive to account for non-normal resource terms and *reduction* in the resource calculus, which we recall in Section 2. In game semantics, this typically relies on a category of strategies, whose composition is defined by interaction between plays. Considering the interaction of augmentations – which was not addressed in [4] – an interesting phenomenon occurs. Indeed, there is no canonical way to *synchronize* two augmentations: they can only interact *via* a mediating map, called a *symmetry*, and the result of the interaction depends on the chosen symmetry! Composition is then obtained by summing over all symmetries, as discussed in Section 3.3. This is not an artificial phenomenon arising from our implementation choices: it is analogous to the non-determinism inherent to the substitution of resource terms. And this is instrumental in our second contribution: the correspondence between normal resource terms and isogmentations refines into a denotational interpretation, invariant under reduction, of resource terms as "strategies" – weighted sums of isogmentations.

To establish this result we expose the structure of the category of strategies that is relevant to obtain a model of the resource calculus: we call *resource categories* the resulting categorical model, which is our third contribution, in Section 4. And, in Section 5, we show that strategies indeed form a resource category, completing the proof of the previous point.

Related and future work. As mentioned above, Tsukada and Ong [23] considered some dynamic aspects of the correspondence: they proved their bijection compatible with the compositions of terms and plays, *via* composition in the relational model. Nonetheless, they did not consider an interpretation of *non-normal* resource terms as strategies: the question of invariance under reduction could not even be formulated, and the relevant structure of the category of strategies could not be exposed. Still, they state that the normal form of the Taylor expansion of a λ-term is isomorphic to its game semantics.

Our results constitute a first step to flesh out this isomorphism into the diagrams:

$$\begin{tikzcd}
M \arrow[r, mapsto, "\mathcal{T}"] \arrow[d, mapsto, "\llbracket - \rrbracket"'] \arrow[dr, phantom, "(a)"] & \mathcal{T}(M) \arrow[r, mapsto, "\mathcal{N}"] \arrow[d, mapsto, "\llbracket - \rrbracket"] \arrow[dr, phantom, "(b)"] & \mathcal{N}(\mathcal{T}(M)) \arrow[d, phantom, "\wr"] \\
\llbracket M \rrbracket \arrow[r, phantom, "="] & \llbracket \mathcal{T}(M) \rrbracket \arrow[r, phantom, "="] & \llbracket \mathcal{N}(\mathcal{T}(M)) \rrbracket
\end{tikzcd}
\qquad
\begin{tikzcd}
s \arrow[r, mapsto, "\mathcal{N}"] \arrow[d, mapsto, "\llbracket - \rrbracket"'] \arrow[dr, phantom, "(c)"] & \mathcal{N}(s) \arrow[d, phantom, "\wr"] \\
\llbracket s \rrbracket \arrow[r, phantom, "="] & \llbracket \mathcal{N}(s) \rrbracket
\end{tikzcd}$$

where M is a λ-term, s is a resource term, $\mathcal{T}$ is Taylor expansion, $\mathcal{N}$ is normalization and $[\![-]\!]$ is game semantics. Square (a) should commute essentially by definition, while square (b) should be deduced from (c): we leave the treatment of Taylor expansion for future work (see also the next paragraphs and Section 6) but (c) already follows from our present results.

A significant aspect of our contributions is to take coefficients into account. This is far from anecdotal: it requires new methods (we cannot get that *via* the relational model), it makes the development significantly more complex, and it is necessary if one expects to apply these tools to a quantitative setting (e.g., with probabilities) and to provide the basis of a full game semantical account of quantitative Taylor expansion.

The exact correspondence between differential categories [5] and resource categories is also left for future work. Anyway, we stress the fact that the resource calculus is the finitary fragment of the differential λ-calculus: it does not contain the pure λ-calculus. Accordingly, models of the resource calculus are rather related to those of promotion-free differential linear logic [14]: the exponential modality (!) need not be a comonad. From such a model, one can recover an interpretation of the full differential λ-calculus *via* Taylor expansion, *provided the necessary infinite sums are available.* So we are convinced (see our concluding remarks in Section 6) that our category of games does induce a cartesian closed differential category [6, 9, 18]; more generally, we plan to study how this generalizes to any resource category – provided the necessary sums of morphisms are available.

Outline. In Section 2, we detail our resource calculus. In Section 3 we introduce *augmentations*, show the correspondence with normal resource terms, and introduce *strategies*. In Section 4 we introduce resource categories, define the interpretation of the resource calculus, and prove that it is invariant under reduction. In Section 5, we show that strategies form a resource category. We conclude in Section 6.

2 The Simply-Typed η-Expanded Resource Calculus

Preliminaries. If X is a set, we write X^* for the set of finite lists, or tuples, of elements of X, ranged over by $\vec{a}, \vec{b}$, *etc.* We write $\vec{a} = \langle a_1, \dots, a_n\rangle$ to list the elements of $\vec{a}$, of length $|\vec{a}| = n$. The empty list is $\langle\rangle$, and concatenation is simply juxtaposition, *e.g.*, $\vec{a}\vec{b}$. We write $\mathcal{B}(X)$ for the set of finite multisets of elements of X, which we call **bags**, ranged over by $\bar{a}, \bar{b}$, *etc.* We write $[a_1, \dots, a_n]$ for the bag $\bar{a}$ defined by a list $\vec{a} = \langle a_1, \dots, a_n\rangle$ of elements: we then say $\vec{a}$ is an **enumeration** of $\bar{a}$. We write $[]$ for the empty bag, and use $*$ for bag concatenation. We also write $|\bar{a}|$ for the size of $\bar{a}$: $|\bar{a}|$ is the length of any enumeration of $\bar{a}$.

We shall often need to *partition* bags, which requires some care. For $\bar{a} \in \mathcal{B}(X)$ and $k \in \mathbf{N}$, a ***k*-partitioning** of $\bar{a}$ is a function $p : \{1, \dots, |\bar{a}|\} \to \{1, \dots, k\}$: we write $p : \bar{a} \lhd k$. Given an enumeration $\langle a_1, \dots, a_n\rangle$ of $\bar{a}$, the associated ***k*-partition** is the tuple $\langle \bar{a} \upharpoonright_p 1, \dots, \bar{a} \upharpoonright_p k\rangle$, where we set $\bar{a} \upharpoonright_p i = [a_j \mid p(j) = i]$ for $1 \le i \le k$, so that $\bar{a} = \bar{a} \upharpoonright_p 1 * \dots * \bar{a} \upharpoonright_p k$. The obtained k-partition does depend on the chosen enumeration of $\bar{a}$ but, for any function $f : \mathcal{B}(X)^k \to \mathcal{M}$ with values in a commutative monoid $\mathcal{M}$ (noted additively), the sum

$$\sum_{\bar{a} \lhd \bar{a}_1 * \dots * \bar{a}_k} f(\bar{a}_1, \dots, \bar{a}_k) \overset{\mathsf{def}}{=} \sum_{p : \bar{a} \lhd k} f(\bar{a} \upharpoonright_p 1, \dots, \bar{a} \upharpoonright_p k)$$

is independent of the enumeration. When indexing a sum with $\bar{a} \lhd \bar{a}_1 * \dots * \bar{a}_k$ we thus mean to sum over all partitionings $p : \bar{a} \lhd k$, $\bar{a}_i$ being shorthand for $\bar{a} \upharpoonright_p i$ in the summand.

We will also use tuples of bags: we write $\mathcal{S}(X)$ for $\mathcal{B}(X)^*$. We denote elements of $\mathcal{S}(X)$ as $\vec{a}, \vec{b}$, *etc.* just like for plain tuples, but we reserve the name **sequence** for such tuples of bags. A ***k*-partitioning** $p : \vec{a} \lhd k$ of $\vec{a} = \langle \bar{a}_1, \dots, \bar{a}_n\rangle$ is a tuple $p = \langle p_1, \dots, p_n\rangle$ of k-partitionings

$$y\langle \bar{t}/x\rangle \overset{\mathsf{def}}{=} \begin{cases} t & \text{if } y = x \text{ and } \bar{t} = [t] \\ y & \text{if } y \neq x \text{ and } \bar{t} = [] \\ 0 & \text{otherwise} \end{cases} \qquad (s\,\bar{u})\langle \bar{t}/x\rangle \overset{\mathsf{def}}{=} \sum_{\bar{t} \lhd \bar{t}_1 * \bar{t}_2} (s\langle \bar{t}_1/x\rangle)\,(\bar{u}\langle \bar{t}_2/x\rangle)$$

$$[s_1, \dots, s_n]\langle \bar{t}/x\rangle \overset{\mathsf{def}}{=} \sum_{\bar{t} \lhd \bar{t}_1 * \cdots * \bar{t}_n} [s_1\langle \bar{t}_1/x\rangle, \dots, s_n\langle \bar{t}_n/x\rangle]$$

$$(\lambda z.s)\langle \bar{t}/x\rangle \overset{\mathsf{def}}{=} \lambda z.s\langle \bar{t}/x\rangle$$

Figure 1 Inductive definition of substitution (z is chosen fresh in the abstraction case).

$$\frac{}{(\lambda x.s)\,\bar{t} \to s\langle \bar{t}/x\rangle} \qquad \frac{s \to S'}{\lambda x.s \to \lambda x.S'} \qquad \frac{s \to S'}{s\,\bar{t} \to S'\,\bar{t}} \qquad \frac{s \to S'}{[s] * \bar{t} \to [S'] * \bar{t}} \qquad \frac{\bar{t} \to \bar{T}'}{s\,\bar{t} \to s\,\bar{T}'}$$

Figure 2 Rules of single-step reduction.

$p_i : \bar{a}_i \lhd k$. This defines a **partition** $\langle \vec{a} \upharpoonright_p 1, \dots, \vec{a} \upharpoonright_p k\rangle$, component-wise: each $\vec{a} \upharpoonright_p i$ is the sequence $\langle \bar{a}_1 \upharpoonright_{p_1} i, \dots, \bar{a}_n \upharpoonright_{p_n} i\rangle$. We obtain $\vec{a} = \vec{a} \upharpoonright_p 1 * \cdots * \vec{a} \upharpoonright_p k$, applying the concatenation of bags component-wise, to sequences all of the same length n. And, just as before, the result of the following sum is independent from the enumeration of the bags of $\vec{a}$:

$$\sum_{\vec{a} \lhd \vec{a}_1 * \cdots * \vec{a}_k} f(\vec{a}_1, \dots, \vec{a}_k) \overset{\mathsf{def}}{=} \sum_{p : \vec{a} \lhd k} f(\vec{a} \upharpoonright_p 1, \dots, \vec{a} \upharpoonright_p k)\ .$$

Resource calculus. The terms of the resource calculus [15] are just like λ-terms, except that, in an application, the argument is a bag of terms instead of just one term. We denote terms by s, t, u and bags of terms by $\bar{s}, \bar{t}, \bar{u}$, and write Δ for the set of terms:

$$\Delta \ni s, t, u, \dots ::= x \mid \lambda x.s \mid s\,[t_1, \dots, t_n] \quad .$$

The dynamics relies on a multilinear variant of substitution, that we will call **resource substitution**: a redex $(\lambda x.s)\,\bar{t}$ reduces to a (non-idempotent) sum $s\langle \bar{t}/x\rangle$ of terms obtained by substituting each element of $\bar{t}$ for exactly one occurrence of x in s. The inductive definition is in Figure 1, relying on an extension of syntactic constructs to finite sums of expressions:

$$\lambda x.S \overset{\mathsf{def}}{=} \sum_{i \in I} \lambda x.s_i \qquad [S] * \bar{T} \overset{\mathsf{def}}{=} \sum_{i \in I} \sum_{j \in J} [s_i] * \bar{t}_j \qquad S\,\bar{T} \overset{\mathsf{def}}{=} \sum_{i \in I} \sum_{j \in J} s_i\,\bar{t}_j\ ,$$

for $S = \sum_{i \in I} s_i$ and $\bar{T} = \sum_{j \in J} \bar{t}_j$. The actual protagonists of the calculus are thus sums of terms rather than single terms. We will generally write $\Sigma(X)$ for the set of finite formal sums on set X – those may be considered as finite multisets, but we adopt a distinct additive notation to avoid confusion with bags. Resource substitution is in turn extended by linearity, setting $S\langle \bar{T}/x\rangle \overset{\mathsf{def}}{=} \sum_{i \in I} \sum_{j \in J} s_i\langle \bar{t}_j/x\rangle$ with the same notations as above.

The **reduction of resource terms** $\to \subseteq \Delta \times \Sigma(\Delta)$ is given in Figure 2. It is extended to $\Sigma(\Delta) \times \Sigma(\Delta)$ by setting $S \to S'$ whenever $S = t + U$ and $S' = T' + U$ with $t \to T'$.

▶ **Theorem 1** ([15]). *The reduction $\to$ on $\Sigma(\Delta)$ is confluent and strongly normalizing.*

$$\frac{\Gamma, \vec{x} : \vec{F} \vdash_{\mathsf{Base}} s : o}{\Gamma \vdash_{\mathsf{Val}} \lambda \vec{x}.s : \vec{F} \to o}\,\mathrm{abs} \qquad \frac{\Gamma \vdash_{\mathsf{Val}} s : \vec{F} \to o \qquad \Gamma \vdash_{\mathsf{Seq}} \vec{t} : \vec{F}}{\Gamma \vdash_{\mathsf{Base}} s\,\vec{t} : o}\,\mathrm{hr} \qquad \frac{\Gamma \vdash_{\mathsf{Var}} x : \vec{F} \to o \qquad \Gamma \vdash_{\mathsf{Seq}} \vec{t} : \vec{F}}{\Gamma \vdash_{\mathsf{Base}} x\,\vec{t} : o}\,\mathrm{hv}$$

$$\frac{}{\Gamma, x : F \vdash_{\mathsf{Var}} x : F}\,\mathrm{id} \qquad \frac{\Gamma \vdash_{\mathsf{Val}} s_1 : F \quad \cdots \quad \Gamma \vdash_{\mathsf{Val}} s_n : F}{\Gamma \vdash_{\mathsf{Bag}} [s_1, \dots, s_n] : F}\,\mathrm{bag} \qquad \frac{\Gamma \vdash_{\mathsf{Bag}} \bar{s}_1 : F_1 \quad \cdots \quad \Gamma \vdash_{\mathsf{Bag}} \bar{s}_n : F_n}{\Gamma \vdash_{\mathsf{Seq}} \langle \bar{s}_1, \dots, \bar{s}_n \rangle : \langle F_1, \dots, F_n \rangle}\,\mathrm{seq}$$

Figure 3 Typing rules for the simply-typed resource calculus.

Typing and expanded terms. In the remainder of the paper, we will consider a simply typed version of the resource calculus, based on the following grammar of types

$$F, G, H, \dots ::= o \mid F \to G$$

for a single base type o. If $\vec{F} = \langle F_1, \dots, F_n \rangle$, we write $\vec{F} \to G \overset{\mathsf{def}}{=} F_1 \to \cdots \to F_n \to G = F_1 \to (\cdots \to (F_n \to G) \cdots)$. Then any type H can be written uniquely as $H = \vec{F} \to o$.

The above strong normalization result holds in the untyped setting. We use typing only to enforce a syntactic constraint on terms: our resource expressions are η-expanded, *i.e.* values of type $\vec{F} \to o$ are terms $\lambda x_1 \dots \lambda x_{|\vec{F}|}.s$ with s of type o. We fix a type for each variable, so that each type has infinitely many variables – and write $x : F$ for F the type of x. A typing context Γ is a finite set of typed variables. As usual we write it as any enumeration $x_1 : F_1, \dots, x_n : F_n$, abbreviated as $\vec{x} : \vec{F}$; we may then also write $\lambda \vec{x}.s \overset{\mathsf{def}}{=} \lambda x_1 \dots \lambda x_n.s$. We call **resource sequence** any sequence $\vec{s} \in \mathcal{S}(\Delta) = \mathcal{B}(\Delta)^*$. Given a term s and a resource sequence $\vec{t} = \langle \bar{t}_1, \dots, \bar{t}_k \rangle$, we also define the application $s\,\vec{t} \overset{\mathsf{def}}{=} s\,\bar{t}_1 \cdots \bar{t}_k = (\cdots (s\,\bar{t}_1) \cdots)\,\bar{t}_k$.

We extend resource substitution to sequences by setting

$$\langle \bar{s}_1, \dots, \bar{s}_n \rangle \langle \bar{t}/x \rangle \overset{\mathsf{def}}{=} \sum_{\bar{t} \lhd \bar{t}_1 * \cdots * \bar{t}_n} \langle \bar{s}_1 \langle \bar{t}_1/x \rangle, \dots, \bar{s}_n \langle \bar{t}_n/x \rangle \rangle$$

so that $(s\,\vec{u})\langle \bar{t}/x \rangle = \sum_{\bar{t} \lhd \bar{t}_1 * \bar{t}_2} (s\langle \bar{t}_1/x \rangle)\,(\vec{u}\langle \bar{t}_2/x \rangle)$, as in the application case of Figure 1.

The type system appears in Figure 3. For $X \in \{\mathsf{Val}, \mathsf{Base}, \mathsf{Bag}, \mathsf{Seq}\}$, we write $X(\Gamma; F)$ for the set of those s *s.t.* $\Gamma \vdash_X s : F$. For $X = \mathsf{Base}$ we have $F = o$, so we set $\mathsf{Base}(\Gamma) \overset{\mathsf{def}}{=} \mathsf{Base}(\Gamma; o)$. If $\Gamma \vdash_X s : F$, then s is in normal form iff the judgment is derived without (hr) – we write $X_{\mathsf{nf}}(\Gamma; F)$ for the elements of $X(\Gamma; F)$ in normal form. We write $\Sigma X(\Gamma; F)$ for $\Sigma(X(\Gamma; F))$.

▶ **Lemma 2** (Subject reduction). *If $S \in \Sigma\mathsf{Val}(\Gamma; F)$ and $S \to S'$ then $S' \in \Sigma\mathsf{Val}(\Gamma; F)$.*

This follows from substitution lemmas for our four kinds of typed terms, proved by mutual induction: if $\Gamma, x : F \vdash_X t : G$ and $\Gamma \vdash_{\mathsf{Bag}} \bar{s} : F$ then $t\langle \bar{s}/x \rangle \in \Sigma X(\Gamma; G)$.

We also consider a many-step variant of resource reduction, following the structure of expanded terms. We set $s\langle \vec{t}/\vec{x} \rangle \overset{\mathsf{def}}{=} s\langle \bar{t}_1/x_1 \rangle \cdots \langle \bar{t}_n/x_n \rangle$ when $\vec{x} = \langle x_1, \dots, x_n \rangle$, $\vec{t} = \langle \bar{t}_1, \dots, \bar{t}_n \rangle$, and no x_i occurs free in $\vec{t}$. The **many-step reduction** $\Rightarrow$ is then defined from the base case

$$(\lambda \vec{x}.s)\,\vec{t} \in \mathsf{Base}(\Gamma) \Rightarrow s\langle \vec{t}/\vec{x} \rangle \in \Sigma\mathsf{Base}(\Gamma) \quad \text{(assuming } |\vec{x}| = |\vec{t}| \neq 0\text{)},$$

extended contextually to each syntactic kind of typed expressions, following inductively the type system of Figure 3, and then to sums as for $\to$. It is clear that $\Rightarrow\, \subset\, \to^+$ (the transitive closure of $\to$), and that an expanded term is $\Rightarrow$-reducible iff it is $\to$-reducible: it follows that $\Rightarrow$ is strongly normalizing, with the same normal forms as $\to$. In particular $\Rightarrow$ is confluent.

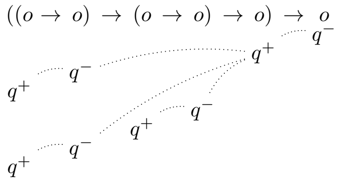


Figure 4 A play in HO games.

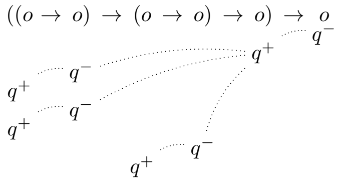


Figure 5 A homotopic play.

3 Resource Terms as Augmentations

Plays in game semantics. In Hyland-Ong game semantics [16] executions are formalized as *plays*, drawn as in Figure 4, read temporally from top to bottom. Nodes are called *moves*, negative (from Opponent / the environment) or positive (from Player / the program) – each corresponds to a resource call, and the dotted lines, called *justification pointers*, carry the hierarchical relationship between those calls. Both Figures 4 and 5 represent plays for

$$\vdash \lambda f^{(o\to o)\to(o\to o)\to o}.\, f\,(\lambda x^o.\, x)\,(\lambda y^o.\, y) : ((o \to o) \to (o \to o) \to o) \to o\,,$$

where Figure 4 reads as follows: Opponent starts computation with the initial q^-, to which Player reacts with the first q^+, corresponding to calling f. With q^- on the third line, Opponents prompts f to call its first argument, to which Player responds the q^+ on the fourth line: a call to x. Subsequently Opponent evaluates the second argument of f – Player responds by calling y – and then Opponent calls the first argument again.

Plays and resource terms. In [23], seeking a syntactic counterpart to the *plays* of HO games, Tsukada and Ong state: "plays in HO/N-games are terms of a well-known and important calculus, the resource calculus". This is natural as both game semantics and the resource calculus are quantitative and represent explicitly resource usage: in Figure 4, the first argument of f is evaluated *twice* while the second one is evaluated *once* – following Tsukada and Ong's correspondence, the play is written $s = \lambda f.\, f\,[\lambda x.\, x, \lambda x.\, x]\,[\lambda y.\, y]$ in the resource calculus. However, Figure 5 *also* corresponds to s! Tsukada and Ong actually establish a bijection between (normal) resource terms and plays *up to* Melliès' *homotopy relation* [19], relating plays which, like Figures 4 and 5, only differ via Opponent's scheduling. But then, is there a more explicit representation of plays up to homotopy?

As a matter of fact, there is. In [4], Blondeau-Patissier and Clairambault introduced a *causal* representation of innocent strategies (inspired from *concurrent games* [11, 12] – see also [22]) as a technical tool to prove a positional injectivity theorem for innocent strategies. There a strategy is not a set of plays, but instead gathers diagrams as in Figure 6 in which the trained eye can read exactly the same data as in the resource term s : the model replaces the chronological *plays* of game semantics with causal structures called *augmentations*, of which the plays are just particular linearizations. Thus as the first contribution of this paper, we refine Tsukada and Ong's result into a bijection of resource terms with *augmentations*.

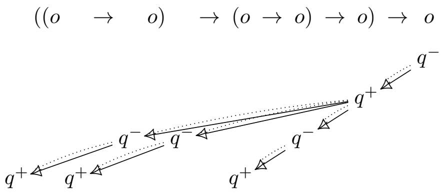


Figure 6 An augmentation.

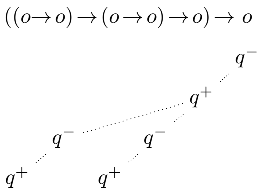


Figure 7 An arena.

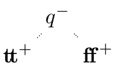


Figure 8 Arena **bool**.

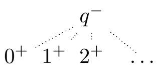


Figure 9 Arena **nat**.

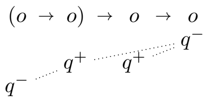


Figure 10 Arena $(o \Rightarrow o) \Rightarrow o \Rightarrow o$.

3.1 Arenas, Positions, Augmentations

▶ **Definition 3.** *An **arena** is $A = \langle |A|, \leq_A, \mathrm{pol}_A \rangle$ where $\langle |A|, \leq_A \rangle$ is a (countable) partial order, and $\mathrm{pol}_A : |A| \to \{-, +\}$ is a **polarity function**. These data must satisfy:*

$$\begin{aligned} \text{finitary:}\quad & \textit{for all } a \in |A|,\ [a]_A \overset{\mathsf{def}}{=} \{a' \in |A| \mid a' \leq_A a\} \textit{ is finite,} \\ \text{forestial:}\quad & \textit{for all } a_1, a_2 \leq_A a, \textit{ then } a_1 \leq_A a_2 \textit{ or } a_2 \leq_A a_1, \\ \text{alternating:}\quad & \textit{for all } a_1 \rightarrowtriangle_A a_2, \textit{ then } \mathrm{pol}_A(a_1) \neq \mathrm{pol}_A(a_2), \end{aligned}$$

*where $a_1 \rightarrowtriangle_A a_2$ means $a_1 <_A a_2$ with no event strictly in between. A **$-$-arena** is additionally* negative, i.e. $\mathrm{pol}_A(a) = -$ *for all $a \in \min(A) \overset{\mathsf{def}}{=} \{a \in |A| \mid a \textit{ minimal}\}$.*

Elements of $|A|$ are called *events* or *moves* interchangeably. An **isomorphism** $\varphi : A \cong B$ between arenas is a bijection between events preserving and reflecting all structure.

Arenas present computational events with their causal dependencies: positive moves for Player, and negative moves for Opponent. We often annotate moves with their polarity. In arenas we draw the immediate causality $\rightarrowtriangle$ as dotted lines, read from top to bottom. Figures 8 and 9 show the arenas **bool** and **nat**. In those arenas, *initial* (*i.e.* minimal) moves are Opponent moves starting computation, to which Player may respond with a value.

Constructions. We write $X + Y$ for the disjoint union $(\{1\} \times X) \cup (\{2\} \times Y)$ of sets.

▶ **Definition 4.** *The **tensor** of arenas A_1 and A_2 is defined in Figure 11.*

*If additionally A_1 and A_2 are $-$-arenas and A_2 is **pointed**,* i.e. $\min(A_2)$ *is a singleton, then the **arrow** $A_1 \Rightarrow A_2$, a pointed $-$-arena, is defined in Figure 12.*

The tensor directly extends to countable arity, and each arena decomposes as $A \cong \otimes_{i \in I} A_i$ with A_i pointed. We set $A^\perp$ as A with polarities reversed. We write $A \vdash B$ for $A^\perp \otimes B$, 1 is the empty arena and o has exactly one (negative) move q. We interpret types as arenas via $[\![o]\!] = o$ and $[\![A \to B]\!] = [\![A]\!] \Rightarrow [\![B]\!]$, and contexts via $[\![\Gamma]\!] = \otimes_{(x:A) \in \Gamma} [\![A]\!]$. Figure 10 shows the interpretation of $(o \to o) \to o \to o$, following the longstanding game semantics convention that keeps moves distinct by attempting to always place them under the corresponding atom.

$$\begin{array}{rcl} |A_1 \otimes A_2| & = & |A_1| + |A_2| \\ (i,a) \leq_{A_1 \otimes A_2} (j,b) & \Leftrightarrow & i = j \quad \wedge \quad a \leq_{A_i} b \\ \mathrm{pol}_{A_1 \otimes A_2}(i,a) & = & \mathrm{pol}_{A_i}(a)\,. \end{array}$$

Figure 11 Tensor of arenas.

$$\begin{array}{rcl} |A_1 \Rightarrow A_2| & = & |A_1| + |A_2| \\ (i,a) \leq_{A_1 \Rightarrow A_2} (j,b) & \Leftrightarrow & (i = j \wedge a \leq_{A_i} b) \\ & \vee & (i = 2 \wedge a \in \min(A_2)) \\ \mathrm{pol}_{A_1 \Rightarrow A_2}(i,a) & = & (-1)^i \cdot \mathrm{pol}_{A_i}(a)\,, \end{array}$$

Figure 12 Arrow of arenas.

Configurations. Next we define the *states* reached when playing on arena A. Intuitively, a state is a sub-tree of A but where each branch may be explored multiple times – such structures were first introduced by Boudes [7] under the name *thick subtrees.* Here, by analogy with concurrent games [11], we call them *configurations*:

▶ **Definition 5.** *A* ***configuration*** $x \in \mathcal{C}(A)$ *of arena* A *is* $x = \langle |x|, \leq_x, \partial_x \rangle$ *such that* $\langle |x|, \leq_x \rangle$ *is a finite forest, and the* ***display map*** $\partial_x : |x| \to |A|$ *is a function s.t.:*

minimality-respecting: *for any* $a \in |x|$*,* a *is* $\leq_x$*-minimal iff* $\partial_x(a)$ *is* $\leq_A$*-minimal,*
causality-preserving: *for all* $a_1, a_2 \in |x|$*, if* $a_1 \rightarrowtriangle_x a_2$ *then* $\partial_x(a_1) \rightarrowtriangle_A \partial_x(a_2)$*,*

and x *is* ***pointed*** *(noted* $x \in \mathcal{C}_\bullet(A)$*) if it has exactly one minimal event* $\mathsf{init}(x)$*.*

A polarity on x is deduced by $\mathrm{pol}(a) = \mathrm{pol}_A(\partial_x(a))$. We write a^- (resp. a^+) for a s.t. $\mathrm{pol}(a) = -$ (resp. $\mathrm{pol}(a) = +$). Ignoring the arrows $\rightarrowtriangle$, Figure 6 is a configuration on Figure 7 – notice that the branch on the left hand side is explored twice.

For $x, y \in \mathcal{C}(A)$, the sets $|x|$ and $|y|$ are arbitrary and only related to A via ∂_x and ∂_y – their specific identity is irrelevant. So configurations should be compared up to *symmetry*: a **symmetry** $\varphi : x \cong_A y$ is an order-iso s.t. $\partial_y \circ \varphi = \partial_x$. Symmetry classes of configurations are called **positions**: the set of positions on A is written $\mathcal{P}(A)$, and they are ranged over by x, y, *etc.* (note the change of font). A position x is **pointed**, written $\mathsf{x} \in \mathcal{P}_\bullet(A)$, if any of its representatives is. If $x \in \mathcal{C}(A)$, we write $\overline{x} \in \mathcal{P}(A)$ for the corresponding position. Reciprocally, if $\mathsf{x} \in \mathcal{P}(A)$, we fix $\underline{\mathsf{x}} \in \mathcal{C}(A)$ a representative. In [4], positions were shown to correspond to points in the relational model (if o is interpreted as a singleton).

If $x \in \mathcal{C}(A)$ and $y \in \mathcal{C}(B)$, then $x \otimes y \in \mathcal{C}(A \otimes B)$ has events the disjoint union $|x| + |y|$, and display map inherited. We define $x \vdash y \in \mathcal{C}(A \vdash B)$ similarly.

Augmentations. We finally define our representation of plays up to homotopy:

▶ **Definition 6.** *An* ***augmentation*** *on arena* A *is a tuple* $\mathsf{q} = \langle |\mathsf{q}|, \leq_{\llparenthesis \mathsf{q} \rrparenthesis}, \leq_\mathsf{q}, \partial_\mathsf{q} \rangle$*, where* $\llparenthesis \mathsf{q} \rrparenthesis = \langle |\mathsf{q}|, \leq_{\llparenthesis \mathsf{q} \rrparenthesis}, \partial_\mathsf{q} \rangle \in \mathcal{C}(A)$*, and* $\langle |\mathsf{q}|, \leq_\mathsf{q} \rangle$ *is a forest satisfying:*

rule-abiding: *for all* $a_1, a_2 \in |\mathsf{q}|$*, if* $a_1 \leq_{\llparenthesis \mathsf{q} \rrparenthesis} a_2$*, then* $a_1 \leq_\mathsf{q} a_2$*,*
courteous: *for all* $a_1 \rightarrowtriangle_\mathsf{q} a_2$*, if* $\mathrm{pol}(a_1) = +$ *or* $\mathrm{pol}(a_2) = -$*, then* $a_1 \rightarrowtriangle_{\llparenthesis \mathsf{q} \rrparenthesis} a_2$*,*
deterministic: *for all* $a^- \rightarrowtriangle_\mathsf{q} a_1^+$ *and* $a^- \rightarrowtriangle_\mathsf{q} a_2^+$*, then* $a_1 = a_2$*,*
$+$-covered: *for all* $a \in |\mathsf{q}|$ *maximal in* q*, we have* $\mathrm{pol}(a) = +$*,*
negative: *for all* $a \in \min(\mathsf{q})$*, we have* $\mathrm{pol}(a) = -$*,*

we then write $\mathsf{q} \in \mathsf{Aug}(A)$*, and call* $\llparenthesis \mathsf{q} \rrparenthesis \in \mathcal{C}(A)$ *the* ***desequentialization*** *of* q*.*
Finally, q *is* ***pointed*** *if it has a unique minimal event, written* $\mathsf{q} \in \mathsf{Aug}_\bullet(A)$*.*

Figure 6 represents an augmentation q by showing both relations $\rightarrowtriangle_{\llparenthesis \mathsf{q} \rrparenthesis}$ (as dotted lines) and $\rightarrowtriangle_\mathsf{q}$. An augmentation $\mathsf{q} \in \mathsf{Aug}(A)$ *augments* a configuration $x \in \mathcal{C}(A)$ by specifying causal constraints imposed by the term: for each event, the augmentation gives the necessary conditions before it can be played. Augmentations are analogous to plays: *plays* in the Hyland-Ong sense can be recovered via the alternating linearizations of augmentations [4].

Isogmentations. Augmentations are also considered up to iso. An **isomorphism** $\varphi : \mathsf{q} \cong \mathsf{p}$ is a bijection preserving and reflecting all structure. An **isogmentation** is an isomorphism class of augmentations, ranged over by $\mathbf{q}, \mathbf{p}$, *etc.*: we write $\mathsf{Isog}(A)$ (resp. $\mathsf{Isog}_\bullet(A)$) for isogmentations (resp. *pointed* isogmentations). If $\mathsf{q} \in \mathsf{Aug}(A)$, we write $\overline{\mathsf{q}} \in \mathsf{Isog}(A)$ for its isomorphism class; reciprocally, if $\mathbf{q} \in \mathsf{Isog}(A)$, we fix a representative $\underline{\mathbf{q}} \in \mathbf{q}$.

3.2 Isogmentations are Normal Resource Terms

Now we spell out the link between isogmentations and normal resource terms. We first show how the structure of each syntactic kind of terms is reflected by augmentations of the appropriate type. The main result (Theorem 12) follows directly.

Tensors and sequences. To reflect the syntactic formation rule for sequences, we show that isogmentations on $A_1 \otimes \ldots \otimes A_n$ *are* tuples. Consider $-$-arenas $\Gamma, A_1, \ldots, A_n$, and $\mathsf{q}_i \in \mathsf{Aug}(\Gamma \vdash A_i)$ for $1 \leq i \leq n$. We set $\vec{\mathsf{q}} = \langle \mathsf{q}_i \mid 1 \leq i \leq n\rangle \in \mathsf{Aug}(\Gamma \vdash \otimes_{1\leq i \leq n} A_i)$ with

$$|\vec{\mathsf{q}}| = \sum_{i=1}^{n} |\mathsf{q}_i|\,, \qquad \left\{ \begin{array}{lcll} \partial_{\vec{\mathsf{q}}}(i,m) & = & (1,g) & \text{if } \partial_{\mathsf{q}_i}(m) = (1,g), \\ \partial_{\vec{\mathsf{q}}}(i,m) & = & (2,(i,a)) & \text{if } \partial_{\mathsf{q}_i}(m) = (2,a), \end{array} \right.$$

with the two orders $\leq_{\vec{\mathsf{q}}}$ and $\leq_{\llparenthesis \vec{\mathsf{q}} \rrparenthesis}$ inherited. It is immediate that this construction preserves isomorphisms, so that it extends to isogmentations.

▶ **Proposition 7.** *There is a bijection* $\langle -, \ldots, - \rangle : \prod_{i=1}^{n} \mathsf{Isog}(\Gamma \vdash A_i) \simeq \mathsf{Isog}(\Gamma \vdash \otimes_{1 \leq i \leq n} A_i)$.

Proof. By *negative* and *forestial*, any $\mathsf{q} \in \mathsf{Aug}(\Gamma \vdash \otimes_{1\leq i\leq n} A_i)$ is isomorphic to some $\langle \mathsf{q}_i \mid 1 \leq i \leq n\rangle$; this is compatible with isos as they respect display maps. ◀

Bags and pointedness. Likewise, isogmentations are *bags of pointed isogmentations.*

We start by showing the corresponding construction. Consider $-$-arenas Γ and A, and $\mathsf{q}_1, \mathsf{q}_2 \in \mathsf{Aug}(\Gamma \vdash A)$. We set $\mathsf{q}_1 * \mathsf{q}_2 \in \mathsf{Aug}(\Gamma \vdash A)$ with events $|\mathsf{q}_1 * \mathsf{q}_2| = |\mathsf{q}_1| + |\mathsf{q}_2|$, and display $\partial_{\mathsf{q}_1 * \mathsf{q}_2}(i,m) = \partial_{\mathsf{q}_i}(m)$, and the two orders $\leq_{\mathsf{q}_1 * \mathsf{q}_2}$ and $\leq_{\llparenthesis \mathsf{q}_1 * \mathsf{q}_2 \rrparenthesis}$ inherited. This generalizes to an n-ary operation in the obvious way, which preserves isomorphisms. The operation induced on isogmentations is associative and admits as neutral element the empty isogmentation $1 \in \mathsf{Isog}(\Gamma \vdash A)$ with (a unique representative with) no event.

▶ **Proposition 8.** *There is a bijection* $- * \cdots * - : \mathcal{B}(\mathsf{Isog}_\bullet(\Gamma \vdash A)) \simeq \mathsf{Isog}(\Gamma \vdash A)$.

Proof. As $\mathsf{q} \in \mathsf{Aug}(\Gamma \vdash A)$ is a finite forest, it is isomorphic to a bag of trees. ◀

Currying. For $-$-arenas Γ, A and B, we have $\Lambda_{\Gamma,A,B} : \mathsf{Aug}(\Gamma \otimes A \vdash B) \simeq \mathsf{Aug}(\Gamma \vdash A \Rightarrow B)$ a bijection compatible with isos, which leaves the core of the augmentation unchanged and only reassigns the display map in the unique sensible way. Hence we obtain:

▶ **Proposition 9.** *For every $-$-arenas $\Gamma, A_1, \ldots, A_n$, there is*

$$\Lambda_{\Gamma,\vec{A}} : \mathsf{Isog}_\bullet(\Gamma \otimes A_1 \otimes \ldots \otimes A_n \vdash o) \simeq \mathsf{Isog}_\bullet(\Gamma \vdash A_1 \Rightarrow \ldots \Rightarrow A_n \Rightarrow o).$$

Head occurrence. The above cases handle syntactic kinds Seq, Bag and Val, following the rules (*seq*), (*bag*) and (*abs*) of the type system of Figure 3. It remains to treat rule (*hv*), *i.e.* to study the kind Base when the function subterm is a variable occurrence.

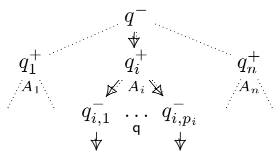


Figure 13 Illustration of $\Box_i(\mathsf{q})$.

$$\begin{array}{lclcl}
\|-\| & : & \mathsf{Val}_{\mathsf{nf}}(\Gamma; F) & \simeq & \mathsf{Isog}_\bullet(\llbracket\Gamma\rrbracket \vdash \llbracket F\rrbracket) \\
\|-\| & : & \mathsf{Base}_{\mathsf{nf}}(\Gamma) & \simeq & \mathsf{Isog}_\bullet(\llbracket\Gamma\rrbracket \vdash o) \\
\|-\| & : & \mathsf{Bag}_{\mathsf{nf}}(\Gamma; F) & \simeq & \mathsf{Isog}(\llbracket\Gamma\rrbracket \vdash \llbracket F\rrbracket) \\
\|-\| & : & \mathsf{Seq}_{\mathsf{nf}}(\Gamma; \vec{F}) & \simeq & \mathsf{Isog}(\llbracket\Gamma\rrbracket \vdash \llbracket\vec{F}\rrbracket)
\end{array}$$

Figure 14 Four bijections.

As above, we start with the corresponding construction on augmentations. We write $\vec{B} \Rightarrow o \overset{\mathsf{def}}{=} B_1 \Rightarrow \ldots \Rightarrow B_p \Rightarrow o$ for $\vec{B} = \langle B_1, \ldots, B_n\rangle$ a tuple of objects, and $\vec{B}^\otimes \overset{\mathsf{def}}{=} B_1 \otimes \cdots \otimes B_n$. Consider $\Gamma = A_1 \otimes \ldots \otimes A_n$ where each A_i is $A_i = \vec{B_i} \Rightarrow o \cong \vec{B}_i^\otimes \Rightarrow o$; consider also $\mathsf{q} \in \mathsf{Aug}(\Gamma \vdash \vec{B}_i^\otimes)$. The ***i*-lifting of q**, written $\Box_i(q) \in \mathsf{Aug}_\bullet(\Gamma \vdash o)$, is the augmentation that after the initial Opponent move, starts by playing the initial move in A_i, then proceeds as q. More precisely:

▶ **Definition 10.** *Consider $\Gamma = A_1 \otimes \ldots \otimes A_n$, with*

$$A_i = B_{i,1} \Rightarrow \ldots \Rightarrow B_{i,p_i} \Rightarrow o \cong \vec{B}_i^\otimes \Rightarrow o\,,$$

*writing $\vec{B}_i^\otimes = B_{i,1} \otimes \ldots \otimes B_{i,p_i}$; consider also $\mathsf{q} \in \mathsf{Aug}(\Gamma \vdash \vec{B}_i^\otimes)$. The **i-lifting of q**, written $\Box_i(\mathsf{q}) \in \mathsf{Aug}_\bullet(\Gamma \vdash o)$, has partial order q prefixed with two additional moves,* i.e. $\ominus \rightarrowtriangle \oplus \rightarrowtriangle \mathsf{q}$. *Its static causality is the least partial order containing dependencies*

$$\begin{array}{lcll}
m & \leq_{(\!|\Box_i(\mathsf{q})|\!)} & n & \text{for } m, n \in |\mathsf{q}| \text{ with } m \leq_{(\!|\mathsf{q}|\!)} n, \\
\oplus & \leq_{(\!|\Box_i(\mathsf{q})|\!)} & m & \text{for all } m \in |\mathsf{q}| \text{ with } \partial_{\mathsf{q}}(m) = (2, -),
\end{array}$$

and with display map given by the following clauses:

$$\begin{array}{lcll}
\partial_{\Box_i(\mathsf{q})}(\ominus) & = & (2, q) & \\
\partial_{\Box_i(\mathsf{q})}(\oplus) & = & (1, (i, (2, q))) & \\
\partial_{\Box_i(\mathsf{q})}(m) & = & (1, a) & \text{if } \partial_{\mathsf{q}}(m) = (1, a), \\
\partial_{\Box_i(\mathsf{q})}(m) & = & (1, (i, (1, a))) & \text{if } \partial_{\mathsf{q}}(m) = (2, a),
\end{array}$$

altogether defining $\Box_i(\mathsf{q}) \in \mathsf{Aug}_\bullet(\Gamma \vdash o)$ as required.

We illustrate this in Figure 13. This construction again preserves isomorphisms, and extends to give, for any $\mathbf{q} \in \mathsf{Isog}(\Gamma \vdash \vec{B}_i^\otimes)$, its ***i*-lifting** $\Box_i(\mathbf{q}) \in \mathsf{Isog}_\bullet(\Gamma \vdash o)$. Additionally:

▶ **Proposition 11.** *Consider $\Gamma, A_1, \ldots, A_n$ $-$-arenas and assume $A_1, \ldots, A_n$ are as above. We have a bijection: $\Box : \sum_{1 \leq i \leq n} \mathsf{Isog}(\Gamma \vdash \vec{B}_i^\otimes) \simeq \mathsf{Isog}_\bullet(\Gamma \vdash o)$.*

Proof. Any $p \in \mathsf{Aug}_\bullet(\Gamma \vdash o)$ has a unique initial move, which cannot be maximal by +-*covered*. By *determinism*, there is a unique subsequent Player move, displayed to the initial move of some A_i. The subsequent moves directly inform $\mathsf{q} \in \mathsf{Aug}(\Gamma \vdash \vec{B}_i^\otimes)$ *s.t.* $\mathsf{p} \cong \Box_i(\mathsf{q})$. ◀

▶ **Theorem 12.** *For Γ a context and F a type, there are bijections as in Figure 14.*

3.3 Strategies and Composition

Next we extend this correspondence to the *dynamics* of resource terms, linking syntactic substitution with an adequate notion of *composition* of augmentations.

Consider A, B and C three $-$-arenas, and fix two augmentations $\mathsf{q} \in \mathsf{Aug}(A \vdash B)$, $\mathsf{p} \in \mathsf{Aug}(B \vdash C)$. We shall compose them via *interaction*, followed by *hiding*.

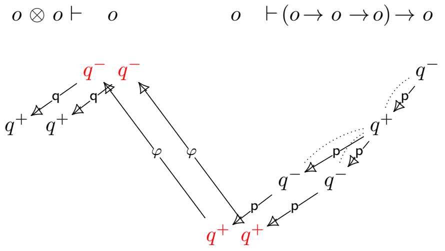

$o \otimes o \vdash \quad o$
$o \quad \vdash (o \to o \to o) \to o$
q^- q^-
q^+ q^+
q^+ q^+
q^- q^-
q^-
q^+
q
p
φ

Figure 15 Construction of an interaction $\mathsf{p} \circledast_\varphi \mathsf{q}$.

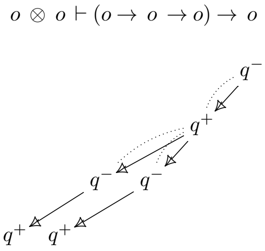


Figure 16 $\mathsf{p} \odot_\varphi \mathsf{q}$.

Interaction of augmentations. We can only compose q and p provided they reach the same state on B, so we first extract this via their desequentializations: observe $\llparenthesis \mathsf{q} \rrparenthesis \in \mathscr{C}(A \vdash B)$ has form $x_A^{\mathsf{q}} \vdash x_B^{\mathsf{q}}$; likewise we write $\llparenthesis \mathsf{p} \rrparenthesis = x_B^{\mathsf{p}} \vdash x_C^{\mathsf{p}} \in \mathscr{C}(B \vdash C)$. But what does it mean to "reach the same state"? In general $x_B^{\mathsf{q}} = x_B^{\mathsf{p}}$ is too much: it means q and p not only agree on a common state, but also on its irrelevant *concrete representation*. States in B are not configurations, but *positions*: symmetry classes of configurations. Thus q and p are **compatible** if x_B^{q} and x_B^{p} are **symmetric**, *i.e.* if there is $\varphi : x_B^{\mathsf{q}} \cong_B x_B^{\mathsf{p}}$ – we write $x_B^{\mathsf{q}} \cong_B x_B^{\mathsf{p}}$ for the equivalence. Accordingly, we must define the composition of two compatible augmentations *along with* a mediating symmetry. We first form *interactions*:

▶ **Proposition 13.** *For* q, p *as above and* $\varphi : x_B^{\mathsf{q}} \cong_B x_B^{\mathsf{p}}$*, setting* $|\mathsf{p} \circledast_\varphi \mathsf{q}| = |\mathsf{q}| + |\mathsf{p}|$ *with*

$$\begin{array}{rcl} \rhd_{\mathsf{q}} & = & \{((1,m),(1,m')) \mid m <_{\mathsf{q}} m'\}\,, \\ \rhd_{\mathsf{p}} & = & \{((2,m),(2,m')) \mid m <_{\mathsf{p}} m'\}\,, \\ \rhd_{\varphi} & = & \{((1,m),(2,\varphi(m))) \mid m \in x_B^{\mathsf{q}} \ \&\ \mathrm{pol}_{A \vdash B}(\partial_{\mathsf{q}}(m)) = +\} \\ & \cup & \{((2,\varphi(m)),(1,m)) \mid m \in x_B^{\mathsf{q}} \ \&\ \mathrm{pol}_{B \vdash C}(\partial_{\mathsf{p}}(m)) = +\}\,, \end{array}$$

then $\rhd = \rhd_{\mathsf{q}} \cup \rhd_{\mathsf{p}} \cup \rhd_\varphi$ *is acyclic: its transitive closure is a strict partial order on* $|\mathsf{p} \circledast_\varphi \mathsf{q}|$.

We write $\leq_{\mathsf{p}\circledast_\varphi\mathsf{q}} \overset{\mathsf{def}}{=} \rhd^*$ for the reflexive and transitive closure of $\rhd$. The acyclicity of $\rhd$ corresponds to a subtle and fundamental property of innocent strategies: they always have a deadlock-free interaction. Our proof is a direct adaptation of a similar fact in concurrent games on event structures [10]. Figure 15 illustrates the construction of an interaction. The two augmentations $\mathsf{q} \in \mathsf{Aug}(o \otimes o \vdash o)$ – on the left hand side – and $\mathsf{p} \in \mathsf{Aug}(o \vdash (o \to o \to o) \to o)$ – on the right hand side – are shown with their common interface in red, with a symmetry $\varphi : qq \cong_o qq$ bridging them.

Composing augmentations. We compose q and p *via* φ, by *hiding* the interaction.

▶ **Proposition 14.** *Write* $\llparenthesis \mathsf{q} \rrparenthesis = x_A^{\mathsf{q}} \vdash x_B^{\mathsf{q}}$*,* $\llparenthesis \mathsf{p} \rrparenthesis = x_B^{\mathsf{p}} \vdash x_C^{\mathsf{p}}$*, and* $\varphi : x_B^{\mathsf{q}} \cong_B x_B^{\mathsf{p}}$*.*
Then, the structure $\mathsf{p} \odot_\varphi \mathsf{q}$ *obtained by restricting* $\mathsf{p} \circledast_\varphi \mathsf{q}$ *to events in* $x_A^{\mathsf{q}} + x_C^{\mathsf{p}}$*, with* $\partial_{\mathsf{p}\odot_\varphi\mathsf{q}}((1,m)) = \partial_{\mathsf{q}}(m)$ *and* $\partial_{\mathsf{p}\odot_\varphi\mathsf{q}}((2,m)) = \partial_{\mathsf{p}}(m)$*, is an augmentation on* $A \vdash C$*.*

The interaction in Figure 15 yields the augmentation in Figure 16, the *composition of* q *and* p *via* φ. This extends to isogmentations: $\mathbf{p} \odot_\varphi \mathbf{q} \overset{\mathsf{def}}{=} \overline{\underline{\mathbf{p}} \odot_\varphi \underline{\mathbf{q}}}$ for $\mathbf{q} \in \mathsf{Isog}(A \vdash B)$ with $\llparenthesis \underline{\mathbf{q}} \rrparenthesis = x_A^{\mathbf{q}} \vdash x_B^{\mathbf{q}}$, $\mathbf{p} \in \mathsf{Isog}(B \vdash C)$ with $\llparenthesis \underline{\mathbf{p}} \rrparenthesis = x_B^{\mathbf{p}} \vdash x_C^{\mathbf{p}}$, and $\varphi : x_B^{\mathbf{q}} \cong_B x_B^{\mathbf{p}}$.

One fact is puzzling: the composition of q and p is only defined once we have fixed a mediating $\varphi : x_B^{\mathsf{q}} \cong_B x_B^{\mathsf{p}}$, which is not unique – for instance there are exactly two symmetries $qq \cong_o qq$. Worse, the result of composition depends on the choice of φ: if Figure 15 was constructed with the symmetry $\psi : qq \cong_o qq$ swapping the two moves, we would get the variant $\mathsf{p} \odot_\psi \mathsf{q}$ of Figure 16 with the two final causal links crossed, different even up to iso.

This reminds of the syntactic substitution $(\lambda f.\, f\, x\, x)\langle [y,z]/x\rangle \to \lambda f.\, f\, y\, z + \lambda f.\, f\, z\, y$. As syntactic substitution of resource terms yields *sums* of resource terms, this suggests that composition of isogmentations should produce *sums* of isogmentations, called *strategies*.

Strategies. Roughly speaking, a *strategy* is simply a weighted sum of isogmentations.

▶ **Definition 15.** *A **strategy** on arena A is a function $\sigma : \mathsf{Isog}(A) \to \overline{\mathbb{R}}_+$, where $\overline{\mathbb{R}}_+$ is the completed half-line of non-negative reals. We then write $\sigma : A$.*

We regard $\sigma : A$ as a weighted sum $\sigma = \sum_{\mathbf{q} \in \mathsf{Isog}(A)} \sigma(\mathbf{q}) \cdot \mathbf{q}$. We lift the composition of isogmentations to strategies via the formula

$$\tau \odot \sigma \quad \overset{\mathsf{def}}{=} \quad \sum_{\mathbf{q} \in \mathsf{Isog}(A \vdash B)} \; \sum_{\mathbf{p} \in \mathsf{Isog}(B \vdash C)} \; \sum_{\varphi : x_B^{\mathbf{q}} \cong_B x_B^{\mathbf{p}}} \sigma(\mathbf{q})\tau(\mathbf{p}) \cdot (\mathbf{p} \odot_\varphi \mathbf{q}) \tag{1}$$

for $\sigma : A \vdash B$ and $\tau : B \vdash C$, *i.e.* $(\tau \odot \sigma)(\mathbf{r})$ is the sum of $\sigma(\mathbf{q})\tau(\mathbf{p})$ over all triples $\mathbf{q}, \mathbf{p}, \varphi$ *s.t.* $\mathbf{r} = \mathbf{p} \odot_\varphi \mathbf{q}$ – there are no convergence issues, as we have been careful to include $+\infty$ as a coefficient in Definition 15 (though this shall not arise in the interpretation).

Identities. We also introduce identities: *copycat strategies*, formal sums of specific isogmentations presenting typical copycat behaviour; we start by defining their concrete representatives.

Consider $x \in \mathcal{C}(A)$ on $-$-arena A. The augmentation $\mathrm{cc}_x \in \mathsf{Aug}(A \vdash A)$, called the **copycat** augmentation on x, has $(\!|\mathrm{cc}_x|\!) = x \vdash x$, and as causal order $x \vdash x$, augmented with

$$\begin{array}{llll} (1,m) & \leq_{\mathrm{cc}_x} & (2,n) & \text{if } m \leq_x n \text{ and } \mathrm{pol}_A(\partial_x(m)) = +, \\ (2,m) & \leq_{\mathrm{cc}_x} & (1,n) & \text{if } m \leq_x n \text{ and } \mathrm{pol}_A(\partial_x(m)) = -, \end{array}$$

so cc_x adds to $x \vdash x$ all immediate causal links of the form $(2,m) \rightarrowtriangle (1,m)$ for negative m, and $(1,m) \rightarrowtriangle (2,m)$ for positive m. Again, this lifts to isogmentations by setting, for $\mathsf{x} \in \mathcal{P}(A)$, the **copycat isogmentation** $\mathbf{cc}_{\mathsf{x}} \in \mathsf{Isog}(A \vdash A)$ as the isomorphism class of $\mathrm{cc}_{\underline{\mathsf{x}}}$.

The strategy $\mathrm{id}_A : A \vdash A$ should have the isogmentation $\mathbf{cc}_{\mathsf{x}}$ for all position $\mathsf{x} \in \mathcal{P}(A)$. But with which coefficient? To cancel the sum over all symmetries in (1), we set:

$$\mathrm{id}_A \quad \overset{\mathsf{def}}{=} \quad \sum_{\mathsf{x} \in \mathcal{P}(A)} \frac{1}{\#\mathsf{Sym}(\mathsf{x})} \cdot \mathbf{cc}_{\mathsf{x}} \tag{2}$$

where $\mathsf{Sym}(\mathsf{x})$ is the group of *endosymmetries* of x, *i.e.* of all $\varphi : \underline{\mathsf{x}} \cong_A \underline{\mathsf{x}}$ – the cardinal of $\mathsf{Sym}(\mathsf{x})$ does not depend on the choice of $\underline{\mathsf{x}}$. This use of such a coefficient to compensate for future sums over sets of permutations is reminiscent of the Taylor expansion of λ-terms [15].

3.4 Proof of the Categorical Laws

In this section, we show the main arguments behind the following result:

▶ **Theorem 16.** *The $-$-arenas and strategies between them form a category,* Strat.

This is proved in several stages. Firstly, we establish isomorphisms corresponding to categorical laws, working concretely on augmentations – this means that these laws will refer to certain isomorphisms explicitly. Then, we show that composition of augmentations is compatible with isomorphisms, so that it carries out to isogmentations. From all that, we are in position to conclude and prove that Strat is indeed a category.

Laws on the composition of augmentations. The following lemma specifies in what sense the copycat augmentation is neutral for composition:

▶ **Lemma 17** (Neutrality). *Consider* $\mathsf{q} \in \mathsf{Aug}(A \vdash B)$*,* $x \in \mathcal{C}(B)$ *and* $\varphi : x_B^{\mathsf{q}} \cong_B x$*.*

Then, $\text{\oe}_x \odot_\varphi \mathsf{q} \cong \mathsf{q}$*. Likewise, for any* $y \in \mathcal{C}(A)$ *and* $\psi : y \cong_A x_A^{\mathsf{q}}$*, we have* $\mathsf{q} \odot_\psi \text{\oe}_y \cong \mathsf{q}$*.*

Proof. Recall that $\text{\oe}_x \odot_\varphi \mathsf{q}$ is obtained by considering $\text{\oe}_x \circledast_\varphi \mathsf{q}$ with events $|\text{\oe}_x| + |\mathsf{q}|$, *i.e.* $|\mathsf{q}| + (x + x)$ with causal order as described in Proposition 13. The composition $\text{\oe}_x \odot_\varphi \mathsf{q}$ is then the restriction to its visible events, *i.e.* $x_A^{\mathsf{q}} + (\emptyset + x)$. Then

$$|\text{\oe}_x \odot_\varphi \mathsf{q}| = x_A^{\mathsf{q}} + (\emptyset + x) \simeq x_A^{\mathsf{q}} + x \overset{x_A^{\mathsf{q}}+\varphi^{-1}}{\simeq} x_A^{\mathsf{q}} + x_B^{\mathsf{q}} \simeq |\mathsf{q}|$$

forms a bijection between the sets of events, which is checked to be an isomorphism of augmentations by a direct analysis of the causal order of $\text{\oe}_x \odot_\varphi \mathsf{q}$. ◀

It may be surprising that $\text{\oe}_x \odot_\varphi \mathsf{q} \cong \mathsf{q}$ regardless of φ: the choice of the symmetry is reflected in the isomorphism $\text{\oe}_x \odot_\varphi \mathsf{q} \cong \mathsf{q}$, which this lemma ignores. Similarly, we have:

▶ **Lemma 18** (Associativity). *Consider* $\mathsf{q} \in \mathsf{Aug}(A \vdash B)$*,* $\mathsf{p} \in \mathsf{Aug}(B \vdash C)$*,* $\mathbf{r} \in \mathsf{Aug}(C \vdash D)$*, and two symmetries* $\varphi : x_B^{\mathsf{q}} \cong_B x_B^{\mathsf{p}}$ *and* $\psi : x_C^{\mathsf{p}} \cong_C x_C^{\mathbf{r}}$*. Then*

$$\mathbf{r} \odot_{\psi'} (\mathsf{p} \odot_\varphi \mathsf{q}) \cong (\mathbf{r} \odot_\psi \mathsf{p}) \odot_{\varphi'} \mathsf{q}\,.$$

with φ', ψ' *obtained from* φ *and* ψ*, ajusting tags for disjoint unions in the obvious way.*

Proof. A routine proof, relating the two compositions to a ternary composition $\mathbf{r} \odot_\psi^3 \mathsf{p} \odot_\varphi^3 \mathsf{q} : \mathsf{Aug}(A \vdash D)$, defined in a way similar to binary composition. ◀

Congruence. Consider augmentations $\mathsf{q}, \mathsf{q}' \in \mathsf{Aug}(A \vdash B)$ with $\varphi : \mathsf{q} \cong \mathsf{q}'$, we know that φ is an isomorphism of configurations $\varphi : \llparenthesis \mathsf{q} \rrparenthesis \cong_{A \vdash B} \llparenthesis \mathsf{q}' \rrparenthesis$ – a symmetry – therefore it has the form $\varphi_A \vdash \varphi_B$, with $\varphi_A : x_A^{\mathsf{q}} \cong_A x_A^{\mathsf{q}'}$ and $\varphi_B : x_B^{\mathsf{q}} \cong_B x_B^{\mathsf{q}'}$.

▶ **Lemma 19.** *Consider* $\mathsf{q}, \mathsf{q}' \in \mathsf{Aug}(A \vdash B)$*,* $\mathsf{p}, \mathsf{p}' \in \mathsf{Aug}(B \vdash C)$*, isomorphisms* $\theta : x_B^{\mathsf{q}} \cong_B x_B^{\mathsf{p}}$ *and* $\theta' : x_B^{\mathsf{q}'} \cong_B x_B^{\mathsf{p}'}$*,* $\varphi : \mathsf{q} \cong \mathsf{q}'$ *and* $\psi : \mathsf{p} \cong \mathsf{p}'$ *such that* $\theta' \circ \varphi_B = \psi_B \circ \theta$*.*

Then, we have an isomorphism $\psi \odot_{\theta,\theta'} \varphi : \mathsf{p} \odot_\theta \mathsf{q} \cong \mathsf{p}' \odot_{\theta'} \mathsf{q}'$*.*

The proof is a direct verification that the obvious morphism between $\mathsf{p} \odot_\theta \mathsf{q}$ and $\mathsf{p}' \odot_{\theta'} \mathsf{q}'$ is indeed an isomorphism. The main consequence of this lemma is the following. Consider $\mathsf{q}, \mathsf{q}' \in \mathsf{Aug}(A \vdash B)$, $\mathsf{p}, \mathsf{p}' \in \mathsf{Aug}(B \vdash C)$, isomorphisms $\varphi : \mathsf{q} \cong \mathsf{q}'$ and $\psi : \mathsf{p} \cong \mathsf{p}'$, not requiring any commutation property as above. Still, φ and ψ project to symmetries

$$\varphi_B : x_B^{\mathsf{q}} \cong_B x_B^{\mathsf{q}'}\,, \qquad\qquad \psi_B : x_B^{\mathsf{p}} \cong_B x_B^{\mathsf{p}'}\,.$$

inducing a bijection

$$\begin{array}{rcrcl} \chi & : & x_B^{\mathsf{q}} \cong_B x_B^{\mathsf{p}} & \simeq & x_B^{\mathsf{q}'} \cong_B x_B^{\mathsf{p}'} \\ & & \theta & \mapsto & \psi_B \circ \theta \circ \varphi_B^{-1}\,, \end{array}$$

so that for any $\theta : x_B^{\mathsf{q}} \cong_B x_B^{\mathsf{p}}$, we have $\mathsf{p} \odot_\theta \mathsf{q} \cong \mathsf{p}' \odot_{\chi(\theta)} \mathsf{q}'$ by Lemma 19. It ensues that we can substitute one representative for another when summing over all mediating symmetries:

$$\sum_{\theta : x_B^{\mathsf{q}} \cong_B x_B^{\mathsf{p}}} \overline{\mathsf{p} \odot_\theta \mathsf{q}} = \sum_{\theta : x_B^{\mathsf{q}} \cong_B x_B^{\mathsf{p}}} \overline{\mathsf{p}' \odot_{\chi(\theta)} \mathsf{q}'} = \sum_{\theta : x_B^{\mathsf{q}'} \cong_B x_B^{\mathsf{p}'}} \overline{\mathsf{p}' \odot_\theta \mathsf{q}'}$$

using the observation above, and reindexing the sum following χ – or in other words, the composition of strategies does not depend on the choice of representative used for isogmentations. This is often used silently throughout the development.

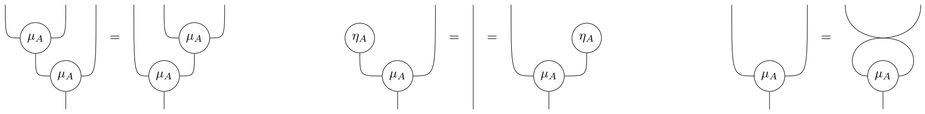


Figure 17 Monoid laws.

Categorical laws. We are now equipped to show Theorem 16. First, the identity laws:

▶ **Proposition 20.** *Consider* $\sigma : A \vdash B$*. Then,* $\mathrm{id}_B \odot \sigma = \sigma \odot \mathrm{id}_A = \sigma$*.*

Proof. We focus on $\mathrm{id}_B \odot \sigma$. For any $\mathbf{p} \in \mathsf{Isog}(B \vdash B)$, we write $(\!|\underline{\mathbf{p}}|\!) = x_1^{\mathbf{p}} \vdash x_{\mathtt{r}}^{\mathbf{p}}$. We have:

$$\begin{aligned}
\mathrm{id}_B \odot \sigma &= \sum_{\mathbf{q} \in \mathsf{Isog}(A \vdash B)} \sum_{\mathbf{p} \in \mathsf{Isog}(B \vdash B)} \sum_{\varphi : x_B^{\mathbf{q}} \cong_B x_1^{\mathbf{p}}} \sigma(\mathbf{q}) \mathrm{id}_B(\mathbf{p}) \cdot \mathbf{p} \odot_\varphi \mathbf{q} \\
&= \sum_{\mathbf{q} \in \mathsf{Isog}(A \vdash B)} \sum_{\mathsf{x} \in \mathcal{P}(B)} \sum_{\varphi : x_B^{\mathbf{q}} \cong_B \underline{\mathsf{x}}} \frac{\sigma(\mathbf{q})}{\#\mathsf{Sym}(\mathsf{x})} \cdot \mathbf{cc}_{\mathsf{x}} \odot_\varphi \mathbf{q}
\end{aligned}$$

using definition of the composition and of the identity. Next, we compute

$$\begin{aligned}
\mathrm{id}_B \odot \sigma &= \sum_{\mathbf{q} \in \mathsf{Isog}(A \vdash B)} \sum_{\mathsf{x} \in \mathcal{P}(B)} \sum_{\varphi : x_B^{\mathbf{q}} \cong_B \underline{x}} \frac{\sigma(\mathbf{q})}{\#\mathsf{Sym}(\mathsf{x})} \cdot \mathbf{q} \\
&= \sum_{\mathbf{q} \in \mathsf{Isog}(A \vdash B)} \sum_{\varphi \in \mathsf{Sym}(\overline{x_B^{\mathbf{q}}})} \frac{\sigma(\mathbf{q})}{\#\mathsf{Sym}(\overline{x_B^{\mathbf{q}}})} \cdot \mathbf{q} \\
&= \sum_{\mathbf{q} \in \mathsf{Isog}(A \vdash B)} \sigma(\mathbf{q}) \cdot \mathbf{q}
\end{aligned}$$

which is σ; by Lemma 17 and direct reasoning on symmetries – $\sigma \odot \mathrm{id}_A = \sigma$ is symmetric. ◀

Notice how the sum over all symmetries exactly compensates for the coefficient in (2). Likewise, associativity of the composition of strategies follows from Lemma 18 and bilinearity of composition, altogether concluding the proof that Strat is a category.

4 Resource Categories

We now develop *resource categories*, models of the resource calculus inspired by Strat.

4.1 Motivation and Definition

As compositions generate sums, we need an additive structure. Following [5], an **additive symmetric monoidal category (asmc)** is a symmetric monoidal category where each hom-set is a commutative monoid, and each operation preserves the additive structure.

Bialgebras. As for differential categories, resource categories build on *bialgebras*:

▶ **Definition 21.** *Consider* $\mathcal{C}$ *an additive symmetric monoidal category.*

A ***bialgebra*** *on* $\mathcal{C}$ *is* $(A, \delta_A, \epsilon_A, \mu_A, \eta_A)$ *with* (A, μ_A, η_A) *a commutative monoid (see Figure 17),* $(A, \delta_A, \epsilon_A)$ *a commutative comonoid, with additional bialgebra laws (Figure 18).*

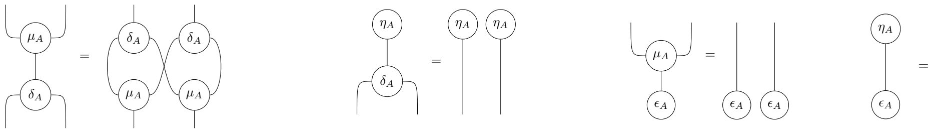


Figure 18 Additional bialgebra laws.

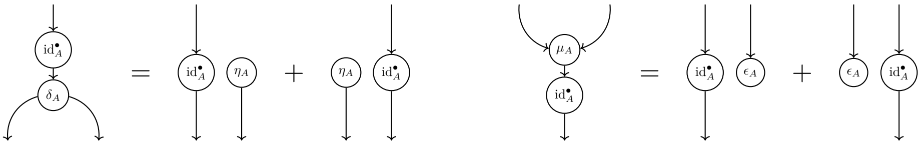


Figure 19 Laws for (co)multiplication and pointed identity.

In a resource category, all objects shall be bialgebras. This means that for each object A, we have morphisms $\delta_A : A \to A \otimes A$, $\epsilon_A : A \to I$, $\mu_A : A \otimes A \to A$, and $\eta_A : I \to A$ satisfying coherence laws [5]. Comonoids $(A, \delta_A, \epsilon_A)$ are the usual categorical description of duplicable objects. Intuitively, requests made to δ_A on either side of the tensor on the rhs, are sent to the left. Categorically the monoid structure (A, μ_A, η_A) is dual, but its intuitive behaviour is different: each request on the rhs is forwarded, non-deterministically, to either side of the tensor on the left, reflecting the *sums* arising in substitutions.

In contrast with differential categories, morphisms in a resource category intuitively correspond to (sums of) *bags* rather than *terms*. Morally, the empty bag from A to B is captured from the bialgebra structure as $\eta_B \circ \epsilon_A \in \mathcal{C}(A, B)$, written 1. Likewise, the **product** $f * g = \mu_B \circ (f \otimes g) \circ \delta_A \in \mathcal{C}(A, B)$ of $f, g \in \mathcal{C}(A, B)$ captures the union of bags. This makes $(\mathcal{C}(A, B), *, 1)$ a commutative monoid, altogether turning $\mathcal{C}(A, B)$ into a commutative semiring, though composition and tensor in $\mathcal{C}$ only preserve the additive monoid.

A bag of morphisms may be "flattened" into a morphism by the following operation: if $\bar{f} = [f_1, \dots, f_n] \in \mathcal{B}(\mathcal{C}(A, B))$, we write $\Pi\bar{f} \stackrel{\mathsf{def}}{=} f_1 * \cdots * f_n \in \mathcal{C}(A, B)$.

Pointed identities. Resource categories axiomatize categorically the *singleton bags*. For that, a pivotal role is played by the **pointed identity**, a chosen idempotent $\mathrm{id}^\bullet_A \in \mathcal{C}(A, A)$ which we think of as a singleton bag with a linear copycat behaviour. More formally:

▶ **Definition 22.** *Consider $\mathcal{C}$ an asmc where each object has a bialgebra structure.*

*For $A \in \mathcal{C}$, a **pointed identity** on A is an idempotent $\mathrm{id}^\bullet_A \in \mathcal{C}(A, A)$ satisfying the equations shown as string diagrams in Figure 19, plus $\epsilon_A \circ \mathrm{id}^\bullet_A = 0$ and $\mathrm{id}^\bullet_A \circ \eta_A = 0$.*

Those laws are reminiscent of the laws of derelictions and coderelictions in bialgebra modalities [5], except that both roles are played by $\mathrm{id}^\bullet_A$. In a resource category $\mathcal{C}$, all objects have a pointed identity. The "singleton bags" are those $f \in \mathcal{C}(A, B)$ that are **pointed**, *i.e.* $\mathrm{id}^\bullet_B \circ f = f$ – we write $\mathcal{C}_\bullet(A, B)$. Dually, we may also capture those morphisms which require *exactly one* resource: $f \in \mathcal{C}(A, B)$ is **co-pointed** if $f \circ \mathrm{id}^\bullet_A = f$, and we write $f \in \mathcal{C}^\bullet(A, B)$.

Resource categories. Altogether, we are now ready to define resource categories:

▶ **Definition 23.** *A **resource category** is an asmc $\mathcal{C}$ where each $A \in \mathcal{C}$ has a bialgebra structure $(A, \delta_A, \epsilon_A, \mu_A, \eta_A)$ and pointed identity $\mathrm{id}^\bullet_A$, such that the bialgebra structure is compatible with the monoidal structure of $\mathcal{C}$ (see Figure 20).*

$$\begin{tikzcd}
 & A \otimes B \arrow[r, "\delta_{A\otimes B}"] \arrow[dl, "\delta_A \otimes \delta_B"'] & (A \otimes B) \otimes (A \otimes B) \arrow[dr, "\alpha_{A\otimes B, A, B}"] & \\
(A \otimes A) \otimes (B \otimes B) \arrow[d, "\alpha_{A\otimes A, B, B}"'] & & & ((A \otimes B) \otimes A) \otimes B \arrow[d, "\alpha^{-1}_{A,B,A}\otimes B"] \\
((A \otimes A) \otimes B) \otimes B \arrow[dr, "\alpha^{-1}_{A,A,B}\otimes B"'] & & & (A \otimes (B \otimes A)) \otimes B \\
 & (A \otimes (A \otimes B)) \otimes B \arrow[urr, "(A\otimes\gamma_{A,B})\otimes B"] & &
\end{tikzcd}$$

$$\begin{tikzcd}
A \otimes B \arrow[r, "\epsilon_{A\otimes B}"] \arrow[d, "\epsilon_A\otimes\epsilon_B"'] & I \\
I \otimes I \arrow[ur, "\lambda_I=\rho_I"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
I \arrow[r, bend left, "\epsilon_I"] \arrow[r, bend right, "\mathrm{id}_I"'] & I
\end{tikzcd}$$

Figure 20 Compatibility of comonoids with the monoidal structure – there are symmetric conditions for the compatibility of monoids with the monoidal structure.

$$\begin{tikzcd}
A \arrow[r, "\delta_A"] \arrow[d, "\Pi \bar{f}"'] & A \otimes A \arrow[d, "\sum_{\bar{f} \lhd \bar{f}_1 * \bar{f}_2} \Pi\bar{f}_1 \otimes \Pi\bar{f}_2"] \\
B \arrow[r, "\delta_B"'] & B \otimes B
\end{tikzcd}
\qquad
\begin{tikzcd}
A \otimes A \arrow[r, "\mu_A"] \arrow[d, "\sum_{\bar{f} \lhd \bar{f}_1 * \bar{f}_2} \Pi\bar{f}_1 \otimes \Pi\bar{f}_2"'] & A \arrow[d, "\Pi \bar{f}"] \\
B \otimes B \arrow[r, "\mu_B"'] & B
\end{tikzcd}$$

Figure 21 Compatibility of bags with δ.

Figure 22 Compatibility of bags with μ.

*Additionally, $\mathcal{C}$ is **closed** if $A \otimes -$ has a right adjoint $A \multimap -$ for each $A \in \mathcal{C}$.*

This simple definition has powerful consequences. In particular, the following key property, derived from the definition of resource categories, expresses how the product of a bag of pointed morphisms interacts with the comonoid structure – and dually for the product of a bag of co-pointed morphisms and monoids. Much of the proof of invariance relies on it:

▶ **Lemma 24.** *Consider $\mathcal{C}$ a resource category, then we have the following properties:*

1. *For any bag of pointed morphisms $\bar{f} \in \mathcal{B}(\mathcal{C}_\bullet(A, B))$,*
   (a) *the diagram of Figure 21 commutes; and*
   (b) *we have $\epsilon_B \circ \Pi\bar{f} = 1$ if $\bar{f}$ is empty, 0 otherwise;*
2. *For any bag of co-pointed morphisms $\bar{f} \in \mathcal{B}(\mathcal{C}^\bullet(A, B))$,*
   (a) *the diagram of Figure 22 commutes; and*
   (b) *we have $\Pi\bar{f} \circ \eta_A = 1$ if $\bar{f}$ is empty, 0 otherwise.*

Proof. This follows from a lengthy but mostly direct diagram chase. ◀

4.2 Interpretation of the Resource Calculus

In order to describe the interpretation of the resource calculus, it will be convenient to introduce some of the combinators from the theory of cartesian closed categories:

Cartesian combinators. The **pairing** of $f \in \mathcal{C}(\Gamma, A)$ and $g \in \mathcal{C}(\Gamma, B)$ is

$$\langle f, g\rangle \overset{\mathsf{def}}{=} (f \otimes g) \circ \delta_\Gamma \in \mathcal{C}(\Gamma, A \otimes B)\,;$$

likewise $\pi_1 \overset{\mathsf{def}}{=} \rho_A \circ (A \otimes \epsilon_B) \in \mathcal{C}(A \otimes B, A)$ and $\pi_2 \overset{\mathsf{def}}{=} \lambda_B \circ (\epsilon_A \otimes B) \in \mathcal{C}(A \otimes B, B)$ are the two **projections** – we shall also use their obvious n-ary generalizations. The laws of cartesian categories fail: we have $\langle \pi_1, \pi_2\rangle = \mathrm{id}_{A\otimes B}$, but *e.g.* $\pi_1 \circ \langle f, h\rangle = f$ only holds if h is *erasable* (*i.e.* $\epsilon_B \circ h = \epsilon_\Gamma$) and $\langle f, g\rangle \circ h = \langle f \circ h, g \circ h\rangle$ if h is duplicable (*i.e.* $\delta_\Gamma \circ h = (h \otimes h) \circ \delta_\Delta$) – so we do get the usual laws if h is a *comonoid morphism* [20].

For any two objects $A, B \in \mathcal{C}$, we have $\mathrm{ev}_{A,B} \in \mathcal{C}((A \to B) \otimes A, B)$ the **evaluation morphism**. If $f \in \mathcal{C}(A \otimes B, C)$, its **currying** is written $\Lambda_{A,B,C}(f) \in \mathcal{C}(A, B \to C)$.

Lemmas on propagation of substitutions. Morphisms coming from the interpretation are not comonoid morphisms, but many structural morphisms are: for instance it follows from a direct diagram chase that projections *are* comonoid morphisms.

As explained above, comonoid morphisms propagate in tuples as in a cartesian category. But importantly, resource categories also specify how some non comonoid morphisms propagate through a pairing, even paired with a comonoid morphism:

▶ **Lemma 25.** *Let* $\bar{b} \in \mathcal{B}(\mathcal{C}_\bullet(\Delta, A))$, $h \in \mathcal{C}(\Delta, \Gamma)$, $f \in \mathcal{C}(\Gamma \otimes A, B)$, $g \in \mathcal{C}(\Gamma \otimes A, C)$. *If* $h \in \mathcal{C}(\Delta, \Gamma)$ *is a comonoid morphism, then we have:*

$$\langle f, g\rangle \circ \langle h, \Pi\bar{b}\rangle = \sum_{\bar{b} \lhd \bar{b}_1 * \bar{b}_2} \langle f \circ \langle h, \Pi\bar{b}_1\rangle, g \circ \langle h, \Pi\bar{b}_2\rangle\rangle$$

Proof. A diagram chase leveraging case *(1)* of Lemma 24. ◀

This is fairly close to how substitutions propagate through terms in the resource λ-calculus (see Section 2): we sum over all the partitions of the bag $\bar{b}$ into two components, to be distributed to the two components of the pair – when using this lemma in the proof of the substitution lemma, the comonoid morphism h shall simply be an identity leaving all the unsubstitued variables unchanged. Syntactic substitution has another important case, namely when a substitution encounters a variable occurrence. Likewise here, we have:

▶ **Lemma 26.** *Consider* $\bar{f} \in \mathcal{B}(\mathcal{C}_\bullet(A, B))$. *Then* $\mathrm{id}^\bullet_B \circ \Pi\bar{f} = v$ *if* $\bar{f} = [v]$, 0 *otherwise.*

This lemma follows from the conditions of a resource category, though in a not so straightforward way. It illustrates how the pointed identity is able to pick a single element of a bag. If the bag has too many elements or not enough, then the composition yields 0.

Interpretation. From now on, we fix a closed resource category $\mathcal{C}$ with a chosen object o.

We first set $[\![o]\!] \overset{\mathsf{def}}{=} o$, $[\![\langle F_1, \dots, F_n\rangle]\!] \overset{\mathsf{def}}{=} [\![F_1]\!] \otimes \cdots \otimes [\![F_n]\!]$ and $[\![\vec{F} \to o]\!] \overset{\mathsf{def}}{=} [\![\vec{F}]\!] \to o$. For contexts, $[\![\Gamma]\!] \overset{\mathsf{def}}{=} \bigotimes_{(x:F)\in\Gamma} [\![F]\!]$. If $(x : F) \in \Gamma$, we write $\mathrm{var}^\Gamma_x \in \mathcal{C}([\![\Gamma]\!], [\![F]\!])$ the projection. For Γ and Δ disjoint we use the iso $\Lambda\!\!\!\Lambda_{\Gamma,\Delta} \in \mathcal{C}([\![\Gamma]\!] \otimes [\![\Delta]\!], [\![\Gamma, \Delta]\!])$.

The interpretation of terms (or, rather, of typing derivations) follows the four kinds of judgements from Section 2: for $\Gamma, A \in \mathcal{C}$ and $\vec{A} = \langle A_1, \dots, A_n\rangle$, we define

$$\begin{array}{rclrcl}
\mathsf{Val}_{\mathcal{C}}(\Gamma; A) & \overset{\mathsf{def}}{=} & \mathcal{C}_\bullet(\Gamma, A) & \mathsf{Seq}_{\mathcal{C}}(\Gamma; \vec{A}) & \overset{\mathsf{def}}{=} & \Pi_{1 \le i \le n} \mathsf{Bag}_{\mathcal{C}}(\Gamma; A_i) \\
\mathsf{Base}_{\mathcal{C}}(\Gamma) & \overset{\mathsf{def}}{=} & \mathcal{C}_\bullet(\Gamma, o) & \mathsf{Bag}_{\mathcal{C}}(\Gamma; A) & \overset{\mathsf{def}}{=} & \mathcal{B}(\mathsf{Val}_{\mathcal{C}}(\Gamma; A))\,.
\end{array}$$

Notably, sequences and bags are interpreted as *actual* sequences and bags at the "meta-level", rather than via the "internal" bags (*i.e.* products of pointed maps) or products (*i.e.* via the monoidal structure) in $\mathcal{C}$. This apparent duplication of structure is resolved when interpreting applications: we set $\langle\!|\vec{f}|\!\rangle \overset{\mathsf{def}}{=} \langle \Pi\bar{f}_1, \dots, \Pi\bar{f}_n\rangle \in \mathcal{C}(\Gamma, \vec{A}^{\otimes})$ for $\vec{f} = \langle \bar{f}_1, \dots, \bar{f}_n\rangle \in \mathsf{Seq}_{\mathcal{C}}(\Gamma, \vec{A})$, called the **packing** of the sequence $\vec{f}$.

Like bags, packed sequences distribute over pairs and products:

▶ **Lemma 27.** *Let* $\vec{c} \in \mathsf{Seq}_{\mathcal{C}}(\Delta; \vec{A})$, $h \in \mathcal{C}(\Delta, \Gamma)$, $f \in \mathcal{C}(\Gamma \otimes \vec{A}, B)$, $g \in \mathcal{C}(\Gamma \otimes \vec{A}, C)$. *If* $h \in \mathcal{C}(\Delta, \Gamma)$ *is a comonoid morphism, then we have:*

$$\langle f, g\rangle \circ \langle h, \langle\!|\vec{c}|\!\rangle\rangle = \sum_{\vec{c} \lhd \vec{c}_1 * \vec{c}_2} \langle f \circ \langle h, \langle\!|\vec{c}_1|\!\rangle\rangle, g \circ \langle h, \langle\!|\vec{c}_2|\!\rangle\rangle\rangle\,.$$

$$\begin{array}{rcl}
[\![\Gamma \vdash_{\mathsf{Val}} \lambda\vec{x}.s : \vec{F} \to o]\!] & = & \Lambda_{[\![\Gamma]\!],[\![\vec{x}:\vec{F}]\!],o}([\![\Gamma, \vec{x} : \vec{F} \vdash_{\mathsf{Base}} s : o]\!] \circ \mathbin{\wedge\!\!\!\wedge}_{[\![\Gamma]\!],[\![\vec{x}:\vec{F}]\!]}) \\
[\![\Gamma \vdash_{\mathsf{Base}} x\,\vec{t} : o]\!] & = & \mathrm{ev}_{[\![\vec{F}]\!],o} \circ \langle \mathrm{id}^{\bullet}_{[\![F]\!]} \circ \mathrm{var}^{\Gamma}_{x}, \langle\!|[\![\Gamma \vdash_{\mathsf{Seq}} \vec{t} : \vec{F}]\!]|\!\rangle\rangle \\
[\![\Gamma \vdash_{\mathsf{Base}} s\,\vec{t} : o]\!] & = & \mathrm{ev}_{[\![\vec{F}]\!],o} \circ \langle [\![\Gamma \vdash_{\mathsf{Val}} s : \vec{F} \to o]\!], \langle\!|[\![\Gamma \vdash_{\mathsf{Seq}} \vec{t} : \vec{F}]\!]|\!\rangle\rangle \\
[\![\Gamma \vdash_{\mathsf{Bag}} [s_1, \ldots, s_n] : F]\!] & = & [[\![\Gamma \vdash_{\mathsf{Val}} s_i : F]\!] \mid 1 \leq i \leq n] \\
[\![\Gamma \vdash_{\mathsf{Seq}} \langle \bar{s}_1, \ldots, \bar{s}_n\rangle : \vec{F}]\!] & = & \langle [\![\Gamma \vdash_{\mathsf{Bag}} \bar{s}_i : F_i]\!] \mid 1 \leq i \leq n\rangle
\end{array}$$

Figure 23 Interpretation of the resource calculus.

Proof. Proved by iterating Lemma 25, for each component of the sequence. ◀

We also have a similar lemma for a substitution propagating through a product:

▶ **Lemma 28.** *Consider* $f, g \in \mathcal{C}(\Gamma \otimes A, B)$*,* $h \in \mathcal{C}(\Delta, \Gamma)$ *a comonoid morphism, and* $\vec{c} \in \mathsf{Seq}_{\mathcal{C}}(\Delta, \vec{A})$*. Then,*

$$(f * g) \circ \langle h, \langle\!|\vec{c}|\!\rangle\rangle = \sum_{\vec{c} \lhd \vec{c}_1 * \vec{c}_2} (f \circ \langle h, \langle\!|\vec{c}_1|\!\rangle\rangle) * (g \circ \langle h, \langle\!|\vec{c}_2|\!\rangle\rangle)\,,$$

and $1 \circ \langle h, \langle\!|\vec{c}|\!\rangle\rangle = 1$ *if* $\vec{c}$ *is empty,* 0 *otherwise.*

Proof. Similar to Lemma 27. ◀

We now define the four interpretation functions

$$\begin{array}{rclcrcl}
\mathsf{Val}(\Gamma; F) & \to & \mathsf{Val}_{\mathcal{C}}([\![\Gamma]\!]; [\![F]\!]) & \qquad & \mathsf{Bag}(\Gamma; F) & \to & \mathsf{Bag}_{\mathcal{C}}([\![\Gamma]\!]; [\![F]\!]) \\
\mathsf{Base}(\Gamma) & \to & \mathsf{Base}_{\mathcal{C}}([\![\Gamma]\!]) & & \mathsf{Seq}(\Gamma; \vec{F}) & \to & \mathsf{Seq}_{\mathcal{C}}([\![\Gamma]\!]; [\![\vec{F}]\!])
\end{array}$$

all written $[\![-]\!]$, by mutual induction, as in Figure 23. The interpretation is extended to sums of terms $\Sigma\mathsf{Val}(\Gamma; F) \to \mathsf{Val}_{\mathcal{C}}([\![\Gamma]\!]; [\![F]\!])$ and $\Sigma\mathsf{Base}(\Gamma) \to \mathsf{Base}_{\mathcal{C}}([\![\Gamma]\!])$ relying on the additive structure of $\mathcal{C}$ – we give no interpretation to sums of bags or sequences.

4.3 The Soundness Theorem

We show that this interpretation is invariant under reduction. The bulk of the proof consists in proving a suitable substitution lemma, for which we must first give a semantic account of substitution. We define three semantic substitution functions:

$$\begin{array}{rcrcl}
-\langle\!\langle -/\vec{x}\rangle\!\rangle & : & \mathsf{Val}_{\mathcal{C}}([\![\Gamma, \vec{x} : \vec{F}]\!]; [\![G]\!]) \times \mathsf{Seq}_{\mathcal{C}}([\![\Gamma]\!]; [\![\vec{F}]\!]) & \to & \mathsf{Val}_{\mathcal{C}}([\![\Gamma]\!]; [\![G]\!]) \\
-\langle\!\langle -/\vec{x}\rangle\!\rangle & : & \mathsf{Base}_{\mathcal{C}}([\![\Gamma, \vec{x} : \vec{F}]\!]) \times \mathsf{Seq}_{\mathcal{C}}([\![\Gamma]\!]; [\![\vec{F}]\!]) & \to & \mathsf{Base}_{\mathcal{C}}([\![\Gamma]\!]) \\
-\langle\!\langle -/\vec{x}\rangle\!\rangle & : & \mathsf{Seq}_{\mathcal{C}}([\![\Gamma, \vec{x} : \vec{F}]\!]; [\![\vec{G}]\!]) \times \mathsf{Seq}_{\mathcal{C}}([\![\Gamma]\!]; [\![\vec{F}]\!]) & \to & \mathcal{C}([\![\Gamma]\!], [\![\vec{G}]\!])
\end{array}$$

using our cartesian-like notations:

$$f\langle\!\langle \vec{g}/\vec{x}\rangle\!\rangle \overset{\mathsf{def}}{=} f \circ \mathbin{\wedge\!\!\!\wedge}_{[\![\Gamma]\!],[\![\vec{x}:\vec{F}]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, \langle\!|\vec{g}|\!\rangle\rangle \qquad \vec{f}\langle\!\langle \vec{g}/\vec{x}\rangle\!\rangle \overset{\mathsf{def}}{=} \langle\!|\vec{f}|\!\rangle \circ \mathbin{\wedge\!\!\!\wedge}_{[\![\Gamma]\!],[\![\vec{x}:\vec{F}]\!]} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, \langle\!|\vec{g}|\!\rangle\rangle$$

where the first applies for $f \in \mathsf{Val}_{\mathcal{C}}([\![\Gamma, \vec{x} : \vec{F}]\!]; [\![G]\!])$ or $f \in \mathsf{Base}_{\mathcal{C}}([\![\Gamma, \vec{x} : \vec{F}]\!])$ and the second for $\vec{f} \in \mathsf{Seq}_{\mathcal{C}}([\![\Gamma, \vec{x} : \vec{F}]\!]; [\![\vec{G}]\!])$. We may now state the substitution lemma:

▶ **Lemma 29.** *Consider* $\vec{t} \in \mathsf{Seq}(\Gamma, \vec{F})$*,* $\Delta = \Gamma, \vec{x} : \vec{F}$ *and* $s \in \mathsf{Val}(\Delta, G)$ *or* $s \in \mathsf{Base}(\Delta)$*. Then,* $[\![s\langle\vec{t}/\vec{x}\rangle]\!] = [\![s]\!]\langle\!\langle [\![\vec{t}]\!]/\vec{x}\rangle\!\rangle$*.*

Proof. We show the stronger statement that for all $\vec{g} \in \mathsf{Seq}(\Gamma, \vec{F})$, and $\Delta = \Gamma, \vec{x} : \vec{F}$,

(1) If $f \in \mathsf{Val}(\Delta; G)$, then $[\![f\langle\vec{g}/\vec{x}\rangle]\!] = [\![f]\!]\langle\!\langle[\![\vec{g}]\!]/\vec{x}\rangle\!\rangle$.

(2) If $f \in \mathsf{Base}(\Delta)$, then $[\![f\langle\vec{g}/\vec{x}\rangle]\!] = [\![f]\!]\langle\!\langle[\![\vec{g}]\!]/\vec{x}\rangle\!\rangle$.

(3) Assume $\vec{f} \in \mathsf{Seq}(\Delta; \vec{G})$ with $\vec{f}\langle\vec{g}/\vec{x}\rangle = \sum_{1\leq i\leq n} \vec{f_i}$, for $\vec{f_i} \in \mathsf{Seq}(\Gamma; \vec{G})$. Then, $\sum_{1\leq i\leq n} (\![[\![\vec{f_i}]\!]]\!) = [\![\vec{f}]\!]\langle\!\langle[\![\vec{g}]\!]/\vec{x}\rangle\!\rangle$;

which follows by induction on typing derivations, using all our lemmas above. ◀

From the substitution lemma above, we may easily deduce:

▶ **Theorem 30.** *If $S \in \Sigma\mathsf{Val}(\Gamma; F)$ and $S \Rightarrow S'$ then $[\![S]\!] = [\![S']\!]$.*

Proof. Preservation of β-reduction follows from Lemma 29. To show that this extends by context closure, we prove the three statements:

(1) If $s \in \mathsf{Val}(\Gamma; F)$ and $s \Rightarrow S'$ then $[\![s]\!] = [\![S']\!]$,

(2) If $s \in \mathsf{Base}(\Gamma)$ and $s \Rightarrow S'$ then $[\![s]\!] = [\![S']\!]$,

(3) If $\vec{s} \in \mathsf{Seq}(\Gamma; \vec{F})$ and $\vec{s} \Rightarrow \sum_{i\in I} \vec{s}_i$ then $(\![[\![\vec{s}]\!]]\!) = \sum_{i\in I} (\![[\![\vec{s}_i]\!]]\!)$.

by mutual induction, following the inductive definition of context closure. Finally, it is immediate that this extends to sums as required. ◀

5 Game Semantics as a Resource Category

It remains to check that Strat is indeed a resource category, and that the induced interpretation of normal forms coincides with the bijections from Theorem 12.

5.1 Additive Symmetric Monoidal Structure

Tensor. As for composition we first define the tensor of augmentations, then isogmentations, then strategies. For A_i, B_i arenas with $\mathsf{q}_i \in \mathsf{Aug}(A_i \vdash B_i)$ for $i = 1, 2$, we set $\mathsf{q}_1 \otimes \mathsf{q}_2 \in \mathsf{Aug}(A_1 \otimes A_2 \vdash B_1 \otimes B_2)$ with $|\mathsf{q}_1 \otimes \mathsf{q}_2| = |\mathsf{q}_1| + |\mathsf{q}_2|$ and $\partial_{\mathsf{q}_1\otimes\mathsf{q}_2}(i, m) = (j, (i, n))$ if $\partial_{\mathsf{q}_i}(m) = (j, n)$, and the orders $\leq_{\mathsf{q}_1\otimes\mathsf{q}_2}$ and $\leq_{(\!|\mathsf{q}_1\otimes\mathsf{q}_2|\!)}$ inherited. This construction preserves isomorphisms, hence the tensor $\mathbf{q}_1 \otimes \mathbf{q}_2 \in \mathsf{Isog}(A_1 \otimes A_2 \vdash B_1 \otimes B_2)$ may be defined via any representative – for definiteness, we use the chosen representatives of $\mathbf{q}_1$ and $\mathbf{q}_2$. We lift the definition to strategies with, for $\sigma_1 : \Gamma_1 \vdash A_1$ and $\sigma_2 : \Gamma_2 \vdash A_2$:

$$\sigma_1 \otimes \sigma_2 \overset{\mathsf{def}}{=} \sum_{\mathbf{q}_1\in\mathsf{Isog}(A_1\vdash B_1)} \sum_{\mathbf{q}_2\in\mathsf{Isog}(A_2\vdash B_2)} \sigma_1(\mathbf{q}_1)\,\sigma_2(\mathbf{q}_2) \cdot (\mathbf{q}_1 \otimes \mathbf{q}_2)\ .$$

Structural morphisms. Structural morphisms are all variations of copycat. As we did for copycat itself, we start with concrete representatives. Consider A, B, C arenas, and $x \in \mathcal{C}(A)$, $y \in \mathcal{C}(B)$, $z \in \mathcal{C}(C)$. Noting $\varnothing$ the empty configuration on 1, we set:

$$\begin{array}{rclrcl} (\!|\lambda^x_A|\!) & = & \varnothing \otimes x \vdash x\,, & (\!|\alpha^{x,y,z}_{A,B,C}|\!) & = & x \otimes (y \otimes z) \vdash (x \otimes y) \otimes z\,, \\ (\!|\rho^x_A|\!) & = & x \otimes \varnothing \vdash x\,, & (\!|\gamma^{x,y}_{A,B}|\!) & = & x \otimes y \vdash y \otimes x\,. \end{array}$$

and λ^x_A, ρ^x_A, $\alpha^{x,y,z}_{A,B,C}$ and $\gamma^{x,y}_{A,B}$ are defined from these, augmented with the obvious copycat behaviour.

We lift this to isogmentations: for $\mathsf{x} \in \mathcal{P}(A)$, $\boldsymbol{\lambda}^{\mathsf{x}}_A$ is the isomorphism class of λ^{x}_A; and likewise for the others. Then the strategy λ_A is defined as for id_A in (2) (page 12) and likewise for ρ_A, $\alpha_{A,B,C}$ and $\gamma_{A,B}$. These structural morphisms satisfy the necessary conditions to make $(\mathsf{Strat}, \otimes, 1)$ a symmetric monoidal category.

Additive Structure. The *sum* of strategies is defined pointwise, and 0 is the sum with coefficients all null. All compatibility conditions are direct, making Strat an asmc.

5.2 Resource Category Structure

Bialgebra. For the strategies for (co)multiplication, we first set configurations $(\!|\delta_A^{x,y}|\!) = x * y \vdash x \otimes y$ and $(\!|\mu_A^{x,y}|\!) = x \otimes y \vdash x * y$ for any A and $x, y \in \mathcal{C}(A)$; $\delta_A^{x,y}$, $\mu_A^{x,y}$ are obtained by adjoining copycat behaviour on x and y. This lifts to isogmentations $\boldsymbol{\delta}_A^{\mathsf{x},\mathsf{y}}$ and $\boldsymbol{\mu}_A^{\mathsf{x},\mathsf{y}}$ for $\mathsf{x}, \mathsf{y} \in \mathcal{P}(A)$, and to strategies by summing over those with coefficients cancelling out symmetries on x and y. The unit ϵ_A and co-unit and η_A are both strategies with only the empty isogmentation in their support, with coefficient 1. We have:

▶ **Proposition 31.** *For any* $-$*-arena* A*, the tuple* $(A, \delta_A, \epsilon_A, \mu_A, \eta_A)$ *is a bialgebra.*

Additionally, it is direct that this is compatible with the monoidal structure.

Pointed Identity. The pointed identity $\mathrm{id}_A^\bullet$ is defined by (2), restricted to *pointed* positions – the laws of Figure 19 follow. The categorical notion of *pointedness* from Section 4.1 agrees with the concrete one in Section 3.1: σ is pointed iff all the isogmentations in its support are.

Closed structure. We use the currying bijection $\Lambda_{\Gamma,A,B}$ from Section 3. For $\sigma : \Gamma \otimes A \vdash B$, we set $\Lambda_{\Gamma,A,B}(\sigma) = \sum_{\mathbf{q} \in \mathsf{Isog}(\Gamma \otimes A \vdash B)} \sigma(\mathbf{q}) \cdot \Lambda_{\Gamma,A,B}(\mathbf{q})$, which directly yields

$$\Lambda_{\Gamma,A,B} : \mathsf{Strat}(\Gamma \otimes A, B) \cong \mathsf{Strat}(\Gamma, A \Rightarrow B)$$

from which evaluation is $\mathrm{ev}_{A,B} = \Lambda^{-1}_{A \Rightarrow B,A,B}(\mathrm{id}_{A \Rightarrow B})$. Altogether:

▶ **Theorem 32.** Strat *is a closed resource category.*

Compatibility with normal forms. Finally, we show compatibility with normal forms – the crux is that the i-lifting in Figure 13 matches the first Base clause in Figure 23, when x is the i-th variable of Γ:

▶ **Proposition 33.** *Consider* $s \in \mathsf{Val}(\Gamma; F)$ *or* $s \in \mathsf{Base}(\Gamma)$ *a normal form.*
Then, $[\![s]\!]$ *is the sum having* $\|s\|$ *with coefficient* 1*, and* 0 *everywhere else.*

▶ **Corollary 34.** *If* $s \in \mathsf{Val}(\Gamma; F)$ *with normal form* $s \Rightarrow^* \sum_{i \in I} s_i$*, then* $[\![s]\!] = \sum_{i \in I} \|s_i\|$*.*

6 Concluding remarks

The correspondence with game semantics relies on the terms of the resource calculus to be η-expanded. This was expected – as in [23] – but some consequences deserve discussion.

Firstly, $x : F \to G$ is not a valid term as it is not η-long: it hides some infinitary copycat behaviour that must be written explicitly in our typed resource calculus, requiring an infinite sum as in (2). This makes our calculus finitary in a stronger sense than usual: each normal resource term describes a simple, finite behaviour, and one can prove that it corresponds to a single point of the relational model of [9]. This also means that in the absence of infinite sums, our typed syntax is *not* a resource category as it lacks identities.

Secondly, one might think that having an η-long syntax puts the pure λ-calculus out of reach. It is in fact possible to enforce η-expandedness on terms without typing, but this requires altering the syntax of the calculus allowing for infinite sequences of abstractions, as well as applications to infinite sequences of (almost always empty) bags. This corresponds to finding the analogue of a reflexive object in the category of games. In [23], Tsukada and Ong suggest the resource calculus with tests [8] as a candidate, but this does not seem fit for the task: it does not allow to represent arbitrary infinite sequences of abstractions; and it gives a syntactic counterpart to points of the relational model that do not correspond to any normal resource term nor any pointed augmentation. It is however possible to design a suitable language, enjoying the same relationship with Nakajima trees [21] (see also [3, Exercise 19.4.4]) as that of the ordinary resource calculus with Böhm trees. We leave the exposition of this for future work.

References

1 Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *CoRR*, abs/1311.6125, 2013.

2 Davide Barbarossa and Giulio Manzonetto. Taylor subsumes Scott, Berry, Kahn and Plotkin. *Proc. ACM Program. Lang.*, 4(POPL):1:1–1:23, 2020.

3 Hendrik Pieter Barendregt. *The lambda calculus - its syntax and semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1985.

4 Lison Blondeau-Patissier and Pierre Clairambault. Positional injectivity for innocent strategies. In Naoki Kobayashi, editor, *6th International Conference on Formal Structures for Computation and Deduction, FSCD 2021, July 17-24, 2021, Buenos Aires, Argentina (Virtual Conference)*, volume 195 of *LIPIcs*, pages 17:1–17:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSCD.2021.17.

5 Richard Blute, J. Robin B. Cockett, Jean-Simon Pacaud Lemay, and Robert A. G. Seely. Differential categories revisited. *Appl. Categorical Struct.*, 28(2):171–235, 2020. doi:10.1007/s10485-019-09572-y.

6 Richard F Blute, J Robin B Cockett, and Robert AG Seely. Cartesian differential categories. *Theory and Applications of Categories*, 22(23):622–672, 2009.

7 Pierre Boudes. Thick subtrees, games and experiments. In *TLCA*, volume 5608 of *Lecture Notes in Computer Science*, pages 65–79. Springer, 2009.

8 Antonio Bucciarelli, Alberto Carraro, Thomas Ehrhard, and Giulio Manzonetto. Full abstraction for the resource lambda calculus with tests, through Taylor expansion. *Log. Methods Comput. Sci.*, 8(4), 2012. doi:10.2168/LMCS-8(4:3)2012.

9 Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. Categorical models for simply typed resource calculi. In *MFPS*, volume 265 of *Electronic Notes in Theoretical Computer Science*, pages 213–230. Elsevier, 2010.

10 Simon Castellan and Pierre Clairambault. Disentangling parallelism and interference in game semantics. *CoRR*, abs/2103.15453, 2021.

11 Simon Castellan, Pierre Clairambault, Silvain Rideau, and Glynn Winskel. Games and strategies as event structures. *Log. Methods Comput. Sci.*, 13(3), 2017. doi:10.23638/LMCS-13(3:35)2017.

12 Simon Castellan, Pierre Clairambault, and Glynn Winskel. Thin games with symmetry and concurrent Hyland-Ong games. *Log. Methods Comput. Sci.*, 15(1), 2019. doi:10.23638/LMCS-15(1:18)2019.

13 Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theor. Comput. Sci.*, 309(1-3):1–41, 2003.

14 Thomas Ehrhard and Laurent Regnier. Differential interaction nets. *Theor. Comput. Sci.*, 364(2):166–195, 2006.

15 Thomas Ehrhard and Laurent Regnier. Uniformity and the Taylor expansion of ordinary lambda-terms. *Theor. Comput. Sci.*, 403(2-3):347–372, 2008. doi:10.1016/j.tcs.2008.06.001.

16 J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000. doi:10.1006/inco.2000.2917.

17 Jim Laird. A game semantics of the asynchronous π-calculus. In *CONCUR*, volume 3653 of *Lecture Notes in Computer Science*, pages 51–65. Springer, 2005.

18 Giulio Manzonetto. What is a categorical model of the differential and the resource λ-calculi? *Math. Struct. Comput. Sci.*, 22(3):451–520, 2012.

19 Paul-André Melliès. Asynchronous games 2: The true concurrency of innocence. *Theor. Comput. Sci.*, 358(2-3):200–228, 2006. doi:10.1016/j.tcs.2006.01.016.

20 Paul-André Mellies. Categorical semantics of linear logic. *Panoramas et syntheses*, 27:15–215, 2009.

21 Reiji Nakajima. Infinite normal forms for the λ-calculus. In Corrado Böhm, editor, *Lambda-Calculus and Computer Science Theory, Proceedings of the Symposium Held in Rome, Italy, March 25-27, 1975*, volume 37 of *Lecture Notes in Computer Science*, pages 62–82. Springer, 1975. doi:10.1007/BFb0029519.

22 Ken Sakayori and Takeshi Tsukada. A truly concurrent game model of the asynchronous π-calculus. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 389–406, 2017. doi:10.1007/978-3-662-54458-7_23.

23 Takeshi Tsukada and C.-H. Luke Ong. Plays as resource terms via non-idempotent intersection types. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 237–246. ACM, 2016. doi:10.1145/2933575.2934553.

Rewriting Modulo Traced Comonoid Structure

Dan R. Ghica
University of Birmingham, UK

George Kaye
University of Birmingham, UK

Abstract

In this paper we adapt previous work on rewriting string diagrams using hypergraphs to the case where the underlying category has a *traced comonoid structure*, in which wires can be forked and the outputs of a morphism can be connected to its input. Such a structure is particularly interesting because any traced Cartesian (dataflow) category has an underlying traced comonoid structure. We show that certain subclasses of hypergraphs are fully complete for traced comonoid categories: that is to say, every term in such a category has a unique corresponding hypergraph up to isomorphism, and from every hypergraph with the desired properties, a unique term in the category can be retrieved up to the axioms of traced comonoid categories. We also show how the framework of double pushout rewriting (DPO) can be adapted for traced comonoid categories by characterising the valid pushout complements for rewriting in our setting. We conclude by presenting a case study in the form of recent work on an equational theory for *sequential circuits*: circuits built from primitive logic gates with delay and feedback. The graph rewriting framework allows for the definition of an *operational semantics* for sequential circuits.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting

Keywords and phrases symmetric traced monoidal categories, string diagrams, graph rewriting, comonoid structure, double pushout rewriting

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.14

Related Version *Full Version*: `https://arxiv.org/abs/2302.09631`

Funding This work was supported by the Engineering and Physical Sciences Research Council [grant EP/V001612/1].

Acknowledgements Thanks to Chris Barrett for helpful comments.

1 Introduction

String diagrams constitute a useful and elegant conceptual bridge between term rewriting and graph rewriting. We will not reprise here, for lack of space, the by now impressive body of theoretical and applied work for and with string diagrams but will take it as a given. The survey [30] is a suitable starting point into the literature.

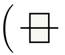
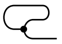
The purpose of this paper is to support reasoning (via graph rewrite) in *traced categories with a comonoid structure* but without a monoid structure. Prior art on this topic exists [21, 10], but it is based on the framework of "framed point graphs" which requires rewriting modulo so called *wire homeomorphisms*. This style of rewriting is awkward and is increasingly considered as obsolete as compared to more recent work on rewriting with hypergraphs [4, 5, 6], possibly modulo *Frobenius* structure. The variation to just a (co)monoid structure ("half a Frobenius") has been studied as well [13, 27].

The study of rewriting for traced categories with a comonoid structure is motivated by an important application, *dataflow categories* [8, 9, 15], which represent a categorical foundation for the semantics of digital circuits [14]. It is also technically challenging, as it falls in a gap between compact closed structures constructible via Frobenius and symmetric

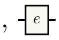
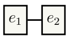
8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 14; pp. 14:1–14:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

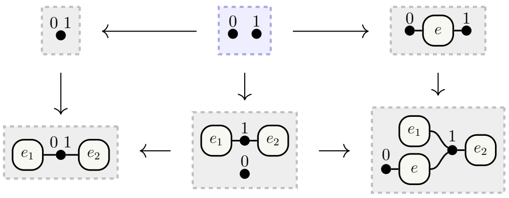
monoidal categories (without trace) so “off the shelf” solutions cannot be currently used. In fact the gap between the kind of semantic models which use an underlying compact closed structure and those which use a traced monoidal structure is significant: the former have a *relational* nature with subtle causality (e.g. quantum or electrical circuits) whereas the latter are *functional* with clear input-output causality (e.g. digital or logical circuits) so it is not surprising that the underlying rewrite frameworks should differ.

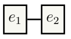
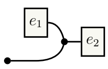
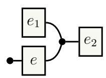
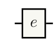
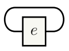
A key feature of compact closed categories is that the Cartesian product, if it exists, is degenerate and identified with the co-product. Even without invoking copying, we will see how trying to perform rewriting in a traced category with a comonoid structure can also lead to inconsistencies. This is a firm indication that a bespoke rewriting framework needs to be constructed to fill this particular situation.

Contributions. This paper makes two distinct technical contributions. The first is to show that one subclass of cospans of hypergraphs (“partial monogamous”) are fully complete for traced terms (Corollary 35), and another class (“partial left-monogamous”) are fully complete for traced comonoid terms (Theorem 46). The challenge is not so much in proving the correctness of the construction but in defining precisely what these combinatorial structures should be. In particular, the extremal point of tracing the identity: Tr $\Big($  $\Big)$ = , corresponding graphically to a closed loop, provides a litmus test. The way this is resolved must be robust enough to handle the addition of the comonoid structure, which graphically corresponds to “tracing a forking wire”: Tr $\Big($ $\Big)$ =  .

The key step in performing double pushout (DPO) rewriting is identifying a *pushout complement*: the context of a rewrite step. For a given rule and graph, there may be multiple such pushout complements, but not all of these may represent a valid rewrite in a given string diagram setting. When rewriting with Frobenius, every pushout complement is valid [4] whereas when rewriting with symmetric monoidal structure exactly one pushout complement is valid [5]. For the traced case some pushout complements are valid and some are not. The second contribution is to characterise the valid pushout complements as “traced boundary complements” (Definition 58).

This is best illustrated with an example in which there is a pushout complement that is valid in a Frobenius setting because it uses the monoid structure, but it is not valid neither in a traced, nor even in a traced comonoid setting. Imagine we have a rule ⟨ — ,  ⟩ and a term , and rewrite it as follows.



This corresponds to the term rewrite  =  = , which holds in a Frobenius setting but not a setting without a commutative monoid structure. On the other hand, the rewriting system for symmetric monoidal categories [5] is too restrictive as it enforces that any matching must be mono: this prevents matchings such as  in . Here again the challenge is precisely identifying the concept of traced boundary complement mathematically. The solution, although not immediately obvious, is not complicated, again requiring a generalisation from monogamy to partial monogamy (Theorems 65 and 69).

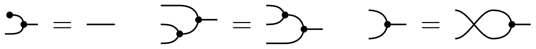


Figure 1 Equations $\mathcal{E}_{\mathbf{CMon}}$ of a *commutative monoid.*

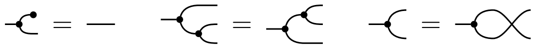


Figure 2 Equations $\mathcal{E}_{\mathbf{CComon}}$ of a *commutative comonoid.*

2 Monoidal theories and hypergraphs

When modelling a system using monoidal categories, its components and properties are specified using a *monoidal theory.* A class of SMCs particularly interesting to us is that of *PROPs* [26] ("categories of *PRO*ducts and *P*ermutations"), which have natural numbers as objects and addition as tensor product.

▶ **Definition 1** (Symmetric monoidal theory). *A* (single-sorted) symmetric monoidal theory *(SMT) is a tuple* $(\Sigma, \mathcal{E})$ *where* Σ *is a set of* generators *in which each generator* $\phi \in \Sigma$ *has an associated* arity $\mathsf{dom}(\phi) \in \mathbb{N}$ *and* coarity $\mathsf{cod}(\phi) \in \mathbb{N}$*, and* $\mathcal{E}$ *is a set of equations. Given a SMT* $(\Sigma, \mathcal{E})$*, let* $\mathbf{S}_\Sigma$ *be the strict symmetric monoidal category freely generated over* Σ *and let* $\mathbf{S}_{\Sigma,\mathcal{E}}$ *be* $\mathbf{S}_\Sigma$ *quotiented by the equations in* $\mathcal{E}$*. We write* $\mathbf{S} := \mathbf{S}_\emptyset$ *for the SMC with terms constructed solely from identities and symmetries.*

▶ Remark 2. One can also define a *multi-sorted* SMT, in which wires can be of multiple colours. For brevity, we will only consider the single-sorted case, but the results generalise easily using the results of [4, 5].

While one could reason in $\mathbf{S}_\Sigma$ using the one-dimensional categorical term language, it is more intuitive to reason with *string diagrams* [19, 30], which represent *equivalence classes* of terms up to the axioms of SMCs. In the language of string diagrams, a generator $\phi\colon m \to n$ is drawn as a box $m\text{-}\boxed{\phi}\text{-}n$, the identity id_n as $n\text{-}\boxed{}\text{-}n$, and the symmetry $\sigma_{m,n}$ as ${}^{m}_{n}\boxed{\times}{}^{n}_{m}$. Composite terms will be illustrated as wider boxes $m\text{-}\boxed{\;f\;}\text{-}n$ to distinguish them from generators: then (diagrammatic order) composition $m\text{-}\boxed{\;f\;}\text{-}n \mathbin{;} n\text{-}\boxed{\;g\;}\text{-}p$ is defined as horizontal juxtaposition $m\text{-}\boxed{\;f\;}\text{-}\boxed{\;g\;}\text{-}p$ and tensor $m\text{-}\boxed{\;f\;}\text{-}n \otimes p\text{-}\boxed{\;g\;}\text{-}q$ as vertical juxtaposition $\begin{matrix} m\text{-}\boxed{\;f\;}\text{-}n \\ p\text{-}\boxed{\;g\;}\text{-}q \end{matrix}$.

The graphical notation clearly illustrates the differences between the *syntactic* category $\mathbf{S}_\Sigma$ and the *semantic* category $\mathbf{S}_{\Sigma,\mathcal{E}}$. In the former, only "structural" equalities of the axioms of SMCs hold: moving boxes around while retaining connectivity. In the latter, more equations hold so terms with completely different boxes and connectivity can be equal.

▶ **Example 3.** The monoidal theory of *special commutative Frobenius algebras* is defined as $(\Sigma_{\mathbf{Frob}}, \mathcal{E}_{\mathbf{Frob}})$ where $\Sigma_{\mathbf{Frob}} := \{$, , , $\}$ and the equations of $\mathcal{E}_{\mathbf{Frob}}$ are listed in Figures 1–3. We write $\mathbf{Frob} := \mathbf{S}_{\Sigma_{\mathbf{Frob}},\mathcal{E}_{\mathbf{Frob}}}$.

Reasoning equationally using string diagrams is certainly attractive as a pen-and-paper method, but for larger systems it quickly becomes intractible to do this by hand. Instead, it is desirable to perform equational reasoning *computationally.* Unfortunately, string diagrams as topological objects are not particularly suited for this purpose; instead, we require a combinatorial representation. Fortunately, this has been well studied recently, first with *string graphs* [10, 21] and later with *hypergraphs* [4, 5, 6], a generalisation of regular graphs in which edges can be the source or target of an arbitrary number of vertices. In this paper we are concerned with the latter.

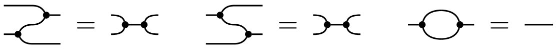


Figure 3 Equations $\mathcal{E}_{\mathbf{Frob}}$ of a *special commutative Frobenius algebra*, in addition to those in Figures 1 and 2.

Hypergraphs are formally defined as objects in a functor category.

▶ **Definition 4** (Hypergraph). *Let* $\mathbf{X}$ *be the category containing objects* (k, l) *for* $k, l \in \mathbb{N}$ *and one additional object* $\star$*. For each* (k, l) *there are* $k + l$ *morphisms* $(k, l) \to \star$*. Let* $\mathbf{Hyp}$ *be the functor category* $[\mathbf{X}, \mathbf{Set}]$*.*

An object in $\mathbf{Hyp}$ maps $\star$ to a set of vertices, and each (k, l) to a set of hyperedges with k sources and l targets. Given a hypergraph $F \in \mathbf{Hyp}$, we write $F_\star$ for its set of vertices and $F_{k,l}$ for the set of edges with k sources and l targets. A morphism of hypergraphs $f \colon F \to G \in \mathbf{Hyp}$ consists of functions $f_\star$ and $f_{k,l}$ for each $k, l \in \mathbb{N}$ preserving sources and targets in the obvious way. Hypergraph morphisms can be used to *label* hypergraphs according to a signature.

▶ **Definition 5** (Slice category [25]). *For a category* $\mathbf{C}$ *and an object* $C \in \mathbf{C}$*, the* slice category $\mathbf{C}/C$ *is the category with objects the morphisms of* $\mathbf{C}$ *with target* C*, and where a morphism* $(f \colon X \to C) \to (f' \colon X' \to C)$ *is a morphism* $g \colon X \to X' \in \mathbf{C}$ *such that* $f' \circ g = f$*.*

▶ **Definition 6** (Hypergraph signature [4]). *For a given monoidal signature* Σ*, its corresponding* hypergraph signature $[\![\Sigma]\!]$ *is the hypergraph with a single vertex* v *and edges* $e_\phi \in [\![\Sigma]\!]_{\mathsf{dom}(\phi),\mathsf{cod}(\phi)}$ *for each* $\phi \in \Sigma$*. For a hyperedge* e_ϕ*,* $i < \mathsf{dom}(\phi)$ *and* $j < \mathsf{cod}(\phi)$*,* $\mathsf{s}_i(e_\phi) = \mathsf{t}_j(e_\phi) = v$*.*

▶ **Definition 7** (Labelled hypergraph [4]). *For a monoidal signature* Σ*, let the category* $\mathbf{Hyp}_\Sigma$ *be defined as the slice category* $\mathbf{Hyp}/[\![\Sigma]\!]$*.*

While (labelled) hypergraphs may have dangling vertices, they do not have *interfaces* specifying the order of inputs and outputs. These can be provided using *cospans.*

▶ **Definition 8** (Categories of cospans [5]). *For a finitely cocomplete category* $\mathbf{C}$*, a* cospan *from* $X \to Y$ *is a pair of arrows* $X \to A \leftarrow Y$*. A* cospan morphism $(X \xrightarrow{f} A \xleftarrow{g} Y) \to (X \xrightarrow{h} B \xleftarrow{k} Y)$ *is a morphism* $\alpha \colon A \to B \in \mathbf{C}$ *such that* $\alpha \circ f = h$ *and* $\alpha \circ g = k$*.*

Two cospans $X \to A \leftarrow Y$ *and* $X \to B \leftarrow Y$ *are* isomorphic *if there exists a morphism of cospans as above where* α *is an isomorphism. Composition is by pushout. The identity is* $X \xrightarrow{\mathsf{id}_X} X \xleftarrow{\mathsf{id}_X} X$*. The category of cospans over* $\mathbf{C}$*, denoted* $\mathsf{Csp}(\mathbf{C})$ *has as objects the objects of* $\mathbf{C}$ *and as morphisms the isomorphism classes of cospans. This category has monoidal product given by the coproduct in* $\mathbf{C}$ *with unit the initial object* $0 \in \mathbf{C}$*.*

The interfaces of a hypergraph can be specified as cospans by having the "legs" of the cospan pick vertices in the graph at the apex.

▶ **Definition 9** (Discrete hypergraph). *A hypergraph is called* discrete *if it has no edges.*

A discrete hypergraph F with $|F_\star| = n$ is written as n when clear from context. Morphisms from discrete hypergraphs to a main graph pick out the vertices in the interface: to assign an order to these vertices some more categorical machinery is required.

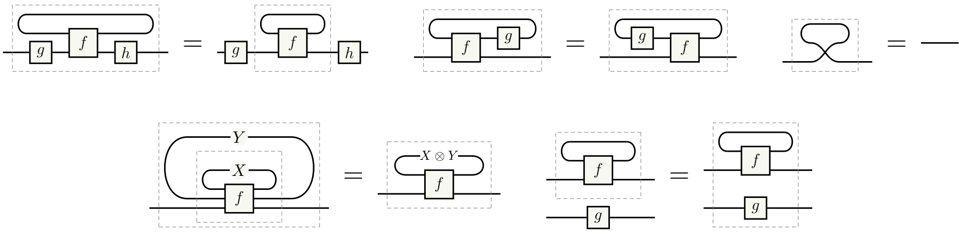


Figure 4 Equations that hold in any *symmetric traced monoidal category.*

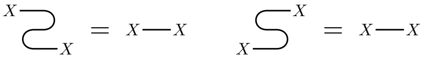


Figure 5 Equations that hold in any *compact closed category.*

▶ **Theorem 10** ([4], Thm. 3.6)**.** *Let $\mathbb{X}$ be a PROP whose monoidal product is a coproduct, $\mathbf{C}$ a category with finite colimits, and $F\colon \mathbb{X} \to \mathbf{C}$ a coproduct-preserving functor. Then there exists a PROP $\mathsf{Csp}_F(\mathbf{C})$ whose arrows $m \to n$ are isomorphism classes of $\mathbf{C}$ cospans $Fm \to C \leftarrow Fn$.*

▶ **Definition 11.** *Let $\mathbb{F}$ be the PROP with morphisms $m \to n$ the functions between finite sets $[m] \to [n]$.*

▶ **Definition 12** ([4])**.** *Let $D\colon \mathbb{F} \to \mathbf{Hyp}_\Sigma$ be the faithful, coproduct-preserving functor that sends each object $m \in \mathbb{F}$ to the discrete hypergraph $m \in \mathbf{Hyp}_\Sigma$ and each morphism to the induced homomorphism of discrete hypergraphs.*

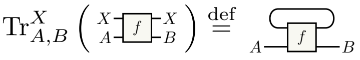
From this we define the category $\mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$ with objects *discrete cospans of hypergraphs*. Since the legs of each cospan are discrete hypergraphs containing some number of vertices, the objects of this category can be viewed as natural numbers, making this another PROP.

3 Hypergraphs for traced categories

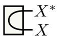
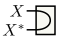
We wish to use the hypergraph framework for a setting with a *trace*.

▶ **Definition 13** (Symmetric traced monoidal category [20, 16])**.** *A* symmetric traced monoidal category *(STMC) is a symmetric monoidal category $\mathbf{C}$ equipped with a family of functions $\mathrm{Tr}^X_{A,B}(-)\colon \mathbf{C}(X \otimes A, X \otimes B) \to \mathbf{C}(A, B)$ for any objects A, B and X satisfying the axioms of STMCs listed in Figure 4.*

In string diagrams, the trace is represented by joining output wires to input wires:



Traced monoidal categories are not the only kind of category in which wires can "bend".

▶ **Definition 14** (Compact closed category)**.** *A* compact closed category *(CCC) is a symmetric monoidal category in which every object X has a* dual *X^* equipped with morphisms called the* unit  *("cup") and the* counit  *("cap") satisfying the equations of CCCs listed in Figure 5.*

Dual objects are conventionally drawn as wires flowing the "other way", but in this paper this is not necessary as all categories will be *self-dual*: any object X is isomorphic to X^*.

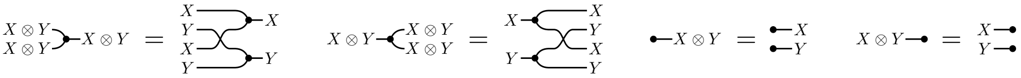
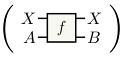
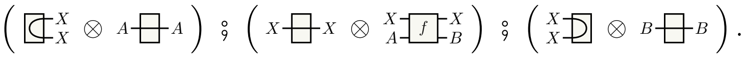


Figure 6 Equations $\mathcal{E}_{\mathbf{Hyp}}$ of a *hypergraph category*, in addition to those in Figures 1–3.

▶ **Proposition 15** (Canonical trace ([20], Prop. 3.1)). *Any CCC has a trace* $\mathrm{Tr}^X_{A,B}$  *called the* canonical trace, *defined for the self-dual case as*



The category of interfaced hypergraphs as defined in the previous section already contains the structure necessary to define a trace.

▶ **Definition 16** (Hypergraph category [11]). *A* hypergraph category *is a symmetric monoidal category in which each object X has a special commutative Frobenius structure in the sense of Example 3 satisfying the equations in Figure 6.*

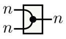
▶ **Proposition 17** ([28]). *Any hypergraph category is self-dual compact closed.*

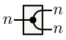
Proof. The cup is constructed as ⨾ and the cap as ⨾ . ◀

A generic "hypergraph category" should not be confused with the category of hypergraphs **Hyp**, which is not itself a hypergraph category. However, the category of *cospans* of hypergraphs is such a category.

▶ **Proposition 18** ([7, 4]). $\mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$ *is a hypergraph category.*

Proof. A Frobenius structure can be defined on $\mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$ for each $n \in \mathbb{N}$ as follows:

 $:= n + n \to n \leftarrow n$ $:= 0 \to n \leftarrow n$

 $:= n \to n \leftarrow n + n$ $:= n \to n \leftarrow 0$ ◀

▶ **Corollary 19.** $\mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$ *is compact closed.*

▶ **Corollary 20.** $\mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$ *has a trace.*

This means that a STMC freely generated over a signature faithfully embeds into a CCC generated over the same signature, mapping the trace in the former to the canonical trace in the latter. However, this mapping is not *full*: there are terms in a CCC that are not terms in a STMC, such as . So we must still restrict the cospans of hypergraphs in $\mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$ we use for *traced* terms.

3.1 Monogamy

In [3], it is shown that terms in a (non-traced) symmetric monoidal category are interpreted via a faithful functor into a sub-PROP of $\mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$. One condition on this sub-PROP is that all hypergraphs are *acyclic*. Clearly, to model trace this condition must be dropped.

However, there is also another condition known as *monogamy*: informally, this means that every vertex has exactly one "in" and "out" connection, be it to an edge or an interface. For the most part, this condition also applies to the traced case: wires cannot arbitrarily fork and join. There is one nuance: the trace of the identity. This is depicted as a closed

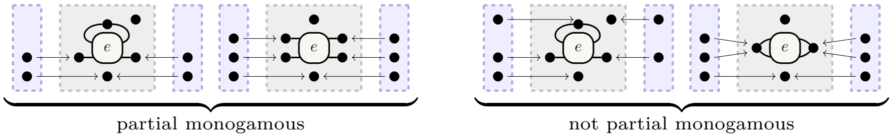


Figure 7 Examples of cospans that are and are not partial monogamous.

loop $\mathrm{Tr}^1\Big($ $\Big) =$, and one might think that it can be discarded, i.e. $=$. This is *not* always the case, such as in the category of finite dimensional vector spaces [15, Sec. 6.1]. These closed loops must be represented in the hypergraph framework: there is a natural representation as a lone vertex disconnected from either interface. In fact, this is exactly how the canonical trace applied to an identity is interpreted in $\mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$.

▶ **Definition 21.** *For a hypergraph $F \in \mathbf{Hyp}$, the* degree *of a vertex $v \in F_\star$ is a tuple (i, o) where i is the number of pairs (e, i) where e is a hyperedge with v as its ith target, and o is similarly the number of pairs (e, j) where e is a hyperedge with v as its jth target.*

▶ **Definition 22.** *For a cospan $m \xrightarrow{f} F \xleftarrow{g} n \in \mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$, we say it is* partial monogamous *if f and g are mono and, for all nodes $v \in F_\star$, the degree of v is*

$$\begin{array}{llrl} (0,0) & \textit{if } v \in f_\star \wedge v \in g_\star & (0,1) & \textit{if } v \in f_\star \\ (1,0) & \textit{if } v \in g_\star & (0,0) \textit{ or } (1,1) & \textit{otherwise} \end{array}$$

Intuitively, partial monogamy means that each vertex has either exactly one "in" and one "out" connection to an edge or to an interface, or none at all.

▶ **Example 23.** Examples of cospans that are and are not partial monogamous are shown in Figure 7.

In order to establish a correspondence between cospans of partial monogamous hypergraphs, they need to be assembled into a sub-PROP of $\mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$.

▶ **Theorem 24.** *Let $m \to F \leftarrow n$, $n \to G \leftarrow p$, $p \to H \leftarrow q$ and $x + m \to K \leftarrow x + n$ be partial monogamous cospans in $\mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$. Then,*

- *identities and symmetries are partial monogamous;*
- *$m \to F \leftarrow n \,⨾\, n \to G \leftarrow p$ is partial monogamous;*
- *$m \to F \leftarrow n \otimes p \to H \leftarrow q$ is partial monogamous; and*
- *$\mathrm{Tr}^x\,(x + m \to K \leftarrow x + n)$ is partial monogamous.*

Proof. Since any monogamous hypergraph is also partial monogamous, the first three points hold due to [4, Prop.16], dropping the acyclicity condition. The final condition is routine by case analysis on the interfaces a vertex occurs in. ◀

▶ **Definition 25.** *Let $\mathsf{PMCsp}_D(\mathbf{Hyp}_\Sigma)$ be the sub-PROP of $\mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$ containing only the partial monogamous cospans of hypergraphs.*

Crucially, while we leave $\mathsf{PMCsp}_D(\mathbf{Hyp}_\Sigma)$ in order to construct the trace using the cup and cap, the resulting cospan *is* in $\mathsf{PMCsp}_D(\mathbf{Hyp}_\Sigma)$.

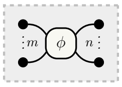
3.2 From terms to graphs

▶ **Definition 26.** *For a SMT $(\Sigma, \mathcal{E})$, let $\mathbf{T}_\Sigma$ be the strict STMC freely generated over the generators in Σ. Let $\mathbf{T}_{\Sigma,\mathcal{E}}$ be $\mathbf{T}_\Sigma$ quotiented by equations in $\mathcal{E}$.*

A *(traced) PROP morphism* is a strict (traced) symmetric monoidal functor between PROPs. For $\mathsf{PMCsp}_{FI}(\mathbf{Hyp}_\Sigma)$ to be suitable for reasoning with a traced category $\mathbf{T}_\Sigma$ for some given signature, there must be a *fully complete* PROP morphism $\mathbf{T}_\Sigma \to \mathsf{PMCsp}_{FI}(\mathbf{Hyp}_\Sigma)$: a full and faithful functor from terms to cospans of hypergraphs.

We exploit the interplay between compact closed and traced categories in order to reuse the existing PROP morphisms from [4] for the traced case. Since $\mathbf{S}_\Sigma$ is freely generated, these PROP morphisms can be defined solely on generators.

▶ **Definition 27** ([4])**.** *Let $\llbracket - \rrbracket_\Sigma \colon \mathbf{S}_\Sigma \to \mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$ be a PROP morphism defined as*

$\llbracket\, {}^{m}\!\!-\!\boxed{\phi}\!-\!{}^{n} \,\rrbracket_\Sigma := m \to$  $\leftarrow n$

$$\llbracket\, {}^{n}\!\!-\!\boxed{}\!-\!{}^{n} \,\rrbracket_\Sigma := n \xrightarrow{\mathsf{id}} n \xleftarrow{\mathsf{id}} n \qquad \llbracket\, {}^{m}_{n}\!\boxed{\times}\!{}^{n}_{m} \,\rrbracket_\Sigma := m + n \xrightarrow{[\mathsf{id},\mathsf{id}]} m + n \xleftarrow{[\mathsf{id},\mathsf{id}]} n + m$$

Let $[-]_\Sigma \colon \mathbf{Frob} \to \mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$ be a PROP morphism defined as in Proposition 18. Then, let $\langle\!\langle - \rangle\!\rangle_\Sigma \colon \mathbf{S}_\Sigma + \mathbf{Frob} \to \mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$ be the copairing of $\llbracket - \rrbracket_\Sigma$ and $[-]_\Sigma$.

▶ **Lemma 28.** *Let ${}^{m}\!\!-\!\boxed{f}\!-\!{}^{n}$ be a term in $\mathbf{T}_\Sigma$. Then there exists at least one ${}^{x}_{m}\!\boxed{g}\!{}^{x}_{n} \in \mathbf{S}_\Sigma$ such that $\mathrm{Tr}^x\left(\boxed{g} \right) = -\boxed{f}-$.*

▶ **Proposition 29.** *There exists a faithful PROP morphism $\lfloor - \rfloor^{\mathbf{T}}_{\Sigma} \colon \mathbf{T}_\Sigma \to \mathbf{S}_\Sigma + \mathbf{Frob}$.*

Proof. Lemma 28 is used to isolate a term in $\mathbf{S}_\Sigma$. The corresponding term in $\mathbf{S}_\Sigma + \mathbf{Frob}$ is then the canonical trace of this term. There may be many such terms in $\mathbf{S}_\Sigma$, but the canonical trace being a trace means that any possible outcomes post-trace are all equal. The equations of **Frob** do not merge any morphisms since the only use of the generators of **Frob** is in the canonical trace, to which the Frobenius equations do not apply. ◀

A summary of these PROP morphisms is shown in Figure 8.

Before progressing to the main theorem, we must show a result about terms in $\mathbf{S}$: terms constructed from just symmetries and identities. There is a correspondence between such terms and *bijective* functions.

▶ **Definition 30.** *Let $\mathbb{P}$ be the sub-PROP of $\mathbb{F}$ containing only the bijective functions.*

▶ **Lemma 31.** $\mathbf{S} \cong \mathbb{P}$.

▶ **Lemma 32.** *Given a monogamous cospan $m \xrightarrow{f} m \xleftarrow{g} m$, there exists a unique term ${}^{m}\!\!-\!\boxed{h}\!-\!{}^{m} \in \mathbf{S}$ up to the axioms of SMCs such that $\llbracket -\boxed{h}- \rrbracket_\Sigma = m \xrightarrow{f} m \xleftarrow{g} m$.*

Proof. Since the cospan is monogamous, f and g are mono. As the cospan is also discrete, there exists a (unique) bijective function $h' \colon [m] \to [m]$ such that $h'(i) = j$ if $f(i) = g(j)$. By Lemma 31, there is a corresponding term ${}^{m}\!\!-\!\boxed{h}\!-\!{}^{m} \in \mathbf{S}$ that is unique up to SMC axioms: a simple induction shows that $\llbracket -\boxed{h}- \rrbracket_\Sigma = m \xrightarrow{f} m \xleftarrow{g} m$. ◀

Cospans of the form above are used in order to reconstruct a term in $\mathbf{T}_\Sigma$ given a cospan of partial monogamous hypergraphs, showing that partial monogamy characterises the image of $\langle\!\langle - \rangle\!\rangle_\Sigma \circ \lfloor - \rfloor^{\mathbf{T}}_{\Sigma}$.

$$\begin{tikzcd}
\mathbf{CComon} \arrow[r, "\lfloor - \rfloor^{\mathbf{C}}"] \arrow[d] & \mathbf{Frob} \arrow[d] \arrow[dr, "{[-]_\Sigma}"] & \\
\mathbf{T}_\Sigma + \mathbf{CComon} \arrow[r, "\lfloor - \rfloor_\Sigma"] & \mathbf{S}_\Sigma + \mathbf{Frob} \arrow[r, "\langle\!\langle - \rangle\!\rangle_\Sigma"] & \mathsf{Csp}_D(\mathbf{Hyp}_\Sigma) \\
\mathbf{T}_\Sigma \arrow[u] \arrow[ur, "\lfloor - \rfloor^{\mathbf{T}}_\Sigma"'] & \mathbf{S}_\Sigma \arrow[u] \arrow[ur, "{[\![-]\!]_\Sigma}"'] &
\end{tikzcd}$$

Figure 8 The various PROP morphisms at play.

▶ **Theorem 33.** *A cospan $m \to F \leftarrow n$ is in the image of $\langle\!\langle - \rangle\!\rangle_\Sigma \circ \lfloor - \rfloor^{\mathbf{T}}_\Sigma$ if and only if it is partial monogamous.*

Proof (Sketch). The ($\Rightarrow$) direction is by induction on the structure of the term. For the ($\Leftarrow$) direction, a cospan isomorphic to the original cospan can be constructed from which a term in $\mathbf{T}_\Sigma$ can be read off. Informally, this cospan is

$$\mathrm{Tr}^{x+n}\left(x+n+m \to V \leftarrow m+x+n \,;\, m+x+n \to L+E+n \leftarrow x+n+n\right) \qquad (1)$$

where V is all the vertices in F, L is the vertices with degree $(0,0)$ not in the image of the interfaces, and E is the all the hyperedges in F, "stacked" in some arbitrary order. The first component corresponds to a term in $\mathbf{S}$ by Lemma 32, and the stack of edges to a tensor of generators in $\mathbf{T}_\Sigma$. ◀

This shows that $\langle\!\langle - \rangle\!\rangle_\Sigma \circ \lfloor - \rfloor^{\mathbf{T}}_\Sigma$ is a *full* mapping from $\mathbf{T}_\Sigma$ to $\mathsf{PMCsp}_D(\mathbf{Hyp}_\Sigma)$. It remains to show that it is faithful: every term in $\mathbf{T}_\Sigma$ has a *unique* cospan of hypergraphs up to isomorphism. By definition, $\lfloor - \rfloor^{\mathbf{T}}_\Sigma$ is faithful, so we only need to consider $\langle\!\langle - \rangle\!\rangle_\Sigma$.

▶ **Proposition 34** ([4]). *$[\![-]\!]_\Sigma$ and $[-]_\Sigma$ are faithful.*

▶ **Corollary 35.** $\mathbf{T}_\Sigma \cong \mathsf{PMCsp}_{FI}(\mathbf{Hyp}_\Sigma)$.

4 Hypergraphs for traced commutative comonoid categories

We are interested in another element of structure in addition to the trace: the ability to *copy* and *discard* wires. This is known as a *(commutative) comonoid structure*: categories equipped with such a structure are also known as *gs-monoidal* (*garbage-sharing*) categories [13].

▶ **Definition 36.** *Let $(\Sigma_{\mathbf{CComon}}, \mathcal{E}_{\mathbf{CComon}})$ be the symmetric monoidal theory of* commutative comonoids, *with $\Sigma_{\mathbf{CComon}} := \{$, $\}$ and $\mathcal{E}_{\mathbf{CComon}}$ defined as in Figure 2. We write $\mathbf{CComon} := \mathbf{S}_{\Sigma_{\mathbf{CComon}}, \mathcal{E}_{\mathbf{CComon}}}$.*

From now on, we write "comonoid" to mean "commutative comonoid". There has already been work using hypergraphs for PROPs with a (co)monoid structure [13, 27] but these consider *acyclic* hypergraphs: we must ensure that removing the acyclicity condition does not lead to any degeneracies.

▶ **Definition 37** (Partial left-monogamy). *For a cospan $m \xrightarrow{f} H \xleftarrow{g} n$, we say it is* partial left-monogamous *if f is mono and, for all nodes $v \in H_\star$, the degree of v is $(0,m)$ if $v \in f_\star$ and $(0,m)$ or $(1,m)$ otherwise, for some $m \in \mathbb{N}$.*

Partial left-monogamy is a weakening of partial monogamy that allows vertices to have multiple "out" connections, which represent the use of the comonoid structure to fork wires.

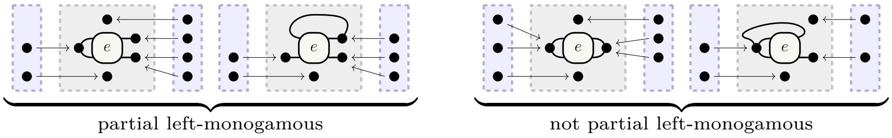


Figure 9 Examples of cospans that are and are not partial left-monogamous.

▶ **Example 38.** Examples of cospans that are and are not partial left-monogamous are shown in Figure 9.

▶ Remark 39. As with the vertices not in the interfaces with degree $(0, 0)$ in the vanilla traced case, the vertices not in the interface with degree $(0, m)$ allow for terms such as $\mathrm{Tr}\Big($ $\Big)$.

▶ **Lemma 40.** *Let $m \to F \leftarrow n$, $n \to G \leftarrow p$, $p \to H \leftarrow q$ and $x + m \to K \leftarrow x + n$ be partial left-monogamous cospans. Then,*

- *identities and symmetries are partial left-monogamous;*
- *$m \to F \leftarrow n \mathbin{;} n \to G \leftarrow p$ is partial left-monogamous;*
- *$m \to F \leftarrow n \otimes p \to H \leftarrow q$ is partial left-monogamous; and*
- *$\mathrm{Tr}^x \left(x + m \to K \leftarrow x + n\right)$ is partial left-monogamous.*

▶ **Definition 41.** *Let $\mathsf{PLMCsp}_D(\mathbf{Hyp}_\Sigma)$ be the sub-PROP of $\mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$ containing only the partial left-monogamous cospans of hypergraphs.*

This category can be equipped with a comonoid structure.

▶ **Definition 42.** *Let $\lfloor - \rfloor^{\mathbf{C}} \colon \mathbf{CComon} \to \mathbf{Frob}$ be the obvious embedding of $\mathbf{CComon}$ into $\mathbf{Frob}$, and let $\lfloor - \rfloor_\Sigma \colon \mathbf{T}_\Sigma + \mathbf{Comon} \to \mathbf{S}_\Sigma + \mathbf{Frob}$ be the copairing of $\lfloor - \rfloor_\Sigma^{\mathbf{T}}$ and $\lfloor - \rfloor^{\mathbf{C}}$.*

As before, these PROP morphisms are summarised in Figure 8. To show that partial left-monogamy is the correct notion to characterise terms in a traced comonoid setting, it is necessary to ensure that the image of these PROP morphisms lands in $\mathsf{PLMCsp}_D(\mathbf{Hyp}_\Sigma)$.

▶ **Lemma 43.** *The image of $[-]_\Sigma \circ \lfloor - \rfloor^{\mathbf{C}}$ is in $\mathsf{PLMCsp}_D(\mathbf{Hyp}_\Sigma)$.*

▶ **Corollary 44.** *The image of $\langle\!\langle - \rangle\!\rangle_\Sigma \circ \lfloor - \rfloor_\Sigma$ is in $\mathsf{PLMCsp}_D(\mathbf{Hyp}_\Sigma)$.*

▶ **Lemma 45.** *Given a partial left-monogamous cospan $m \xrightarrow{f} m \xleftarrow{g} n$, there exists a unique term ${}_{m}\!-\!\boxed{h}\!-\!{}_{n} \in \mathbf{CComon}$ up to the axioms of SMCs and comonoids such that $\left[\left\lfloor -\boxed{h}- \right\rfloor^{\mathbf{C}}\right]_\Sigma = m \xrightarrow{f} m \xleftarrow{g} n$.*

▶ **Theorem 46.** $\mathbf{T}_\Sigma + \mathbf{CComon} \cong \mathsf{PLMCsp}_{FI}(\mathbf{Hyp}_\Sigma)$.

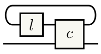
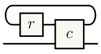
Proof. Since $\langle\!\langle - \rangle\!\rangle_\Sigma$ and $[-]_\Sigma^{\mathbf{C}}$ are faithful, it suffices to show that a cospan $m \to F \leftarrow n$ in $\mathsf{PLMCsp}_D(\mathbf{Hyp}_\Sigma)$ can be decomposed in such a way that each component is in the image of either $\langle\!\langle - \rangle\!\rangle_\Sigma \circ \lfloor - \rfloor_\Sigma^{\mathbf{T}}$ or $[-]_\Sigma \circ \lfloor - \rfloor^{\mathbf{C}}$. This is achieved by taking the construction of Theorem 33 and allowing the first component to be partial left-monogamous; by Lemma 45 a term in $\mathbf{CComon}$ can be retrieved from this. ◀

$$\begin{tikzcd}
L \arrow[d] \arrow[dr, phantom, "\urcorner", very near start] & i+j \arrow[l] \arrow[r] \arrow[d] & R \arrow[d] \arrow[dl, phantom, "\ulcorner", very near start] \\
G & C \arrow[l] \arrow[r] & H \\
 & m+n \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}
\qquad
\begin{tikzcd}
L \arrow[d, "f"'] \arrow[dr, phantom, "\urcorner", very near start] & & i+j \arrow[ll, "a:={[a_1,a_2]}"'] \arrow[d, "c:={[c_1,c_2]}"] \\
G & & C \arrow[ll, "g"] \\
 & & m+n \arrow[ull, "{[b_1,b_2]}"] \arrow[u, "d:={[d_1,d_2]}"']
\end{tikzcd}$$

Figure 10 The DPO diagram and a pushout complement.

5 Graph rewriting

We have now shown that we can reason up to the axioms of symmetric traced categories with a comonoid structure using hypergraphs: string diagrams equal by topological deformations are translated into isomorphic hypergraphs. However, to reason about an *monoidal theory* with extra equations we must actually rewrite the components of the graph. In the syntactic realm this is performed with *term rewriting*.

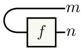
▶ **Definition 47** (Term rewriting). *A* rewriting system $\mathcal{R}$ *for a traced PROP* $\mathbf{T}_\Sigma$ *consists of a set of* rewrite rules $\langle\, i\!-\!\boxed{l}\!-\!j \;,\; i\!-\!\boxed{r}\!-\!j \,\rangle$. *Given terms* $m\!-\!\boxed{g}\!-\!n$ *and* $m\!-\!\boxed{h}\!-\!n$ *in* $\mathbf{T}_\Sigma$ *we write* $-\boxed{g}- \Rightarrow_{\mathcal{R}} -\boxed{h}-$ *if there exists rewrite rule* $(\, i\!-\!\boxed{l}\!-\!j \;,\; i\!-\!\boxed{r}\!-\!j \,)$ *in* $\mathcal{R}$ *and* ${}^{j}_{m}\!\boxed{c}\!{}^{i}_{n}$ *in* $\mathbf{T}_\Sigma$ *such that* $-\boxed{g}- =$  *and* $-\boxed{h}- =$  *by axioms of STMCs.*

The equivalent for graphs is *graph rewriting.* A common framework is that of *double pushout rewriting* (DPO rewriting); we use an extension, known as *double pushout rewriting with interfaces* (DPOI rewriting).

▶ **Definition 48** (DPO rule). *Given interfaced hypergraphs* $i \xrightarrow{a_1} L \xleftarrow{a_2} j$ *and* $i \xrightarrow{b_1} R \xleftarrow{b_2} j$*, their* DPO rule *in* $\mathbf{Hyp}_\Sigma$ *is a span* $L \xleftarrow{[a_1,a_2]} i+j \xrightarrow{[b_1,b_2]} R$*.*

▶ **Definition 49** (DPO(I) rewriting). *Let* $\mathcal{R}$ *be a set of DPO rules. Then, for morphisms* $G \leftarrow m+n$ *and* $H \leftarrow m+n$ *in* $\mathbf{Hyp}_\Sigma$*, there is a rewrite* $G \rightsquigarrow_{\mathcal{R}} H$ *if there exist a rule* $L \leftarrow i+j \rightarrow R \in \mathcal{R}$ *and cospan* $i+j \rightarrow C \leftarrow n+m \in \mathbf{Hyp}_\Sigma$ *such that diagram in the left of Figure 10 commutes.*

The first thing to note is that the graphs in the DPO diagram have a *single* interface $G \leftarrow m+n$ instead of the cospans $m \rightarrow G \leftarrow n$ we are used to. Before performing DPO rewriting in $\mathbf{Hyp}_\Sigma$, the interfaces must be "folded" into one.

▶ **Definition 50** ([5]). *Let* $\ulcorner - \urcorner \colon \mathbf{S}_\Sigma + \mathbf{Frob} \rightarrow \mathbf{S}_\Sigma + \mathbf{Frob}$ *be defined as having action* $m\!-\!\boxed{f}\!-\!n \mapsto$  *.*

Note that the result of applying $\ulcorner - \urcorner$ is not in the image of $\langle\!\langle - \rangle\!\rangle_\Sigma \circ \lfloor - \rfloor^{\mathbf{T}}_{\Sigma}$ any more. This is not an issue, so long as we "unfold" the interfaces once rewriting is completed.

▶ **Proposition 51** ([4], Prop. 4.8). *If* $\langle\!\langle\, m\!-\!\boxed{f}\!-\!n \,\rangle\!\rangle_\Sigma = m \xrightarrow{i} F \xleftarrow{o} n$ *then* $\ulcorner \langle\!\langle -\boxed{f}- \rangle\!\rangle_\Sigma \urcorner$ *is isomorphic to* $0 \rightarrow F \xleftarrow{i+o} m+n$*.*

In order to apply a given DPO rule $L \leftarrow i+j \rightarrow R$ in some larger graph $m \rightarrow G \leftarrow n$, a morphism $L \rightarrow G$ must first be identified. The next step is to "cut out" the components of L that exist in G.

▶ **Definition 52** (Pushout complement). *Let $i + j \to L \to G \to m + n$ be morphisms in $\mathbf{Hyp}_\Sigma$. Then the* pushout complement *of these morphisms is an object C with morphisms $i + j \to C \to G$ such that $L \to G \leftarrow C$ is a pushout and the diagram on the right of Figure 10 commutes.*

Given a rule $L \leftarrow i+j \to R$ and morphism $L \to G$, a pushout complement $i+j \to C \to G$ represents the context of a valid rewrite step. Once a pushout complement is computed, the pushout of $C \leftarrow i + j \to R$ can be performed to obtain the completed rewrite H. However, a pushout complement may not exist for a given rule and matching.

▶ **Definition 53** ([4], Def. 3.16). *Let $i+j \xrightarrow{a} L \xrightarrow{f} G$ be morphisms in $\mathbf{Hyp}_\Sigma$. The morphisms satisfy the* no-dangling *condition if, for every hyperedge not in the image of f, each of its source and target vertices is either not in the image of f or are in the image of $f \circ a$. The morphisms satisfy the* no-identification *condition if any two distinct elements merged by f are also in the image of $f \circ a$.*

▶ **Proposition 54** ([4], Prop. 3.17). *The morphisms $i + j \to L \to G$ have at least one pushout complement if and only if they satisfy the no-dangling and no-identification conditions.*

▶ **Definition 55.** *Given a partial monogamous cospan $i \to L \leftarrow j$, a morphism $L \to G$ is called a* matching *if it has at least one pushout complement.*

In certain settings, known as *adhesive categories* [23], it is possible to be more precise about the number of pushout complements for a given matching and rewrite rule.

▶ **Proposition 56** ([23]). *In an adhesive category, pushout complements of $i + j \xrightarrow{a} L \to G$ are unique if they exist and a is mono.*

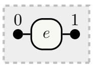
▶ **Proposition 57** ([24]). *$\mathbf{Hyp}_\Sigma$ is adhesive.*

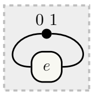
A given pushout complement uniquely determines the rewrite performed, so it might seem advantageous to always have exactly one. However, when writing modulo traced comonoid structure there are settings where having multiple pushout complements is beneficial.

5.1 Rewriting with traced structure

While in the Frobenius case considered in [4], all valid pushout complements correspond to a valid rewrite, this is not the case for the traced monoidal case. In [5], pushout complements that correspond to a valid rewrite in the non-traced symmetric monoidal case are identified as *boundary complements.* We will use a weakening of this definition.

▶ **Definition 58** (Traced boundary complement). *A pushout complement as in Definition 52 is called a* traced boundary complement *if c_1 and c_2 are mono and $j + m \xrightarrow{[c_2,d_1]} C \xleftarrow{[d_2,c_1]} n + i$ is a partial monogamous cospan.*

Unlike [5], we do not enforce that the matching is mono, as this cuts off potential rewrites in the *traced* setting, such as a matching inside a loop: 

→ 

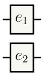
.

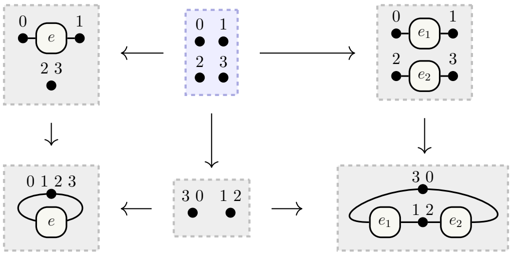
▶ **Definition 59** (Traced DPO). *For morphisms $G \leftarrow m + n$ and $H \leftarrow m + n$ in $\mathbf{Hyp}_\Sigma$, there is a traced rewrite $G \rightsquigarrow_{\mathcal{R}} H$ if there exists a rule $L \leftarrow i + j \to G \in \mathcal{R}$ and cospan $i + j \to C \leftarrow n + m \in \mathbf{Hyp}_\Sigma$ such that diagram in Definition 49 commutes and $i + j \to C$ is a traced boundary complement.*

Some intuition on the construction of traced boundary complements may be required: this will be provided through a lemma and some examples.

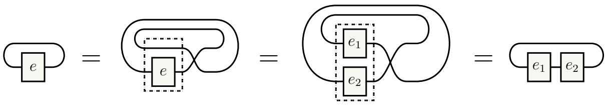
▶ **Lemma 60.** *In a traced boundary complement, let $v \in i$ and $w_0, w_1, \cdots w_k$ such that $f(a_1(v)) = f(a_2(w_0))$, $f(a_1(v)) = f(a_2(w_1))$ and so on. Then either (1) there exists exactly one w_l not in the image of d_1 such that $c_1(v) = c_2(w_l)$; (2) $c_1(v)$ is in the image of d_1; or (3) $c_1(v)$ has degree $(1,0)$. The same also holds for $w \in j$, with the interface map as d_2 and the degree as $(0,1)$.*

Often there can be valid rewrites in the realm of graphs that are non-obvious in the term language. This is because we are rewriting modulo *yanking*.

▶ **Example 61.** Consider the rule ⟨ ,  ⟩. The interpretation of this as a DPO rule in a valid traced boundary complement is illustrated below.

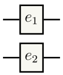


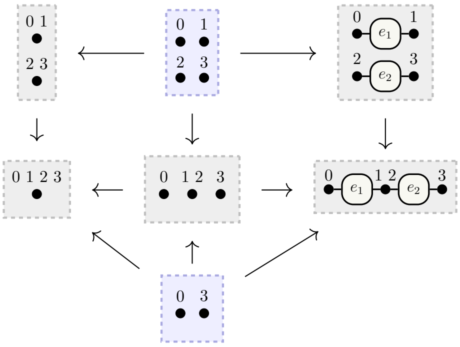
This corresponds to a valid term rewrite:

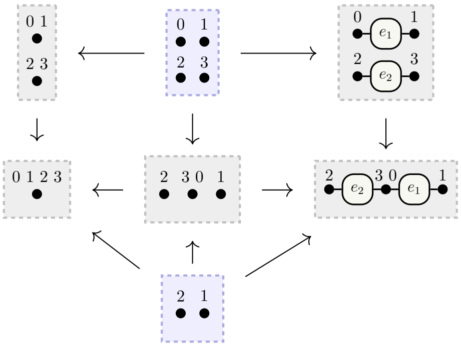


Note that applying yanking is required in the term setting because the traced wire is flowing from right to left, whereas applying the rule requires all wires flowing left to right.

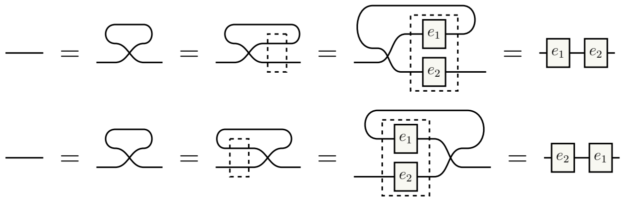
Unlike regular boundary complements, traced boundary complements need not be unique. However, this is not a problem since all pushout complements can be enumerated given a rule and matching [18].

▶ **Example 62.** Consider the rule ⟨ ,  ⟩. Below are two valid traced boundary complements involving a matching of this rule.



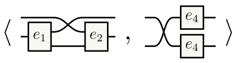


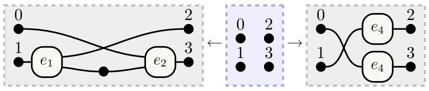
Once again, these derivations arise through yanking:



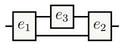
Rewriting modulo yanking also eliminates another foible of rewriting modulo (non-traced) symmetric monoidal structure. In the SMC case, the image of the matching must be *convex*: any path between vertices must also be captured. This is not necessary in the traced case.

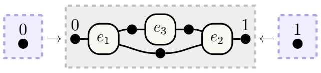
▶ **Example 63.** Consider the following rewrite rule and its interpretation.



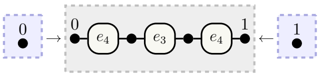


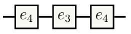
Now consider the following term and interpretation:



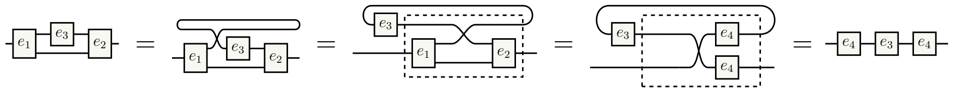


Although it is not obvious in the original string diagram, there is in fact a matching of of the former in the latter. Performing the DPO procedure yields the following:





In a non-traced setting this is an invalid rule! However, it is possible with yanking.



We are almost ready to show the soundness and completeness of this DPO rewriting system. The final prerequisite is a decomposition lemma, akin to a similar result in [4].

▶ **Lemma 64** (Traced decomposition). *Given partial monogamous cospans* $m \xrightarrow{d_1} G \xleftarrow{d_2} n$ *and* $i \xrightarrow{a_1} L \xleftarrow{a_2} j$*, along with a morphism* $L \xrightarrow{f} G$ *such that* $i + j \to L \to G$ *satisfies the no-dangling and no-identification conditions, then there exists* $j + m \xrightarrow{[c_2, d_1]} C \xleftarrow{[c_1, d_2]} i + n$ *such that* $m \to G \leftarrow n$ *can be factored as*

$$\mathrm{Tr}^i \left(\begin{array}{c} i \xrightarrow{a_1} L \xleftarrow{a_2} j \\ \otimes \\ m \to m \leftarrow m \end{array} \mathbin{;} j + m \xrightarrow{[c_2,d_1]} C \xleftarrow{[c_1,d_2]} i + n \right) \tag{2}$$

where all cospans are partial monogamous and $j + m \xrightarrow{c_2, d_1} C \xleftarrow{c_1, d_2} i + n$ *is a traced boundary complement.*

We write $\ulcorner \lfloor \mathcal{R} \rfloor_{\Sigma}^{\mathbf{T}} \urcorner$ for the pointwise map $(\,\boxed{l}\,,\,\boxed{r}\,) \mapsto (\ulcorner \lfloor \boxed{l} \rfloor_{\Sigma}^{\mathbf{T}} \urcorner, \ulcorner \lfloor \boxed{r} \rfloor_{\Sigma}^{\mathbf{T}} \urcorner)$.

▶ **Theorem 65.** *Let* $\mathcal{R}$ *be a rewriting system on* $\mathbf{T}_\Sigma$*. Then,* $m\boxed{g}n \Rightarrow_{\mathcal{R}} m\boxed{h}n$ *if and only if* $\langle\!\langle \ulcorner \lfloor \boxed{g} \rfloor_{\Sigma}^{\mathbf{T}} \urcorner \rangle\!\rangle_{\Sigma} \rightsquigarrow_{\langle\!\langle \ulcorner \lfloor \mathcal{R} \rfloor_{\Sigma}^{\mathbf{T}} \urcorner \rangle\!\rangle_{\Sigma}} \langle\!\langle \ulcorner \lfloor \boxed{h} \rfloor_{\Sigma}^{\mathbf{T}} \urcorner \rangle\!\rangle_{\Sigma}$.

Proof. (⇒) follows by defining cospans corresponding each part of Definition 47 and composing them together: since composition of cospans is by pushout, the DPO diagram can be recovered and the pushouts checked to be traced boundary complements. (⇐) follows by using Lemma 64 and the fullness of $\langle\!\langle - \rangle\!\rangle_{\Sigma}$ to obtain the pieces of Definition 47. ◀

5.2 Rewriting with traced comonoid structure

To extend rewriting with traced structure to the comonoid case, the traced boundary complement conditions need to be weakened to the case of *left*-monogamous cospans.

▶ **Definition 66** (Traced left-boundary complement). *For partial left-monogamous cospans* $i \xrightarrow{a_1} L \xleftarrow{a_2} j$ *and* $n \xrightarrow{b_1} G \xleftarrow{b_2} m \in \mathbf{Hyp}_\Sigma$*, a pushout complement as in Definition 58 is called a* traced left-boundary complement *if* c_2 *is mono and* $j + m \xrightarrow{[c_2,d_1]} C \xleftarrow{[c_1,d_2]} i + n$ *is a partial left-monogamous cospan.*

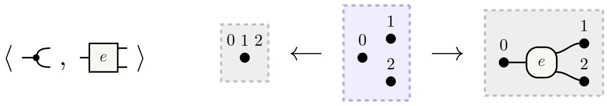
▶ **Definition 67** (Traced comonoid DPO). *For morphisms* $G \leftarrow m + n$ *and* $H \leftarrow m + n$ *in* $\mathbf{Hyp}_\Sigma$*, there is a traced comonoid rewrite* $G \rightsquigarrow_\mathcal{R} H$ *if there exists a rule* $L \leftarrow i + j \rightarrow G \in \mathcal{R}$ *and cospan* $i + j \rightarrow C \leftarrow n + m \in \mathbf{Hyp}_\Sigma$ *such that diagram in Definition 49 commutes and* $i + j \rightarrow C \rightarrow G$ *is a traced left-boundary complement.*

▶ **Lemma 68** (Traced comonoid decomposition). *Lemma 64 holds when all cospans are partial left-monogamous and* $j + m \xrightarrow{[c_2,d_1]} C \xleftarrow{[c_1,d_2]} i + n$ *is a traced left-boundary complement.*

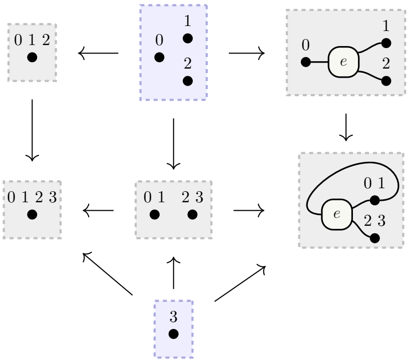
▶ **Theorem 69.** *Let* $\mathcal{R}$ *be a rewriting system on* $\mathbf{T}_\Sigma + \mathbf{CComon}$*. Then,* $-\boxed{g}- \Rightarrow_\mathcal{R} -\boxed{h}-$ *in* $\mathbf{T}_\Sigma + \mathbf{CComon}$ *if and only if* $\langle\!\langle \ulcorner \lfloor -\boxed{g}- \rfloor_\Sigma \urcorner \rangle\!\rangle_\Sigma \rightsquigarrow_{\langle\!\langle \ulcorner \lfloor \mathcal{R} \rfloor_\Sigma \urcorner \rangle\!\rangle_\Sigma} \langle\!\langle \ulcorner \lfloor -\boxed{h}- \rfloor_\Sigma \urcorner \rangle\!\rangle_\Sigma$*..*

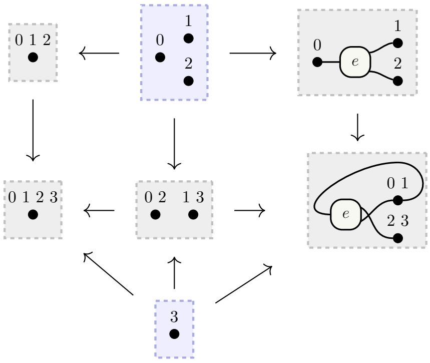
Proof. As Theorem 65, but with traced left-boundary complements. ◀

▶ **Example 70.** As with the traced case, there may be multiple valid rewrites given a particular interface. The comonoid structure adds more possibilities, as there are the equations of commutative comonoids to consider. Consider the following rule and its interpretation.

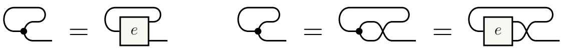


Two valid rewrites are as follows:





The first rewrite is the “obvious” one, but the second also holds by cocommutativity:



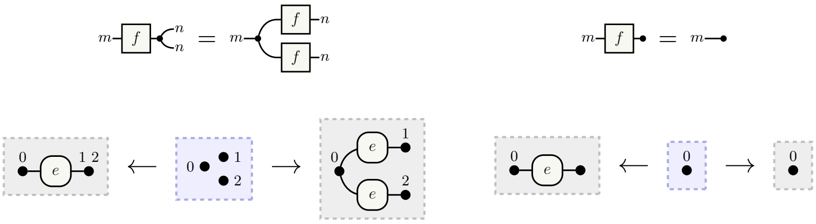


Figure 11 Equations of the monoidal theory $\mathbf{Cart}_{\mathcal{C}}$, where $m\text{-}\boxed{f}\text{-}n$ is an arbitrary morphism in $\mathcal{C}$, and the interpretations of these equations as rewrite rules for an arbitrary generator e.

6 Case studies

6.1 Cartesian structure

One important class of categories with a traced comonoid structure are *traced Cartesian*, or *dataflow*, categories [8, 15]. These categories are interesting because any traced Cartesian category admits a fixpoint operator [15, Thm. 3.1].

▶ **Definition 71** (Cartesian category [12]). *A monoidal category is* Cartesian *if its tensor is given by the Cartesian product.*

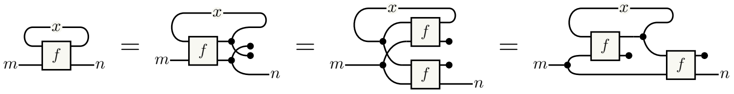
As a result of this, the unit is a terminal object in any Cartesian category, and any object has a comonoid structure. Cartesian categories are settings in which objects can be *copied* and *discarded*. These two operations are more clearly illustrated when viewed through the lens of a monoidal theory.

▶ **Definition 72.** *For a given base PROP $\mathbf{T}_{\Sigma_{\mathbf{C}}}$ with a comonoid structure, the monoidal theory $(\Sigma_{\mathbf{Cart_C}}, \mathcal{E}_{\mathbf{Cart_C}})$ is defined with $\Sigma_{\mathbf{Cart_C}} := \Sigma_{\mathbf{C}}$ and $\mathcal{E}_{\mathbf{Cart_C}}$ as the equations in Figure 11.*

Note that as the equations in $\mathcal{E}_{\mathbf{Cart_C}}$ are parameterised over any morphism $m\text{-}\boxed{f}\text{-}n$, a separate DPO rewrite rule is required for every combination of generators as in Figure 11. However, as is the case in the next section, it is often possible to characterise the copying behaviour through a finite number of equations.

▶ Remark 73. The combination of Cartesian equations with the underlying compact closed structure of $\mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$ may prompt alarm bells, as a compact closed category in which the tensor is the Cartesian product is trivial. However, it is important to note that $\mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$ is *not* subject to these equations: it is only a setting for performing graph rewrites.

Reasoning about fixpoints can be performed using the *unfolding* rule, which holds in any traced Cartesian category.



In the syntactic setting, this requires the application of multiple equations: the two counitality equations followed by the copy equation and optionally some axioms of STMCs for housekeeping. However, if we interpret this in the hypergraph setting, the comonoid equations are absorbed into the notation so only the copy equation needs to be applied.

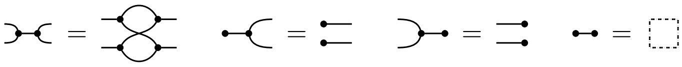
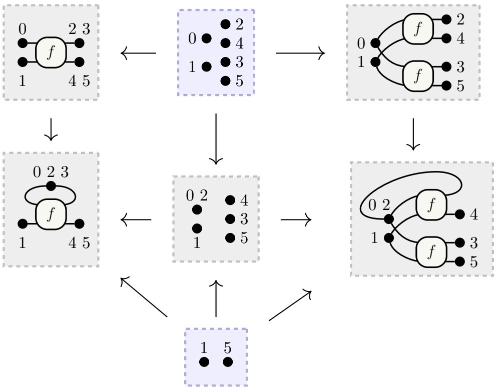


Figure 12 Equations $\mathcal{E}_{\mathbf{Bialg}}$ of a *bialgebra*, in addition to those in Figures 1 and 2.



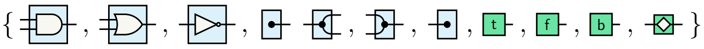
The dual notion of traced *cocartesian* categories [2] are also important in computer science: a trace in a traced cocartesian category corresponds to *iteration* in the context of *control flow.* The details of this section could also be applied to the cocartesian case by flipping all the directions and working with partial *right*-monogamous cospans.

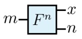
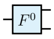
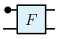
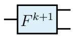
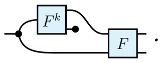
However, attempting to combine the product and coproduct approaches for settings with a *biproduct* would simply yield the category $\mathsf{Csp}_D(\mathbf{Hyp}_\Sigma)$, a hypergraph category (Proposition 18) subject to the Frobenius equations in Figure 3. A category with biproducts is not necessarily subject to such equations, so this would not be a suitable approach.

6.2 Digital circuits

As mentioned above, traced Cartesian categories are useful for reasoning in settings with fixpoint operators. One such setting is that of *digital circuits* built from primitive logic gates: in [14], digital circuits are modelled as morphisms in a STMC. Here, the trace models a feedback loop, and the comonoid structure represents forking wires. The semantics of digital circuits can be expressed as a monoidal theory [14, Sec. 6].

▶ **Definition 74** (Gate-level circuits). *Let the monoidal theory of* gate-level sequential circuits *be defined as* $(\Sigma_{\mathbf{SCirc}}, \mathcal{E}_{\mathbf{SCirc}})$*, where*

$\Sigma_{\mathbf{SCirc}} :=$ 

and the equations of $\mathcal{E}_{\mathbf{SCirc}}$ *are listed in Figures 1, 2, 12, and 13, where* $[\![-]\!]^{\mathbf{G}}$ *maps gates to the corresponding truth table,* $\sqcup$ *is the join in a lattice structure on* $\{\bullet, \mathsf{t}, \mathsf{f}, \mathsf{b}\}$*, and*  *is defined inductively as*  $:=$  *and*  $:=$ .

The generators in $\Sigma_{\mathbf{SCirc}}$ are, respectively: AND, OR and NOT gates; constructs for introducing, forking, joining and stubbing wires; *values* representing a true signal, a false signal, and both signals at once; and a delay of one unit of time. Note that while the equations in $\mathcal{E}_{\mathbf{SCirc}}$ contain those of a commutative comonoid, they do *not* explicitly contain the general Cartesian equations: instead, these are derived from smaller equations.

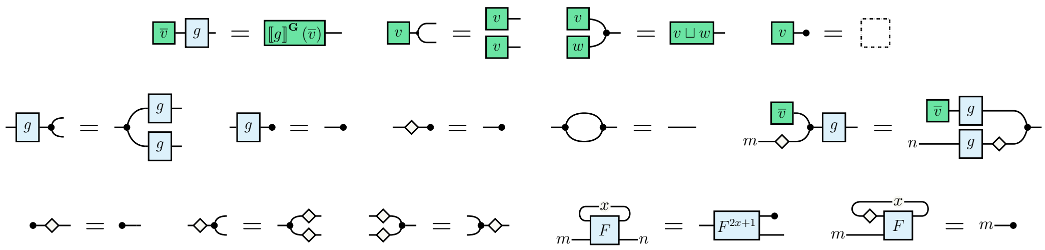


Figure 13 The equations of $\mathcal{E}_{\mathbf{SCirc}}$, from the monoidal theory of gate-level sequential circuits.

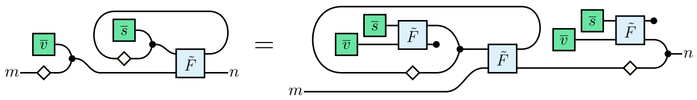
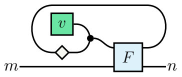


Figure 14 The cycle equation, which is derivable from the equations in $\mathcal{E}_{\mathbf{SCirc}}$.

Using graph rewriting, we can sketch out an *operational semantics* for sequential circuits. For the interests of brevity, we will only consider circuits of the form  : circuits with no "non-delay-guarded feedback" in which the registers of the circuit have been isolated from a core containing only "blue" (*combinational*) components, which models a function. Any sequential circuit can be translated into such a form by the equational theory.

We can "apply" such a circuit to an input as shown in the left-hand side of Figure 14; the equations in $\mathcal{E}_{\mathbf{SCirc}}$ can be used to derive the right-hand side. The four equations in the top row of Figure 13 can then be repeatedly applied to reduce the two "new" cores down to values, representing the output and new state of the circuit.

When the circuits are interpreted as hypergraphs and the equations as rewrites, a computer could perform this sequence of rewrites in order to evaluate circuits in a step-by-step manner.

7 Conclusion, related and further work

We have shown how the frameworks for rewriting string diagrams modulo Frobenius [4] and symmetric monoidal [5] structure using hypergraphs can also be adapted for rewriting modulo traced comonoid structure, by using hypergraphs that sit between the two settings.

Graphical languages for traced categories have seen many applications, such as to illustrate cyclic lambda calculi [15], or to reason graphically about programs [29]. The presentation of traced categories as *string diagrams* has existed since the 90s [19, 20]; a soundness and completeness theorem for traced string diagrams, folklore for many years but only proven for certain signatures [30], was finally shown in [22]. Combinatorial languages predate even this, having existed since at least the 80s in the guise of *flowchart schemes* [33, 8, 9]. These diagrams have also been used to show the completeness of finite dimensional vector spaces [17] with respect to traced categories and, when equipped with a dagger, Hilbert spaces [31].

We are not just concerned with diagrammatic languages as a standalone concept: we are interested in performing *graph rewriting* with them to reason about monoidal theories. This has been been studied in the context of traced categories before using *string graphs* [21, 10]. We have instead opted to use the *hypergraph* framework of [4, 5, 6] instead, as it allows rewriting modulo *yanking*, is more extensible for rewriting modulo comonoid structure, and one does not need to awkwardly reason modulo wire homeomorphisms.

As mentioned during the case studies, there are still elements of the rewriting framework that are somewhat informal. One such issue involves defining rewrite spans for arbitrary subgraphs: this is hard to do at a general level because the edges must be concretely specified in DPO rewriting. However, if we performed rewriting with *hierarchical hypergraphs* [1], in which edges can have hypergraphs as labels, we could "compress" the subgraph into a single edge that can be rewritten: this is future work.

In regular PROP notation, wires are annotated with numbers in order to avoid drawing multiple wires in parallel: when interpreted as hypergraphs a vertex is created for each wire, and simple diagrams can quickly get very large. The results of [5] also extend to the multi-sorted case, in which vertices are labelled in addition to wires. We could use this in combination with the *strictifiers* of [32]: these are additional generators for transforming buses of wires into thinner or thicker ones. This could drastically reduce the number of elements in a hypergraph, which is ideal from a computational point of view. Work has already begun on implementing the rewriting system for digital circuits using these techniques.

References

1 Mario Alvarez-Picallo, Dan Ghica, David Sprunger, and Fabio Zanasi. Functorial String Diagrams for Reverse-Mode Automatic Differentiation. In Bartek Klin and Elaine Pimentel, editors, *31st EACSL Annual Conference on Computer Science Logic (CSL 2023)*, volume 252 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:20, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2023.6.

2 E. S. Bainbridge. Feedback and generalized logic. *Information and Control*, 31(1):75–96, May 1976. doi:10.1016/S0019-9958(76)90390-9.

3 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. Rewriting modulo symmetric monoidal structure. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, pages 710–719, New York, NY, USA, July 2016. Association for Computing Machinery. doi:10.1145/2933575.2935316.

4 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobociński, and Fabio Zanasi. String Diagram Rewrite Theory I: Rewriting with Frobenius Structure. *Journal of the ACM*, 69(2): 14:1–14:58, March 2022. doi:10.1145/3502719.

5 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. String diagram rewrite theory II: Rewriting with symmetric monoidal structure. *Mathematical Structures in Computer Science*, 32(4):511–541, April 2022. doi:10.1017/S0960129522000317.

6 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. String diagram rewrite theory III: Confluence with and without Frobenius. *Mathematical Structures in Computer Science*, pages 1–41, June 2022. doi:10.1017/S0960129522000123.

7 A. Carboni and R. F. C. Walters. Cartesian bicategories I. *Journal of Pure and Applied Algebra*, 49(1):11–32, November 1987. doi:10.1016/0022-4049(87)90121-6.

8 Virgil Emil Căzănescu and Gheorghe Ştefănescu. Towards a New Algebraic Foundation of Flowchart Scheme Theory. *Fundamenta Informaticae*, 13(2):171–210, January 1990. doi:10.3233/FI-1990-13204.

9 Virgil Emil Căzănescu and Gheorghe Ştefănescu. Feedback, Iteration, and Repetition. In *Mathematical Aspects of Natural and Formal Languages*, volume Volume 43 of *World Scientific Series in Computer Science*, pages 43–61. World Scientific, October 1994. doi:10.1142/9789814447133_0003.

10 Lucas Dixon and Aleks Kissinger. Open-graphs and monoidal theories. *Mathematical Structures in Computer Science*, 23(2):308–359, April 2013. doi:10.1017/S0960129512000138.

11 Brendan Fong and David I. Spivak. Hypergraph categories. *Journal of Pure and Applied Algebra*, 223(11):4746–4777, November 2019. doi:10.1016/j.jpaa.2019.02.014.

12 Thomas Fox. Coalgebras and cartesian categories. *Communications in Algebra*, 4(7):665–667, January 1976. doi:10.1080/00927877608822127.

13 Tobias Fritz and Wendong Liang. Free gs-Monoidal Categories and Free Markov Categories. *Applied Categorical Structures*, 31(2):21, April 2023. doi:10.1007/s10485-023-09717-0.

14 Dan R. Ghica, George Kaye, and David Sprunger. A compositional theory of digital circuits. *CoRR*, abs/2201.10456, February 2023. doi:10.48550/arXiv.2201.10456.

15 Masahito Hasegawa. Recursion from cyclic sharing: Traced monoidal categories and models of cyclic lambda calculi. In Philippe de Groote and J. Roger Hindley, editors, *Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science, pages 196–213, Berlin, Heidelberg, 1997. Springer. doi:10.1007/3-540-62688-3_37.

16 Masahito Hasegawa. On traced monoidal closed categories. *Mathematical Structures in Computer Science*, 19(2):217–244, April 2009. doi:10.1017/S0960129508007184.

17 Masahito Hasegawa, Martin Hofmann, and Gordon Plotkin. Finite Dimensional Vector Spaces Are Complete for Traced Symmetric Monoidal Categories. In Arnon Avron, Nachum Dershowitz, and Alexander Rabinovich, editors, *Pillars of Computer Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, Lecture Notes in Computer Science, pages 367–385. Springer, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-78127-1_20.

18 Marvin Heumüller, Salil Joshi, Barbara König, and Jan Stückrath. Construction of Pushout Complements in the Category of Hypergraphs. *Electronic Communications of the EASST*, 39(0), September 2011. doi:10.14279/tuj.eceasst.39.647.

19 André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, 1991. doi:10.1016/0001-8708(91)90003-P.

20 André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):447–468, April 1996. doi:10.1017/S0305004100074338.

21 Aleks Kissinger. *Pictures of Processes: Automated Graph Rewriting for Monoidal Categories and Applications to Quantum Computing.* PhD thesis, University of Oxford, March 2012. doi:10.48550/arXiv.1203.0202.

22 Aleks Kissinger. Abstract Tensor Systems as Monoidal Categories. In Claudia Casadio, Bob Coecke, Michael Moortgat, and Philip Scott, editors, *Categories and Types in Logic, Language, and Physics: Essays Dedicated to Jim Lambek on the Occasion of His 90th Birthday*, Lecture Notes in Computer Science, pages 235–252. Springer, Berlin, Heidelberg, 2014. doi:10.1007/978-3-642-54789-8_13.

23 Stephen Lack and Paweł Sobociński. Adhesive Categories. In Igor Walukiewicz, editor, *Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science, pages 273–288, Berlin, Heidelberg, 2004. Springer. doi:10.1007/978-3-540-24727-2_20.

24 Stephen Lack and Paweł Sobociński. Adhesive and quasiadhesive categories. *RAIRO - Theoretical Informatics and Applications*, 39(3):511–545, July 2005. doi:10.1051/ita:2005028.

25 F. William Lawvere. Functorial semantics of algebraic theories. *Proceedings of the National Academy of Sciences*, 50(5):869–872, November 1963. doi:10.1073/pnas.50.5.869.

26 Saunders MacLane. Categorical algebra. *Bulletin of the American Mathematical Society*, 71(1):40–106, 1965. doi:10.1090/S0002-9904-1965-11234-4.

27 Aleksandar Milosavljevic, Robin Piedeleu, and Fabio Zanasi. String Diagram Rewriting Modulo Commutative (Co)monoid Structure. *CoRR*, abs/2204.04274, March 2023. doi:10.48550/arXiv.2204.04274.

28 R. Rosebrugh, N. Sabadini, and R. F. C. Walters. Generic commutative separable algebras and cospans of graphs. *Theory and Applications of Categories*, 15:164–177, 2005.

29 Ralf Schweimeier and Alan Jeffrey. A Categorical and Graphical Treatment of Closure Conversion. *Electronic Notes in Theoretical Computer Science*, 20:481–511, January 1999. doi:10.1016/S1571-0661(04)80090-2.

30 Peter Selinger. A Survey of Graphical Languages for Monoidal Categories. In Bob Coecke, editor, *New Structures for Physics*, Lecture Notes in Physics, pages 289–355. Springer, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-12821-9_4.

31 Peter Selinger. Finite dimensional Hilbert spaces are complete for dagger compact closed categories. *Logical Methods in Computer Science*, 8(3), August 2012. doi:10.2168/LMCS-8(3:6)2012.

32 Paul Wilson, Dan Ghica, and Fabio Zanasi. String Diagrams for Non-Strict Monoidal Categories. In Bartek Klin and Elaine Pimentel, editors, *31st EACSL Annual Conference on Computer Science Logic (CSL 2023)*, volume 252 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 37:1–37:19, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2023.37.

33 Gheorghe Ştefănescu. Feedback Theories (A Calculus for Isomorphism Classes of Flowchart Schemes). *Romanian Journal of Pure and Applied Mathematics*, 35(1):73–79, 1990.

Cost–Size Semantics for Call-By-Value Higher-Order Rewriting

Cynthia Kop
Institute for Computing and Information Sciences, Radboud University, Nijmegen, The Netherlands

Deivid Vale
Institute for Computing and Information Sciences, Radboud University, Nijmegen, The Netherlands

Abstract

Higher-order rewriting is a framework in which higher-order programs can be described by transformation rules on expressions. A computation occurs by transforming an expression into another using such rules. This step-by-step computation model induced by rewriting naturally gives rise to a notion of complexity as the number of steps needed to reduce expressions to a normal form, i.e., an expression that cannot be reduced further. The study of *complexity analysis* focuses on the development of automatable techniques to provide bounds to this number. In this paper, we consider a form of higher-order rewriting with a call-by-value evaluation strategy, so as to model call-by-value programs. We provide a cost–size semantics: a class of algebraic interpretations to map terms to tuples which bound both the reduction cost and the size of normal forms.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting

Keywords and phrases Call-by-Value Evaluation, Complexity Theory, Higher-Order Rewriting

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.15

Funding This work is supported by the NWO TOP project "Implicit Complexity through Higher-Order Rewriting", NWO 612.001.803/7571 and the NWO VIDI project "Constrained Higher-Order Rewriting and Program Equivalence", NWO VI.Vidi.193.075.

1 Introduction

Term rewriting is a logical framework that, among other applications, provides a computational model to specify algorithms. Simple programs (especially functional programs) can typically be modeled as a term rewriting system where a program state is expressed as a term and evaluation is modeled by rewriting expressions using reduction rules. *Higher-order* term rewriting in particular provides a natural model for functional programming languages. Due to the abstract nature of rewriting, it is feasible to forgo specific language details and still derive useful term rewriting results that may carry over to program analysis [3, 10, 15, 24].

In this paper, we study *complexity*, which in the context of term rewriting is typically understood as the number of steps needed to reach a normal from when starting in terms of a certain shape and size. A natural way to determine these bounds is adapting termination proof techniques to deduce the complexity. There is a myriad of works following this idea. To mention a few, see [4, 7, 9, 18, 19, 25] for interpretation methods, [8, 17, 31] for lexicographic and path orders, and [16, 28] for dependency pairs.

However, those ideas are focused on *first-order* term rewriting. There is very little work on complexity of *higher-order* term rewriting. While there is a lot of work on complexity of functional programs [2, 13, 20, 26], this work uses quite different ideas from the methods developed for term rewriting. It would be beneficial to combine these ideas.

In a previous work [21], we introduced an extension of the method of *weakly monotonic algebras* [14, 29] to *tuple interpretations*. The idea of algebras is to choose an interpretation domain A, and interpret terms s as elements $[\![s]\!]$ of A compositionally, in such a way that

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 15; pp. 15:1–15:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

whenever $s \to t$ we have $[\![s]\!] > [\![t]\!]$. Hence, a rewriting step on terms implies a strict decrease on A. The defining characteristic of tuple interpretations is to split the complexity measure into abstract notions of cost and size. This coincides with ideas often used in resource analysis of functional programs [2, 13]. This is a popular idea, as a very similar approach was introduced for first-order rewriting around the same time [33].

This previous work considered full higher-order rewriting, so without an evaluation strategy. However, this is not a very realistic setting, especially with the goal of eventually extending the methodology to various functional programming languages. In practice, program evaluation is deterministic, i.e., it follows a specific strategy such as call-by-value evaluation. Reduction below a λ-binder is also not usually allowed. The difference can be substantial: for instance for a pair of rules $\mathsf{f}\ x\ \mathsf{0} \to x$, $\mathsf{f}\ x\ (\mathsf{s}\ y) \to \mathsf{f}\ (\mathsf{pair}\ x\ x)\ y$, if x is instantiated by a term that is not in normal form, the complexity is linear if we evaluate call-by-value, and exponential with an arbitrary evaluation strategy. Also in complexity analysis of first-order term rewriting, considering innermost evaluation is common [27, 28].

In this paper, our goal is to extend the work of [21] to *weak call-by-value* reduction. To our knowledge, this is the first complexity method for higher-order term rewriting with an evaluation strategy. While the restriction of the strategy leads to tighter complexity bounds, the definitions needed to obtain these bounds are much more intricate, largely due to the potential for rules and β-redexes of higher type. We believe that this will bring the method of weakly monotonic algebras closer to the reality of functional program analysis.

Tuple interpretations do not provide a complete termination proof method: there are terminating systems for which interpretations cannot be found. Consequently, it does not induce a complete complexity analysis framework either. Notwithstanding, it has the potential to be very powerful if we choose the cost–size sets wisely. A second limitation is that the question whether a suitable interpretation exists is undecidable in general, which is expected already in the polynomial case [23]. Undecidability never hindered computer scientists' efforts on mechanizing difficult problems, however. Indeed, several proof search methods have been developed over the years to find interpretations automatically [6, 11, 12, 18, 33].

Contribution. This paper will introduce tuple interpretations for higher-order term rewriting systems using a weak call-by-value evaluation strategy, and use them to define both a termination method under this strategy, and a new definition of weak call-by-value runtime complexity along with a methodology to derive bounds for it.

This paper builds on the ideas of [21], which introduces tuple interpretations and a notion of runtime complexity for full higher-order rewriting (without evaluation strategy). The key difference here is our focus on a weak call-by-value evaluation strategy. This allows for tighter bounds, but also requires significant technical changes. since the "cost" for a term of higher type can no longer be captured by just a function (as we will explain in Section 3).

An additional change compared to [21] is that we have separated the cost and size components into distinct functions. In [21], it is in theory allowed for the size component to depend on the cost component, even though in practice this never happened. By fully separating the components, it is easier to prove correctness of a given tuple interpretation.

Paper Overview. In Section 2 we review basic notation on higher-order rewriting and define our notion of call-by-value strategy. In Section 3 we give an informal overview of how the technique works. These ideas are formalized in Sections 4 and 5 where we respectively provide a formal cost–size semantics for simple types, and interpret terms as cost–size tuples as well as proving some basic properties of them. We provide additional examples in Section 6. In Section 7 we conclude the paper and discuss future work.

2 Preliminaries

Unlike first-order rewriting, there is no single consensus formalism for higher-order rewriting, but rather a variety of sometimes incompatible formats. The formalism we consider here is a style of simply typed lambda calculus extended with function symbols and rules. The matching mechanism is modulo alpha, and beta reduction is included in the rewriting relation. This is essentially the formalism used in the higher-order category of the international termination competition [32], but slightly simplified for easier representation.[1]

Types, Terms, and Equality. Let $\mathbb{B}$ be a nonempty set whose elements are called *base types* and range over ι, κ, ν. The set $\mathbb{T}_{\mathbb{B}}$ of *simple types* over $\mathbb{B}$ is generated by the grammar: $\mathbb{T}_{\mathbb{B}} := \mathbb{B} \mid \mathbb{T}_{\mathbb{B}} \Rightarrow \mathbb{T}_{\mathbb{B}}$. Types from $\mathbb{T}_{\mathbb{B}}$ range over σ, τ, ρ. The $\Rightarrow$ type constructor is right-associative, so we write $\sigma \Rightarrow \tau \Rightarrow \rho$ for $(\sigma \Rightarrow (\tau \Rightarrow \rho))$. Notice that every simple type σ can be written as $\tau_1 \Rightarrow \cdots \Rightarrow \tau_n \Rightarrow \iota$. We informally say that the τ_i's are the *input types* and the base type ι is the *output type*. We abbreviate such types by $\boldsymbol{\tau} \Rightarrow \iota$. The *type order* of a type is the number: (a) $\mathtt{ord}(\iota) = 0$ and (b) $\mathtt{ord}(\sigma \Rightarrow \tau) = \max(1 + \mathtt{ord}(\sigma), \mathtt{ord}(\tau))$. A *signature* $\mathbb{F}$ is a triple $(\mathbb{B}, \Sigma, \mathtt{ar})$ where $\mathbb{B}$ is a set of base types, Σ is a nonempty finite set of symbols, and $\mathtt{ar}$ is a function $\mathtt{ar} : \Sigma \longrightarrow \mathbb{T}_{\mathbb{B}}$. For each type σ, we postulate the existence of a nonempty set $\mathbb{X}_\sigma$ of countably many variables. Furthermore, we impose that $\mathbb{X}_\sigma \cap \mathbb{X}_\tau = \emptyset$ whenever $\sigma \neq \tau$. Let $\mathbb{X}$ denote the family of sets $(\mathbb{X}_\sigma)_{\sigma \in \mathbb{T}_{\mathbb{B}}}$ indexed by $\mathbb{T}_{\mathbb{B}}$ and assume that $\Sigma \cap \mathbb{X} = \emptyset$.

The set $\mathsf{T}(\mathbb{F}, \mathbb{X})$ – of terms built from $\mathbb{F}$ and $\mathbb{X}$ – collects those expressions s for which the judgment $s : \sigma$ can be deduced using the following rules:

$$\frac{x \in \mathbb{X}_\sigma}{x : \sigma} \qquad \frac{\mathsf{f} \in \Sigma \qquad \mathtt{ar}(\mathsf{f}) = \sigma}{\mathsf{f} : \sigma} \qquad \frac{s : \sigma \Rightarrow \tau \qquad t : \sigma}{(s\,t) : \tau} \qquad \frac{x \in \mathbb{X}_\sigma \qquad s : \tau}{(\lambda x.\, s) : \sigma \Rightarrow \tau}$$

Application of terms is left-associative, so we write $s\,t\,u$ for $((s\,t)\,u)$. Abstraction is right-associative, so we write $\lambda xyz.\, s$ for $\lambda x.\, (\lambda y.\, (\lambda z.\, s))$. Application takes precedence over abstraction, which allows us to write $\lambda x.\, s\,t$ for $\lambda x.\, (s\,t)$. Unnecessary parentheses are removed, and we write terms following these rules. The set $\mathtt{fv}(s)$ of *free variables* occurring in s is defined as expected. A term s is *closed* if $\mathtt{fv}(s) = \emptyset$. It is *ground* if no variable occurs in it. A symbol $\mathsf{f} \in \Sigma$ is called the *head symbol* of s if $s = \mathsf{f}\, s_1 \ldots s_k$. A *subterm* of s is a term t (we write $s \trianglerighteq t$) such that (i) $s = t$; or (ii) t is a subterm of s' or s'', if $s = s'\, s''$; or t is a subterm of s', if $s = \lambda x.\, s'$. A *proper subterm* of s is a subterm of s which is not equal to s.

A *substitution* γ is a type-preserving map from variables to terms such that the set $\mathtt{dom}(\gamma) = \{x \in \mathbb{X} \mid \gamma(x) \neq x\}$ is finite. We may explicitly represent γ as a list of mappings $[x_1 := s_1, \ldots, x_k := s_k]$. The *capture avoiding* application of γ to s is defined as follows:

$$\begin{aligned} x\gamma &= \gamma(x) & (s\,t)\gamma &= (s\gamma)\,(t\gamma) \\ \mathsf{f}\gamma &= \mathsf{f} & (\lambda x.\, s)\gamma &= \lambda y.\, (s^{\{x \mapsto y\}}\gamma), \text{ for } y \text{ fresh} \end{aligned}$$

Here, $s^{\{x \mapsto y\}}$ denotes the term obtained by replacing every free occurrence of x by y in s. The result of $s\gamma$ is unique modulo α-renaming. We identify terms modulo α-equality, so $s = t$ denotes $s =_\alpha t$.

[1] The format in the competition allows both function application and application as separate notions, admitting the formation of terms such as $\mathsf{f}(s) \cdot t$. We here omit the functional notation, which is not necessary since any term can be represented in a curried form. Beyond this, the formalism is the same, including the permissiveness that left-hand sides do not need to be patterns or even in β-normal form.

Higher-Order Rewriting. A *rewrite rule* $\ell \to r$ is a pair of terms of the same type such that $\ell = \mathsf{f}\,\ell_1 \dots \ell_k$ and $\mathtt{fv}(r) \subseteq \mathtt{fv}(\ell)$. A *term rewriting system* (TRS)[2] $\mathbb{R}$ is a set of rules. A relation $\to$ on terms is *monotonic* if $s \to s'$ implies $t\,s \to t\,s'$, $s\,u \to s'\,u$, and $\lambda x.\,s \to \lambda x.\,s'$; for all terms t and u of appropriate types. The *rewrite relation* $\to_{\mathbb{R}}$ induced by $\mathbb{R}$ is the smallest monotonic relation containing $\mathbb{R}$ union the β rule-scheme (i.e., $(\lambda x.\,s)\,t \to_\beta s[x := t]$) and closed under application of substitution. An $\mathbb{R}$-*reducible expression* (redex) is a term of form $\ell\gamma$ for some rule $\ell \to r$ and substitution γ. A β-redex is of the form $(\lambda x.\,s)\,t$.

Every rewrite rule $\ell \to r$ *defines* a symbol f, namely, the head symbol of ℓ. For each $\mathsf{f} \in \Sigma$, let $\mathbb{R}_\mathsf{f}$ denote the set of rewrite rules that define f in $\mathbb{R}$. A symbol $\mathsf{f} \in \Sigma$ is a *defined symbol* if $\mathbb{R}_\mathsf{f} \neq \emptyset$. A *constructor symbol* is a symbol $\mathsf{c} \in \Sigma$ such that $\mathbb{R}_\mathsf{f} = \emptyset$. We let Σ^{def} be the set of defined symbols and Σ^{con} the set of constructor symbols. Hence, $\Sigma = \Sigma^{\mathsf{def}} \uplus \Sigma^{\mathsf{con}}$. A *ground constructor term* is a term $\mathsf{c}\,s_1 \dots s_n$ with $n \geq 0$, where each s_i is a ground constructor term.

▶ **Example 1.** In this example we collect some common higher-order functions encoded as rules: map applies a function to each element of a list; comp composes two functions, app is the application functional, and rec encodes primitive recursion. Their monomorphic signature is defined as expected with functional arguments of type $\mathsf{nat} \Rightarrow \mathsf{nat}$ and lists having type list.

$$\begin{aligned}
&\mathsf{map}\,F\,\mathsf{nil} \to \mathsf{nil} && \mathsf{comp}\,F\,G \to \lambda x.\,F\,(G\,x)\\
&\mathsf{map}\,F\,(\mathsf{cons}\,x\,xs) \to \mathsf{cons}\,(F\,x)\,(\mathsf{map}\,F\,xs) && \mathsf{app}\,F \to \lambda x.\,F\,x\\
&\mathsf{rec}\,0\,y\,F \to y && \mathsf{rec}\,(\mathsf{s}\,x)\,y\,F \to F\,x\,(\mathsf{rec}\,x\,yF)
\end{aligned}$$

▶ **Example 2.** Some first-order functions over natural numbers:

$$\begin{aligned}
&\mathsf{dbl}\,0 \to 0 && \mathsf{add}\,x\,0 \to 0 && \mathsf{mult}\,x\,0 \to 0\\
&\mathsf{dbl}\,(\mathsf{s}\,x) \to \mathsf{s}(\mathsf{s}\,(\mathsf{dbl}\,x)) && \mathsf{add}\,x\,(\mathsf{s}\,y) \to \mathsf{s}\,(\mathsf{add}\,x\,y) && \mathsf{mult}\,x\,(\mathsf{s}\,y) \to \mathsf{add}\,x\,(\mathsf{mult}\,x\,y)
\end{aligned}$$

Call-by-Value Higher-order Rewriting. In this paper, we are interested in a restricted evaluation strategy, which limits reduction to terms whose immediate subterms are *values*:

▶ **Definition 3.** A term s is a *value* whenever s is:

- of the form $\mathsf{f}\,v_1 \dots v_n$, with each v_i a value and there is no rule $\mathsf{f}\,\ell_1 \dots \ell_k \to r$ with $k \leq n$;
- an abstraction, i.e., $s = \lambda x.\,t$.

Notice that by definition ground constructor terms are values, since there is no rule $\mathsf{c}\ \ell_1\ \dots\ \ell_k \to r$ for any k if $\mathsf{c} \in \Sigma^{\mathsf{con}}$. More complex values include partially applied functions and lambda-terms; for example, $\mathsf{add}\,0$ or a list of functions $[\mathsf{add}\,0; \lambda x.x; \mathsf{mult}\,0; \mathsf{dbl}]$. In the weak call-by-value reduction strategy defined below, we shall not reduce under abstractions.

▶ **Definition 4.** The **higher-order weak call-by-value rewrite relation** $\to_v$ induced by $\mathbb{R}$ is defined as follows:

- $\mathsf{f}\,(\ell_1\gamma) \dots (\ell_k\gamma) \to_v r\gamma$, if $\mathsf{f}\,\ell_1 \dots \ell_k \to r \in \mathbb{R}$ and each $\ell_i\gamma$ is a value;
- $(\lambda x.\,s)\,v \to_v s[x := v]$, if v is a value;
- $s\,t \to_v s'\,t$ if $s \to_v s'$; and $s\,t \to_v s\,t'$ if $t \to_v t'$.

[2] Note that we use the acronym TRS for the style of higher-order term rewriting systems introduced in this section; *not* for a limitation to first-order term rewriting systems as is sometimes done in the literature. In this paper, we will not consider first-order TRSs as a special case at all.

Notice that when instantiating rules we use *value substitutions*, that is, their image for any nontrivial variable is always a value. All reductions in this paper are weak call-by-value. So we drop the v from the arrow, and $s \rightarrow t$ should be read as $s \rightarrow_v t$. We use explicit notation whenever confusion may arise.

We say that a term s is in *normal form* if there is no term t such that $s \rightarrow_v t$. A term s *has* a normal form t if $s \rightarrow_v^* t$ and t is in normal form. A TRS $\mathbb{R}$ is *terminating* if no infinite rewrite sequence $s \rightarrow_v s_1 \rightarrow_v \dots$ exists.

Ordered Sets and Monotonic Functions. A *quasi-ordered set* $(A, \sqsupseteq)$ consists of a nonempty set A and a quasi-order (reflexive and transitive) $\sqsupseteq$ on A. An *extended well-founded set* $(A, >, \gtrsim)$ is a nonempty set A together with a well-founded order $>$ and a quasi-order $\gtrsim$ on A such that $\gtrsim$ is compatible with $>$, i.e., $x > y$ implies $x \gtrsim y$ and $x > y \gtrsim z$ implies $x > z$. Below we refer to an extended well-founded set simply as *well-founded set*. The `unit` set is the quasi-ordered set $(\{\mathtt{u}\}, \sqsupseteq)$, with $\mathtt{u} \sqsupseteq \mathtt{u}$.

Given quasi-ordered sets $(A, \sqsupseteq)$ and $(B, \sqsupseteq)$, a function $f : A \longrightarrow B$ is *weakly monotonic* if $x \sqsupseteq y$ implies $f(x) \sqsupseteq f(y)$. Let $A \Longrightarrow B$ denote the set of weakly monotonic functions from A to B. The comparison operator $\sqsupseteq$ on B induces point-wise comparison on $A \Longrightarrow B$ as follows: $f \sqsupseteq g$ if $f(x) \sqsupseteq g(x)$ for all $x \in A$. This way $(A \Longrightarrow B, \sqsupseteq)$ is also quasi-ordered. Given well-founded sets $(A, >, \gtrsim)$ and $(B, >, \gtrsim)$, a function $f : A \longrightarrow B$ is said to be *strongly monotonic* if $x > y$ implies $f(x) > f(y)$ and $x \gtrsim y$ implies $f(x) \gtrsim f(y)$.

3 Cost–Size Overview

In this section we sketch the broad idea of the methodology, focusing on intuition.

To start, every term is associated with a *size*. For a closed term of base type, this size could for instance be the number of symbols in its normal form; or a pair of integers, or a set of terms (e.g., the set of all normal forms of the term). We only require that each base type is associated with a *quasi-ordered set* with a minimum element. For a term of higher type, the size is a *weakly monotonic function*, which provides a bound for applications of the term.

▶ **Example 5.** In the signature of Examples 1 and 2, we may let $\mathcal{S}ize(\mathsf{0}) = 1$ and $\mathcal{S}ize(\mathsf{s}\ t) = 1 + \mathcal{S}ize(t)$; intuitively, the size of a ground constructor term of type nat is the number of function symbols in it. For lists, we could let $\mathcal{S}ize(\mathsf{nil}) = (0, 0)$ and $\mathcal{S}ize(\mathsf{cons}\ s\ t) = (\mathcal{S}ize(t)_1 + 1,\ \max(\mathcal{S}ize(s), \mathcal{S}ize(t)_2))$; intuitively, the size of a list of numbers is the pair (list length, size of its greatest element). We could let $\mathcal{S}ize(\mathsf{add}\ s)$ be the function that maps n to $\mathcal{S}ize(s) + n$, and $\mathcal{S}ize(\mathsf{map})$ the function that takes a (weakly monotonic) function F and a pair (l, m), and returns $(l, F(m))$; intuitively, if F bounds the size of the first argument, and we are given a list with maximum element of size m and length l, then applying map to these arguments yields a list which has length l, and elements have sizes bounded by $F(m)$.

Aside from a size, we need to calculate a *cost* for each term to associate a bound on the number of steps that can be taken from a given starting term. Aside from associating a natural number bounding this cost to each term, terms of higher type have computational content even in normal form; hence, we should associate a *cost function* to such terms: a weakly monotonic function that indicates the cost of applying this term to a value.

▶ **Example 6** (First idea for costs). Intuitively, the number of steps to evaluate $\mathsf{add}\ s\ t$ is bounded by the cost of evaluating the arguments, plus $\mathcal{S}ize(s)$ (as we easily see by inspecting the rules defining add). Hence, we would let $\mathcal{C}ost(\mathsf{add}\ s\ t) = \mathcal{C}ost(s) + \mathcal{C}ost(t) + \mathcal{S}ize(s)$, and could define $\mathcal{C}ost(\mathsf{add}) = \boldsymbol{\lambda}(c_1, s_1), (c_2, s_2).\ c_1 + c_2 + s_1$. Note that the cost function takes a *pair* of values for each argument: respectively, the cost and size of the argument.

For map, the number of steps to evaluate $\mathsf{map}\ s\ t$ depends heavily on s, even if both s and t are values: $\mathsf{map}\ (\lambda x.\mathsf{add}\ x\ \mathsf{0})\ t$ will take substantially fewer steps than evaluating $\mathsf{map}\ (\lambda x.\mathsf{mult}\ x\ x)\ t$. Hence, we should take the cost function for s into account as well as its size. This yields $\mathcal{C}ost(\mathsf{map}) = \boldsymbol{\lambda}(F_{cost}, F_{size}), (q_{cost}, (l, m)).\ q_{cost} + l + 1 + l * F_{cost}(0, m)$: the number of steps to evaluate $\mathsf{map}\ s\ t$ is bounded by the cost of evaluating t first, then applying s $\langle$*length of list*$\rangle$ times to the largest element of t, plus the $1 + \langle$*length of list*$\rangle$ steps for the evaluation of map itself. Note that since we use a call-by-value strategy, the list q is evaluated to a value *before* the map rule fires, which is why F_{cost} is given a zero argument.

The cost for constructor applications $\mathsf{c}\ s_1 \cdots s_m$ is always just $\mathcal{C}ost(s_1) + \cdots + \mathcal{C}ost(s_m)$, since applying a constructor to terms does not lead to a further computation being done.

Examples 5 and 6 sketch an idea where $\mathcal{S}ize(s\ t) = \mathcal{S}ize(s)(\mathcal{S}ize(t))$ and $\mathcal{C}ost(s\ t) = \mathcal{C}ost(s)(\mathcal{C}ost(t), \mathcal{S}ize(t))$. Unfortunately, while this idea works well for *sizes*, it has some issues for *costs*; most importantly, that the computational content of terms of higher types is ignored. Although a term $\lambda x.s$ cannot be reduced, a term such as $\mathsf{add}\ (\mathsf{dbl}\ \mathsf{0})$ can be, and the cost for the $\mathsf{dbl}\ \mathsf{0}$ reduction should be included. Moreover, terms of higher type can also reduce directly even when their subterms are values; e.g., $\mathsf{comp}\ s\ t$ or $(\lambda x.s)\ t$ of type $\mathsf{nat} \Rightarrow \mathsf{nat}$.

Hence, we will instead consider a *pair* of costs: each term has a *cost number* (a bound on the number of steps to reduce this term to normal form), and a *cost function* (which bounds the cost of applying this normal form to a value, or is $\mathtt{unit}$ for base-type terms).

Unfortunately, this choice necessarily imposes a more complicated definition, since a pair cannot be applied like a function can; e.g., if the cost of s is $(12, \boldsymbol{\lambda}(x_{cost}, x_{size}).\ x_{cost} + x_{size})$, then when computing the cost for $s\ t$, we cannot just apply the function and forget the 12. Hence, we will define (formally in Definition 16) an alternative interpretation of application, so that, for $s : \sigma \Rightarrow \tau$ and $t : \sigma$, $\mathcal{C}ost(s\ t) = (\ \mathcal{C}ost\mathcal{N}um(s) + \mathcal{C}ost\mathcal{N}um(t) + c,\ \text{fun}\ )$, where $\mathcal{C}ost\mathcal{F}un(s)(\mathcal{C}ost\mathcal{F}un(t), \mathcal{S}ize(t))$ is the pair (c, fun).

▶ **Example 7** (Cost pairs). We let $\mathcal{C}ost(\mathsf{add}) = (\ 0,\ \boldsymbol{\lambda}(u_1, n).\ (\ 0,\ \boldsymbol{\lambda}(u_2, m).\ n\ )\ )$: the first 0 is the "cost number" for add, which is 0 because add is in normal form; and the function $\boldsymbol{\lambda}(u_1, n).\ (\ 0,\ \boldsymbol{\lambda}(u_2, m).\ n\ )$ takes a unit element and the size of a value, and returns a new pair. With the rough definition of application above, we have $\mathcal{C}ost(\mathsf{add}\ s) = (\ \mathcal{C}ost\mathcal{N}um(s),\ \boldsymbol{\lambda}(u, n).\ (\ \mathcal{S}ize(s),\ \mathtt{u}\ )\ )$. This matches the intuition that the number of steps needed to reduce $\mathsf{add}\ s$ to normal form is just the number of steps needed to reduce s, and the result is a value of function type which, if applied to a value with size n, can be normalized in $\mathcal{S}ize(s)$ steps. We obtain $\mathcal{C}ost(\mathsf{add}\ s\ t) = \mathcal{C}ost(s) + \mathcal{C}ost(t) + \mathcal{S}ize(s)$ as expected.

The notation is rather cumbersome but is needed for the formal definition. In practice, we can identify $\mathtt{unit} \times A$ and $A \times \mathtt{unit}$ with A for any set, and use $(x_1, \ldots, x_n) \mapsto \varphi$ as shorthand for $(\ 0,\ \boldsymbol{\lambda}x_1.\ (\ 0,\ \boldsymbol{\lambda}x_2.\ldots\ \varphi\ )\ )$. Then we can use the more palatable notation $\mathcal{C}ost(\mathsf{add}) = (n, m) \mapsto n$, or $\mathcal{C}ost(\mathsf{comp}) = ((F_{cost}, F_{size}), (G_{cost}, G_{size})) \mapsto (\ 2, \boldsymbol{\lambda}x_{size}.G_{cost}(x_{size}) + F_{cost}(G_{size}(x_{size}))\ )$ for the symbol comp which admits a rule of higher type $\mathsf{nat} \Rightarrow \mathsf{nat}$.

With these definitions, if we can show that $(\mathcal{C}ost(\ell), \mathcal{S}ize(\ell)) \succ (\mathcal{C}ost(r), \mathcal{S}ize(r))$ for all *value instances* of rules, then $\mathcal{C}ost\mathcal{N}um(s)$ defines a bound on the number of steps that can be taken to reduce s to normal form. We can use this to define bounds on the runtime complexity of the rewriting system – that is, on the number of steps that can be done when starting in certain kinds of terms of a given size (as we will discuss in Section 6).

▶ **Example 8.** We choose $\mathcal{S}ize(\mathsf{nil})$, $\mathcal{S}ize(\mathsf{cons})$ and $\mathcal{S}ize(\mathsf{map})$ following Example 5, and let $\mathcal{C}ost(\mathsf{nil}) = 0$, $\mathcal{C}ost(\mathsf{cons}) = (n, m) \mapsto 0$ and $\mathcal{C}ost(\mathsf{map}) = ((F_{cost}, F_{size}), (l, m)) \mapsto l * F_{cost}(m) + l + 1$. Then, for a list $\mathsf{cons}\ h\ t$ with $\mathcal{S}ize(t) = (l, m)$, we have

$$\begin{array}{rcl} \mathcal{S}ize(\mathsf{map}\ F\ (\mathsf{cons}\ h\ t)) & = & (l+1, \mathcal{S}ize(F)(\max(\mathcal{S}ize(h)), m)) \\ & = & (l+1, \max(\mathcal{S}ize(F)(\mathcal{S}ize(h)), \mathcal{S}ize(F)(m))) \\ & = & \mathcal{S}ize(\mathsf{cons}\ (F\ h)\ (\mathsf{map}\ F\ t)) \end{array}$$

by weak monotonicity of $\mathcal{S}ize(F)$. Taking into account that if F, h and t are values, then they all have a cost number of 0, we also have:

$$\begin{array}{rcl} \mathcal{C}ost(\mathsf{map}\ F\ (\mathsf{cons}\ h\ t)) & = & (l+1) * \mathcal{C}ost\mathcal{F}un(F)(\max(\mathcal{S}ize(h), m)) + l + 2 \\ & > & \mathcal{C}ost\mathcal{F}un(F)(\mathcal{S}ize(h)) + l * \mathcal{C}ost\mathcal{F}un(F)(m) + l + 1 \\ & = & \mathcal{C}ost(\mathsf{cons}\ (F\ h)\ (\mathsf{map}\ F\ t)) \end{array}$$

Hence, all value instantiations of the left-hand side of this rule both have greater cost, and greater-than-or-equal size, to the right-hand sides. If the other rules are similarly oriented, we can conclude that $\mathcal{C}ost\mathcal{N}um(s)$ provides a bound on the reduction cost of s.

In the rest of this paper, the ideas above will be formally defined and their correctness proven. We will not use the elaborate names $\mathcal{C}ost\mathcal{N}um$, $\mathcal{S}ize$, etc., but rather define interpretations as tuples that contain all these components.

4 Cost–Size Semantics for Simple Types

In this section we build a set-theoretical cost–size semantics to the simple types in $\mathbb{T}_{\mathbb{B}}$. The goal is to define a function $(\!|\cdot|\!)$ that maps each type $\sigma \in \mathbb{T}_{\mathbb{B}}$ to a well-founded set $(\!|\sigma|\!)$, the cost–size interpretation of σ. We start by formally defining what we mean by cost–size sets.

▶ **Definition 9.** Given a well-founded set $(\mathcal{C}, >, \gtrsim)$, called the *cost set*, and a quasi-ordered set $(\mathcal{S}, \sqsupseteq)$, called the *size set*, we call $\mathcal{C} \times \mathcal{S}$ the **cost–size product** of $(\mathcal{C}, >, \gtrsim)$ and $(\mathcal{S}, \sqsupseteq)$, and its elements *cost–size tuples.*

Given a cost–size product $\mathcal{C} \times \mathcal{S}$, the well-foundedness of $\mathcal{C}$ and quasi-ordering on $\mathcal{S}$ naturally induce an order structure on the product $\mathcal{C} \times \mathcal{S}$ as follows.

▶ **Definition 10** (Product Order). Let $(\mathcal{C}, >, \gtrsim) \times (\mathcal{S}, \sqsupseteq)$ be a cost–size product. Then we define the relations $\succ, \succcurlyeq$ over $\mathcal{C} \times \mathcal{S}$ as follows: for all $\langle x, y\rangle$ and $\langle x', y'\rangle$ in $\mathcal{C} \times \mathcal{S}$,

- $\langle x, y\rangle \succ \langle x', y'\rangle$ iff $x > x'$ and $y \sqsupseteq y'$, and
- $\langle x, y\rangle \succcurlyeq \langle x', y'\rangle$ iff $x \gtrsim x'$ and $y \sqsupseteq y'$.

Next, we show that the triple $(\mathcal{C} \times \mathcal{S}, \succ, \succcurlyeq)$ is well-founded.

▶ **Lemma 11.** *The triple $(\mathcal{C} \times \mathcal{S}, \succ, \succcurlyeq)$ defined in Definition 10 is a well-founded set.*

Proof. It follows immediately from Definition 10 that $\succ, \succcurlyeq$ are transitive and $\succcurlyeq$ is reflexive. To show well-foundedness of $\succ$ we note that the existence of an infinite chain $\langle x_1, y_1\rangle \succ \langle x_2, y_2\rangle \succ \cdots$ would imply $x_1 > x_2 > \cdots$, which cannot be the case since $>$ is well-founded. We still need to check that $\succcurlyeq$ is compatible with $\succ$.

- Suppose $\langle x, y\rangle \succ \langle x', y'\rangle$. Since $x > x'$ implies $x \gtrsim x'$, we have $\langle x, y\rangle \succcurlyeq \langle x', y'\rangle$.
- Suppose $\langle x, y\rangle \succ \langle x', y'\rangle \succcurlyeq \langle x'', y''\rangle$. Since $x > x' \gtrsim x''$ implies $x > x''$ and $\sqsupseteq$ is transitive, we have $\langle x, y\rangle \succ \langle x'', y''\rangle$. ◀

We shall use product orders to induce well-founded ordering on cost–size sets. Let us define next the requirements for the sets used for size interpretations.

▶ **Definition 12** (Type Interpretation Key). Let $\mathbb{B}$ be a set of base types. An **interpretation key** for $\mathbb{B}$, denoted $\mathcal{J}_{\mathbb{B}}$, is a function that maps each base type $\iota \in \mathbb{B}$ to a quasi-ordered set $(\mathcal{J}_{\mathbb{B}}(\iota), \sqsupseteq)$ with a minimum element, i.e., it contains an element $\bot$, such that $x \sqsupseteq \bot$ for all x.

▶ **Example 13** (Cost–Size Tuples over natural numbers). A first example of an interpretation key is that of tuples over $\mathbb{N}$. For each $\iota \in \mathbb{B}$, $\mathcal{J}_{\mathbb{B}/\mathbb{N}}(\iota)$ is a set of the form $(\mathbb{N}^{K(\iota)}, \sqsupseteq)$, with $K(\iota) \geq 1$ and $(x_1, \dots, x_{K(\iota)}) \sqsupseteq (y_1, \dots, y_{K(\iota)})$ iff $x_i \geq y_i$ for all $1 \leq i \leq K(\iota)$. A minimum element for such sets is $(0, \dots, 0)$. Notice that $(\mathbb{N}^{K(\iota)}, \sqsupseteq)$ is quasi-ordered for any choice of $K(\iota)$ and $\mathcal{J}_{\mathbb{B}/\mathbb{N}}$ is completely determined by a function mapping each $\iota \in \mathbb{B}$ to $K(\iota) \in \mathbb{N}$.

The definition below formalizes our intuition for cost and size from Section 3. Given an interpretation key $\mathcal{J}_{\mathbb{B}}$ we inductively interpret the elements of $\mathbb{T}_{\mathbb{B}}$ as cost–size products.

▶ **Definition 14** (Interpretation of Types). Let $\mathcal{J}_{\mathbb{B}}$ be an interpretation key. We define for each type σ the **cost–size tuple interpretation** of σ as the set $(\!|\sigma|\!) = \mathcal{C}_\sigma \times \mathcal{S}_\sigma$ where $\mathcal{C}_\sigma$ and $\mathcal{S}_\sigma$ are defined as follows (mutually with the set $\mathcal{F}^{\mathsf{c}}_\sigma$):

$$\begin{aligned} \mathcal{C}_\sigma &= \mathbb{N} \times \mathcal{F}^{\mathsf{c}}_\sigma & \qquad \mathcal{S}_\iota &= \mathcal{J}_{\mathbb{B}}(\iota) \\ \mathcal{F}^{\mathsf{c}}_\iota &= \texttt{unit} & \mathcal{S}_{\sigma \Rightarrow \tau} &= \mathcal{S}_\sigma \Longrightarrow \mathcal{S}_\tau \\ \mathcal{F}^{\mathsf{c}}_{\sigma \Rightarrow \tau} &= (\mathcal{F}^{\mathsf{c}}_\sigma \times \mathcal{S}_\sigma) \Longrightarrow \mathcal{C}_\tau \end{aligned}$$

The set $(\!|\sigma|\!)$ is ordered as follows:
- $\langle (n, f_1), f_2 \rangle \succ \langle (m, g_1), g_2 \rangle$ if $n > m$, $f_1 \gtrsim g_1$ and $f_2 \sqsupseteq g_2$, and
- $\langle (n, f_1), f_2 \rangle \succcurlyeq \langle (m, g_1), g_2 \rangle$ if $n \geq m$, $f_1 \gtrsim g_1$ and $f_2 \sqsupseteq g_2$.

We say a function f is a *cost (size) function* whenever $f \in \mathcal{F}^{\mathsf{c}}_\sigma$ ($f \in \mathcal{S}_\sigma$), for some type σ.

▶ **Lemma 15.** *For any type σ, $(\mathcal{C}_\sigma, >, \gtrsim)$ is well-founded and $(\mathcal{S}_\sigma, \sqsupseteq)$ is quasi-ordered with a minimum. Therefore, $(\!|\sigma|\!)$ is a cost–size product.*

Proof. When σ is a base type, $\mathcal{C}_\sigma = \mathbb{N} \times \texttt{unit} \cong \mathbb{N}$ and $\mathcal{S}_\sigma = \mathcal{J}_{\mathbb{B}}(\sigma)$, so the statement is trivially true. Let $\sigma = \tau \Rightarrow \rho$, then by induction hypothesis $\mathcal{S}_\tau$ and $\mathcal{S}_\rho$ are quasi-ordered. Quasi-ordering of $(\mathcal{S}_{\tau \Rightarrow \rho}, \sqsupseteq)$ follows from the induced point-wise comparison. A minimum for this size set is the function $\boldsymbol{\lambda} x.\bot$. Well-foundedness of $(\mathcal{C}_\sigma, \succ, \succcurlyeq)$ follows from Lemma 11 by showing that $\mathcal{F}^{\mathsf{c}}_{\tau \Rightarrow \rho}$ is quasi ordered. ◀

To map each term $s : \sigma$ to an element of $(\!|\sigma|\!)$ (Definition 25), we need a notion of application for cost-size tuples. More precisely, assume given a type $\sigma \Rightarrow \tau$ and cost–size tuples $\boldsymbol{f} \in (\!|\sigma \Rightarrow \tau|\!)$ and $\boldsymbol{x} \in (\!|\sigma|\!)$. We define the application of $\boldsymbol{f}$ to $\boldsymbol{x}$, denoted $\boldsymbol{f} \cdot \boldsymbol{x}$, as follows.

▶ **Definition 16.** Let $\sigma \Rightarrow \tau$ be an arrow type, $\boldsymbol{f} = \langle (n, f^{\mathsf{c}}), f^{\mathsf{s}} \rangle \in (\!|\sigma \Rightarrow \tau|\!)$, and $\boldsymbol{x} = \langle (m, x^{\mathsf{c}}), x^{\mathsf{s}} \rangle \in (\!|\sigma|\!)$. The **semantic application** of $\boldsymbol{f}$ to $\boldsymbol{x}$, denoted $\boldsymbol{f} \cdot \boldsymbol{x}$, is defined by:

$$\text{let } f^{\mathsf{c}}(x^{\mathsf{c}}, x^{\mathsf{s}}) = (k, h)\text{; then } \langle (n, f^{\mathsf{c}}), f^{\mathsf{s}} \rangle \cdot \langle (m, x^{\mathsf{c}}), x^{\mathsf{s}} \rangle = \langle (n + m + k, h), f^{\mathsf{s}}(x^{\mathsf{s}}) \rangle$$

We set the semantic application to be left-associative, so $f \cdot g \cdot h$ denotes $(f \cdot g) \cdot h$.

▶ **Example 17.** Let us illustrate semantic application with a concrete example: consider the type $\sigma = (\mathsf{nat} \Rightarrow \mathsf{nat}) \Rightarrow \mathsf{list} \Rightarrow \mathsf{list}$, which is the type of map defined in Example 1. The function map takes as argument a function $F : \mathsf{nat} \Rightarrow \mathsf{nat}$ and list q and applies F to each element of q. This formalizes the cost and size ideas in Examples 5 and 6. Hence, the cost–size interpretation of map is an element $\langle (n, f^{\mathsf{c}}), f^{\mathsf{s}} \rangle$ of $(\!|\sigma|\!)$. Its cost component (n, f^{c}) is in $\mathcal{C}_\sigma = \mathbb{N} \times \mathcal{F}^{\mathsf{c}}_\sigma$ which is composed of a numeric and functional component. The numeric component n carries the cost of partial application. Meanwhile, the functional component in $\mathcal{F}^{\mathsf{c}}_\sigma$ is parametrized by functional arguments carrying the cost and size information of F. Indeed, Definition 14 gives us $f^{\mathsf{c}} : \mathcal{F}^{\mathsf{c}}_{\mathsf{nat} \Rightarrow \mathsf{nat}} \times \mathcal{S}_{\mathsf{nat} \Rightarrow \mathsf{nat}} \Longrightarrow \mathcal{C}_{\mathsf{list} \Rightarrow \mathsf{list}}$, which can be written explicitly as:

$$\overbrace{\left(\underbrace{(\mathtt{unit} \times \mathbb{N} \Longrightarrow \mathbb{N} \times \mathtt{unit})}_{\text{cost of } F} \times \underbrace{(\mathcal{S}_{\mathsf{nat}} \Longrightarrow \mathcal{S}_{\mathsf{nat}})}_{\text{size of } F}\right) \Longrightarrow \left(\mathbb{N} \times \left[\underbrace{\mathtt{unit}}_{q^{\mathsf{c}}} \times \underbrace{\mathcal{S}_{\mathsf{list}}}_{q^{\mathsf{s}}} \Longrightarrow \mathbb{N} \times \mathtt{unit}\right]\right)}^{\text{the functional cost of } \mathsf{map}}$$

The set for the size function is somewhat simpler with $f^{\mathsf{s}} : (\mathcal{S}_{\mathsf{nat}} \Longrightarrow \mathcal{S}_{\mathsf{nat}}) \Longrightarrow \mathcal{S}_{\mathsf{list}} \Longrightarrow \mathcal{S}_{\mathsf{list}}$.

Therefore, we apply $\boldsymbol{f}$ to a cost-size tuple $\boldsymbol{x}$ of the form $\langle (m, x^{\mathsf{c}}), x^{\mathsf{s}} \rangle$ where x^{c} is the cost of computing F (so an element of $\mathcal{F}^{\mathsf{c}}_{\mathsf{nat} \Rightarrow \mathsf{nat}}$) and x^{s} is the size of F, so an element of $\mathcal{S}_{\mathsf{nat} \Rightarrow \mathsf{nat}}$. We proceed by applying the respective functions so $f^{\mathsf{c}}(x^{\mathsf{c}}, x^{\mathsf{s}}) = (k, h)$ belongs to $\mathcal{C}_{\mathsf{list} \Rightarrow \mathsf{list}}$ and $f^{\mathsf{s}}(x^{\mathsf{s}})$ is in $\mathcal{S}_{\mathsf{list} \Rightarrow \mathsf{list}}$. We put everything together and add the numeric components to obtain: $\boldsymbol{f} \cdot \boldsymbol{x} = \langle (n + m + k, h), f^{\mathsf{s}}(x^{\mathsf{s}}) \rangle$. Notice that this gives us a new cost–size tuple with the cost component in $\mathbb{N} \times (\mathcal{C}_{\mathsf{list}} \Longrightarrow \mathcal{C}_{\mathsf{list}})$ and size component in $\mathcal{S}_{\mathsf{list}} \Longrightarrow \mathcal{S}_{\mathsf{list}}$, which is a tuple in $(\!|\mathsf{list} \Rightarrow \mathsf{list}|\!)$.

Observe that our intention with Definition 16 is that the semantic application conforms with a form of "application typing rule". A straightforward analysis on Definition 16 shows that this is indeed the case. This is summarized in the lemma below.

▶ **Lemma 18.** *If $\boldsymbol{f} \in (\!|\sigma \Rightarrow \tau|\!)$ and $\boldsymbol{x} \in (\!|\sigma|\!)$, then $\boldsymbol{f} \cdot \boldsymbol{x}$ belongs to $(\!|\tau|\!)$.*

Definition 14 gives us a family of cost–size sets $\mathcal{T} = \{(\!|\sigma|\!)\}_{\sigma \in \mathbb{T}_{\mathbb{B}}}$ indexed by $\mathbb{T}_{\mathbb{B}}$, and combined with Definition 16 we get a family of *application operators*

$$(\mathcal{T}, \cdot) = \left(\{(\!|\sigma|\!)\}_{\sigma \in \mathbb{T}_{\mathbb{B}}}, \{\cdot_{\sigma,\tau}\}_{\sigma,\tau \in \mathbb{T}_{\mathbb{B}}}\right), \text{ with } \cdot_{\sigma,\tau} : (\!|\sigma \Rightarrow \tau|\!) \times (\!|\sigma|\!) \longrightarrow (\!|\tau|\!)$$

We call the pair $(\mathcal{T}, \cdot)$ the cost–size type structure generated by the interpretation key $\mathcal{J}_{\mathbb{B}}$. Indeed, in the next Lemma we show that such structure preserves the orderings $\succ$ and $\succcurlyeq$ on cost–size tuples.

▶ **Lemma 19.** *The application operator is strongly monotonic in both arguments.*

Proof. We need to prove the following: (i) if $\boldsymbol{f} \succ \boldsymbol{g}$ and $\boldsymbol{x} \succcurlyeq \boldsymbol{y}$, then $\boldsymbol{f} \cdot \boldsymbol{x} \succ \boldsymbol{g} \cdot \boldsymbol{y}$; (ii) if $\boldsymbol{f} \succcurlyeq \boldsymbol{g}$ and $\boldsymbol{x} \succ \boldsymbol{y}$, then $\boldsymbol{f} \cdot \boldsymbol{x} \succ \boldsymbol{g} \cdot \boldsymbol{y}$; (iii) if $\boldsymbol{f} \succcurlyeq \boldsymbol{g}$ and $\boldsymbol{x} \succcurlyeq \boldsymbol{y}$, then $\boldsymbol{f} \cdot \boldsymbol{x} \succcurlyeq \boldsymbol{g} \cdot \boldsymbol{y}$. Consider cost–size tuples $\boldsymbol{f}, \boldsymbol{g} \in (\!|\sigma \Rightarrow \tau|\!)$ and $\boldsymbol{x}, \boldsymbol{y} \in (\!|\sigma|\!)$. Let $\boldsymbol{f} = \langle (n, f^{\mathsf{c}}), f^{\mathsf{s}} \rangle$, $\boldsymbol{g} = \langle (m, g^{\mathsf{c}}), g^{\mathsf{s}} \rangle$, $\boldsymbol{x} = \langle (j, x^{\mathsf{c}}), x^{\mathsf{s}} \rangle$, and $\boldsymbol{y} = \langle (j', y^{\mathsf{c}}), y^{\mathsf{s}} \rangle$. We proceed to show (i) and observe that (ii) and (iii) follow similar reasoning. Indeed, if $\boldsymbol{f} \succ \boldsymbol{g}$ and $\boldsymbol{x} \succcurlyeq \boldsymbol{y}$ we have that $n > m$, $f^{\mathsf{c}} \gtrsim g^{\mathsf{c}}$, $f^{\mathsf{s}} \sqsupseteq g^{\mathsf{s}}$, $j \geq j'$, $x^{\mathsf{c}} \gtrsim y^{\mathsf{c}}$, and $x^{\mathsf{s}} \sqsupseteq y^{\mathsf{s}}$. Let $f^{\mathsf{c}}(x^{\mathsf{c}}, x^{\mathsf{s}}) = (k, h)$ and $g^{\mathsf{c}}(y^{\mathsf{c}}, y^{\mathsf{s}}) = (k', h')$, we get:

$$\boldsymbol{f} \cdot \boldsymbol{x} = \langle (n + j + k, h), f^{\mathsf{s}}(x^{\mathsf{s}}) \rangle \succ \langle (m + j' + k', h'), g^{\mathsf{s}}(y^{\mathsf{s}}) \rangle = \boldsymbol{g} \cdot \boldsymbol{y}$$ ◀

▶ Remark 20. Notice that the type structure $(\mathcal{T}, \cdot)$ is nonstandard. Indeed, the intended standard semantics given to arrow types is usually a functional space [5, Chapter 3]. So inhabitants of functional types are interpreted as functions. Since our intention with defining *cost–size type structures* as above is to capture the complexity-wise behavior of functions (defined by rewriting rules) and a cost component associated with the computational environment, this non-standardness is expected. In the next sections we show that even though our interpretations do not give rise to a standard semantic of simple types, we can still prove classical lemmata for substitution and compatibility.

▶ **Example 21.** In Examples 1 and 2 we have two examples of base types: nat and list. Values of type nat are built using the constructors $\mathsf{0} : \mathsf{nat}$ and $\mathsf{s} : \mathsf{nat} \Rightarrow \mathsf{nat}$. Similarly, for list we have $\mathsf{nil} : \mathsf{list}$ and $\mathsf{cons} : \mathsf{nat} \Rightarrow \mathsf{list} \Rightarrow \mathsf{list}$.

Let us give a cost–size type structure over $\mathbb{N}$ (Example 13) for $\mathbb{B} = \{\mathsf{nat}, \mathsf{list}\}$. Essentially, we need to choose the numbers $K(\mathsf{nat}), K(\mathsf{list})$ associated with nat and list, respectively. To do so we take the intentional size semantic of $\mathsf{nat}, \mathsf{list}$ into account. Let us set $K(\mathsf{nat}) = 1$ and $K(\mathsf{list}) = 2$. This exactly gives the size sets we used in Section 3, and allows us to use "number of symbols" as a notion of size in a unary representation of numbers, and (*length*, *maximum element size*) as a size notion for lists. Intuitively, since a list is a container-like data structure we want to be able to simultaneously give upper bounds to "the size of the container" (which is *length* for lists) and "the size of its elements". This choice of $\mathcal{J}_{\mathbb{B}/\mathbb{N}}$ affects the shape of interpretations for symbols in Σ, as we will see in Example 23.

Even though we have manually chosen the size tuples for $\mathcal{J}_{\mathbb{B}/\mathbb{N}}$ above, an automated procedure can still be devised to determine the number $K(\iota)$, for $\iota \in \mathbb{B}$. A description of such a procedure can be found in [22].

5 Cost–Size Semantics for Terms

In the previous section, we established a cost–size semantics for the simple types in $\mathbb{T}_{\mathbb{B}}$. Our goal in this section is to interpret terms as elements of those sets.

An interpretation of a signature $\mathbb{F} = (\mathbb{B}, \Sigma, \mathtt{ar})$ interprets the base types in $\mathbb{B}$ and each $\mathsf{f} \in \Sigma$ of arity $\mathtt{ar}(f) = \sigma$ as an element of $\llparenthesis \sigma \rrparenthesis$ which is constructed by Definition 14. This is formally stated in the definition below.

▶ **Definition 22.** A **cost–size tuple interpretation** $\mathcal{F}$ for a signature $\mathbb{F} = (\mathbb{B}, \Sigma, \mathtt{ar})$ consists of a pair of functions $(\mathcal{J}_{\mathbb{B}}, \mathcal{J}_{\Sigma})$ where

- $\mathcal{J}_{\mathbb{B}}$ is a type interpretation key (Definition 12),
- $\mathcal{J}_{\Sigma}$ is an *interpretation of symbols* in Σ which maps each $\mathsf{f} \in \Sigma$ with $\mathtt{ar}(\mathsf{f}) = \sigma$ to a cost–size tuple in $\llparenthesis \sigma \rrparenthesis$, where $\llparenthesis \sigma \rrparenthesis$ is built using $\mathcal{J}_{\mathbb{B}}$ in Definition 14.

In what follows we slightly abuse notation by writing $\mathcal{J}_{\mathsf{f}}$ for $\mathcal{J}_{\Sigma}(\mathsf{f})$ and just $\mathcal{J}$ for $\mathcal{J}_{\Sigma}$.

▶ **Example 23.** As a first example of interpretation, let us interpret the data signature from Example 21. Recall that $\mathsf{0} : \mathsf{nat},\ \mathsf{s} : \mathsf{nat} \Rightarrow \mathsf{nat}$ are the constructors for nat and $K(\mathsf{nat}) = 1$.

$$\mathcal{J}_{\mathsf{0}} = \left\langle\ (0, \mathtt{u})\ , 1 \right\rangle \qquad\qquad \mathcal{J}_{\mathsf{s}} = \left\langle\ (0, \boldsymbol{\lambda} x.(0, \mathtt{u}))\ , \boldsymbol{\lambda} x.x + 1 \right\rangle$$

The highlighted cost components for the constructors are filled with zeroes. That is because in the rewriting cost model data values do not fire rewriting sequences. In the language of Section 3: the *cost number* for $\mathsf{0}$ is 0, (because it is a value), the *cost function* is $\mathtt{u}$ (because it has base type), and *size component* is 1 (since we chose a notion of size for terms of type nat to mean "number of symbols"). The cost number for s is 0, the cost function is the constant function mapping to 0, and the size component is the function $\boldsymbol{\lambda} x.x + 1$ in $\mathcal{S}_{\mathsf{nat} \Rightarrow \mathsf{nat}}$. We interpret the constructors for list, i.e., nil and cons, following the same principle, with $K(\mathsf{list}) = 2$. We write a size tuple q in $\mathcal{S}_{\mathsf{list}}$ as $(q_{\mathsf{l}}, q_{\mathsf{m}})$ since the first component is to mean the length of the list and the second a bound on the size of its elements.

$$\mathcal{J}_{\mathsf{nil}} = \left\langle\ (0, \mathtt{u})\ , (0, 0) \right\rangle \quad \mathcal{J}_{\mathsf{cons}} = \left\langle\ (0, \boldsymbol{\lambda} x.(0, \boldsymbol{\lambda} q.(0, \mathtt{u})))\ , \boldsymbol{\lambda} xq.(q_{\mathsf{l}} + 1, \max(x, q_{\mathsf{m}})) \right\rangle$$

The highlighted cost components are filled with zeroes for lists as well. Size components are interpreted as expected, and exactly following Example 5.

The next step is to extend the interpretation of a signature $\mathbb{F}$ to the set of terms. But first, we define *valuation functions* to interpret the variables in $x : \sigma$ as elements of $\llparenthesis \sigma \rrparenthesis$.

▶ **Definition 24.** A **cost–size valuation** α is a function that maps each $x : \sigma$ to a cost-size tuple in $(\!|\sigma|\!)$ such that:

- $\alpha(x) = \langle (0, \mathtt{u}), x^{\mathsf{s}} \rangle$, for all $x \in \mathbb{X}$ of base type, and
- $\alpha(F) = \langle (0, F^{\mathsf{c}}), F^{\mathsf{s}} \rangle$ when $F :: \sigma \Rightarrow \tau$.

Notice that, in this definition, the cost component of $\alpha(x)$ has the form $(0, \mathtt{u})$, if $x : \iota$. This interpretation is motivated by Definition 4, where a matching substitution γ (i.e., a substitution such that $\ell\gamma \rightarrow_v r\gamma$) must map each $x : \iota$ to a value of base type. Those can only have the form $\mathsf{c}(v_1, \dots, v_m)$ with $\mathsf{c} \in \Sigma^{\mathsf{con}}$. Variables of arrow type still have a cost number 0; however, they can be instantiated to values that carry *indirect* computational content: a partial application or abstraction. For instance, a variable of type $F : \mathsf{nat} \Rightarrow \mathsf{nat}$ can be instantiated with $\mathsf{add}\,0$, which is a value that produces a cost as soon as it is applied to the next argument. We use the notation $F^{\mathsf{c}}/F^{\mathsf{s}}$ to denote the cost/size component of $\alpha(F)$.

▶ **Definition 25.** Assume given a signature $\mathbb{F} = (\mathbb{B}, \Sigma, \mathtt{ar})$ and its cost–size tuple interpretation $\mathcal{F} = (\mathcal{J}_{\mathbb{B}}, \mathcal{J})$ together with a valuation α. The **term interpretation** $[\![s]\!]^{\mathcal{J}}_{\alpha}$ of s under $\mathcal{J}$ and α is defined by induction on the structure of s as follows:

$$[\![x]\!]^{\mathcal{J}}_{\alpha} \;=\; \alpha(x) \qquad [\![\mathsf{f}]\!]^{\mathcal{J}}_{\alpha} \;=\; \mathcal{J}_{\mathsf{f}} \qquad [\![s\,t]\!]^{\mathcal{J}}_{\alpha} \;=\; [\![s]\!]^{\mathcal{J}}_{\alpha} \cdot [\![t]\!]^{\mathcal{J}}_{\alpha}$$
$$[\![\lambda x.\, s]\!]^{\mathcal{J}}_{\alpha} = \Big\langle \Big(0, \boldsymbol{\lambda} d.(1 + \pi_{11}([\![s]\!]^{\mathcal{J}}_{[x:=d]\alpha}), \pi_{12}([\![s]\!]^{\mathcal{J}}_{[x:=d]\alpha}))\Big), \boldsymbol{\lambda} d^{\mathsf{s}}.\pi_2([\![s]\!]^{\mathcal{J}}_{[x:=(\underline{0},d)]\alpha})\Big\rangle,$$

where π_i is the projection on the ith-component and π_{ij} is the composition $\pi_j \circ \pi_i$, and $\underline{0}$ is a cost function of the form $\boldsymbol{\lambda} x_1.(0, \boldsymbol{\lambda} x_2 \dots (0, \mathtt{u}) \dots)$. If $d = (d^c, d^s)$, the notation $[x := d]\alpha$ denotes the valuation that maps x to $\langle (0, d^c), d^s \rangle$ and every other variable y to $\alpha(y)$.

We write $[\![s]\!]$ for $[\![s]\!]^{\mathcal{J}}_{\alpha}$ whenever α and $\mathcal{J}$ are universally quantified or clear from the context.

The interpretation for abstractions may seem baroque, but can be understood as follows: an abstraction is a value, so its cost number is 0. The cost of applying that abstraction on a value v is 1 plus the cost number for $s[x := v]$ – which is obtained by evaluating $[\![s]\!]^{\mathcal{J}}_{[x:=d]\alpha}$ if d is the cost function/size pair for v. The cost *function* of this application is exactly the cost function of $s[x := v]$. The *size* of an abstraction $\lambda x.s$ is exactly the function that takes a size and maps it to the size interpretation of s where x is mapped to that size. Technically, to obtain the size component of $[\![s]\!]^{\mathcal{J}}_{[x:=d]\alpha}$ we also need a cost component, but by definition, this component does not play a role, so we can safely choose an arbitrary pair $\underline{0}$ in the right set.

▶ **Example 26.** We continue with Example 23 by interpreting ground constructor terms fully. A ground constructor term d of type nat is of the form $\mathsf{s}\,(\mathsf{s} \dots (\mathsf{s}\,0) \dots)$ where the number $n \in \mathbb{N}$ is represented by n successive applications of s to 0. Let us write n as shorthand notation for such terms. Similarly, for ground constructor terms of type list, we write $[\mathsf{n}_1; \dots; \mathsf{n}_k]$ for the term $\mathsf{cons}\,\mathsf{n}_1 \dots (\mathsf{cons}\,\mathsf{n}_k\,\mathsf{nil})$. The empty list constructor nil is written as $[]$ in this notation. Hence, the cost–size interpretation of $3 : \mathsf{nat}$ is given by:

$$[\![3]\!] = [\![\mathsf{s}\,(\mathsf{s}\,(\mathsf{s}\,0))]\!] = [\![\mathsf{s}]\!] \cdot ([\![\mathsf{s}]\!] \cdot ([\![\mathsf{s}]\!] \cdot [\![0]\!])) = \Big\langle \; (0, \mathtt{u}) \;\; , 4 \Big\rangle .$$

Consider, for instance, the list $[1; 7; 9]$. Its cost–size interpretation is given by:

$$[\![[1; 7; 9]]\!] = [\![\mathsf{cons}\,1\,(\mathsf{cons}\,7\,(\mathsf{cons}\,9\,\mathsf{nil}))]\!] \;\; = \Big\langle \; (0, \mathtt{u}) \;\; , (3, 10) \Big\rangle .$$

The important information we can extract from such interpretations is their size component. Indeed, $[\![3]\!]^{\mathsf{s}} = 4$ counts the number of constructor symbols in the term representation 3 and $[\![[1; 7; 9]]\!]^{\mathsf{s}} = (3, 10)$ gives us the length and an upper bound on the size of each element in $[1; 7; 9]$. The size interpretation for the constructors of nat and list correctly capture our notion of “size” given in Example 21.

The next Lemma expresses the soundness of term interpretation, that is, the interpretation of terms preserves the type structure:

▶ **Lemma 27** (Type Soundness). *If $s : \sigma$ then $[\![s]\!] \in (\!|\sigma|\!)$.*

Proof. The proof is by induction on the structure of s. The base cases follow directly from Definitions 22 and 24. We use Lemmas 18 and 19 in the application case. The abstraction case follows from the induction hypothesis and weak monotonicity of π_i. ◀

Up to now, we have given cost–size semantics for types and terms. Observe that Definition 22 only requires that we interpret function symbols as cost–size tuples in the correct domain. For instance, we might interpret all function components as constant functions. This is a valid, but not so useful, interpretation of terms. So we move on to the next component of our interpretation framework: we want to interpret terms in such a way that $[\![s]\!] \succ [\![t]\!]$ whenever $s \to t$, for any pair of terms s, t.

▶ **Definition 28.** Consider a signature $\mathbb{F} = (\mathbb{B}, \Sigma, \mathtt{ar})$. A **cost–size call-by-value termination model** for a term rewriting system $(\mathbb{F}, \mathbb{R})$ consists of the following ingredients:

- an interpretation key $\mathcal{J}_{\mathbb{B}}$ (Definition 12), together with
- a cost–size interpretation $(\mathcal{J}_{\mathbb{B}}, \mathcal{J}_{\Sigma})$ (Definition 22),

such that the following **compatibility conditions** hold:

- for all value substitutions γ and all terms s and t, $[\![s\gamma]\!] \succ [\![t\gamma]\!]$ whenever $[\![s]\!] \succ [\![t]\!]$;
- for every term s and value v, $[\![(\lambda x.\, s)\, v]\!] \succ [\![s[x := v]]\!]$;
- for all terms s and t,
  - $[\![s\, t]\!] \succ [\![s'\, t]\!]$ whenever $[\![s]\!] \succ [\![s']\!]$, and $[\![s\, t]\!] \succ [\![s\, t']\!]$ whenever $[\![t]\!] \succ [\![t']\!]$;
- for all rules $\ell \to r \in \mathbb{R}$, we have $[\![\ell]\!] \succ [\![r]\!]$.

Roughly speaking, a call-by-value termination model is an interpretation of types and terms that is compatible with each rule in $\mathbb{R}$, the call-by-value beta rule and the formation of terms, and which is closed under value substitutions. By a straightforward induction on the reduction $s \to_v t$, we can establish the following result.

▶ **Theorem 29.** *Let $(\mathbb{F}, \mathbb{R})$ be a TRS. If we have a termination model of $(\mathbb{F}, \mathbb{R})$, then the higher-order call-by-value rewriting relation $\to_v$ is strongly normalizing.*

Hence, termination models collect sufficient conditions for strong normalization. The lemmata below are to show that cost–size interpretations satisfy some of the compatibility conditions for termination models. Let us first prove closure under substitutions.

▶ **Definition 30.** Given a substitution γ and valuation α, we define the **γ-extension of α** as the valuation defined by $\alpha^{\gamma} = [\![\cdot]\!]_{\alpha}^{\mathcal{J}} \circ \gamma$.

▶ **Lemma 31.** *If $x \notin \mathtt{fv}(s)$ then $[\![s]\!]_{[x:=d]\alpha} = [\![s]\!]_{\alpha}$. Consequently, if x is not free in $y\gamma$ for any variable y, then $([x := d]\alpha)^{\gamma} = [x := d]\alpha^{\gamma}$.*

▶ **Lemma 32** (Substitution Lemma). *For any value substitution γ and valuation α, we have that $[\![s\gamma]\!]_{\alpha} = [\![s]\!]_{\alpha^{\gamma}}$.*

Proof. Let us work out the abstraction case $s = \lambda x.\, t$. Since we assume that the application of substitution is capture-avoiding, we can assume that x does not occur free in any term in the range of γ. Hence,

$$\begin{aligned}
[\![\lambda x.\,(t\gamma)]\!]_{\alpha} &= \Big\langle \Big(0, \boldsymbol{\lambda} d.(1 + \pi_{11}([\![t\gamma]\!]^{\mathcal{J}}_{[x:=d]\alpha}), \pi_{12}([\![t\gamma]\!]^{\mathcal{J}}_{[x:=d]\alpha}))\Big)\,, \boldsymbol{\lambda} d^{\mathsf{s}}.\pi_2([\![t\gamma]\!]^{\mathcal{J}}_{[x:=(\underline{0},d)]\alpha})\Big\rangle \\
&\overset{IH}{=} \Big\langle \Big(0, \boldsymbol{\lambda} d.(1 + \pi_{11}([\![t]\!]^{\mathcal{J}}_{([x:=d]\alpha)^{\gamma}}), \pi_{12}([\![t]\!]^{\mathcal{J}}_{([x:=d]\alpha)^{\gamma}}))\Big), \boldsymbol{\lambda} d^{\mathsf{s}}.\pi_2([\![t]\!]^{\mathcal{J}}_{([x:=(\underline{0},d)]\alpha)^{\gamma}})\Big\rangle \\
&= \Big\langle \Big(0, \boldsymbol{\lambda} d.(1 + \pi_{11}([\![t]\!]^{\mathcal{J}}_{[x:=d]\alpha^{\gamma}}), \pi_{12}([\![t]\!]^{\mathcal{J}}_{[x:=d]\alpha^{\gamma}}))\Big)\,, \boldsymbol{\lambda} d^{\mathsf{s}}.\pi_2([\![t]\!]^{\mathcal{J}}_{[x:=(\underline{0},d)]\alpha^{\gamma}})\Big\rangle \\
&= [\![\lambda x.\, t]\!]_{\alpha^{\gamma}}.
\end{aligned}$$

◀

As a consequence of the substitution lemma, if $[\![s]\!]^{\mathcal{J}}_{\alpha} \succ [\![t]\!]^{\mathcal{J}}_{\alpha}$ for all α, then $[\![s\gamma]\!]^{\mathcal{J}}_{\alpha} \succ [\![t\gamma]\!]^{\mathcal{J}}_{\alpha}$ for all α. Consequently, the first compatibility condition is valid for any interpretation. The second compatibility requirement is for β reductions.

▶ **Lemma 33.** *The call-by-value beta rule scheme $(\lambda x.\, s)\, v \to_v s[x := v]$ is strictly decreasing for any cost–size interpretation.*

Proof. The proof reduces to checking $[\![(\lambda x.\, s)\, v]\!] \succ [\![s[x := v]]\!]$. Let $[\![v]\!] = \langle (0, v^{\mathsf{c}}), v^{\mathsf{s}} \rangle$, and denote V for the pair $(v^{\mathsf{c}}, v^{\mathsf{s}})$. Then we have the following:

$$\begin{aligned}
[\![(\lambda x.\, s)\, v]\!] &= [\![\lambda x.\, s]\!] \cdot [\![v]\!] \\
&= \Big\langle \Big(0, \boldsymbol{\lambda} d.(1 + \pi_{11}([\![s]\!]^{\mathcal{J}}_{[x:=d]\alpha}), \pi_{12}([\![s]\!]^{\mathcal{J}}_{[x:=d]\alpha}))\Big)\,, \boldsymbol{\lambda} d^{\mathsf{s}}.\pi_2([\![s]\!]^{\mathcal{J}}_{[x:=(\underline{0},d^{\mathsf{s}})]\alpha})\Big\rangle \cdot [\![v]\!] \\
&= \Big\langle \Big(0 + 0 + 1 + \pi_{11}([\![s]\!]^{\mathcal{J}}_{[x:=V]\alpha}), \pi_{12}([\![s]\!]^{\mathcal{J}}_{[x:=V]\alpha})\Big)\,, \pi_2([\![s]\!]^{\mathcal{J}}_{[x:=\langle\underline{0},v^{\mathsf{s}}\rangle]\alpha})\Big\rangle \\
&\succ \Big\langle \Big(\pi_{11}([\![s]\!]^{\mathcal{J}}_{[x:=V]\alpha}), \pi_{12}([\![s]\!]^{\mathcal{J}}_{[x:=V]\alpha})\Big)\,, \pi_2([\![s]\!]^{\mathcal{J}}_{[x:=V]\alpha})\Big\rangle \\
&= [\![s[x := v]]\!]_{\alpha}.
\end{aligned}$$

In the second-to-last step, we use that the size component of $[\![s]\!]^{\mathcal{J}}_{\alpha}$ does not regard any cost component in α, so $\pi_2([\![s]\!]^{\mathcal{J}}_{[x:=\langle\underline{0},v^{\mathsf{s}})\rangle]\alpha}) = \pi_2([\![s]\!]^{\mathcal{J}}_{[x:=V]\alpha})$. In the last step, we use the substitution lemma. ◀

Compatibility over applicative terms is a consequence of Lemma 19. Notice that the results above do not depend on a particular interpretation. Hence, to establish a termination model for a TRS, only the last compatibility condition remains to be checked, i.e., $[\![\ell]\!] \succ [\![r]\!]$ for all rules $\ell \to r$ in $\mathbb{R}$. We collect this fact below, which is a consequence of Theorem 29 and the Lemmas above.

▶ **Corollary 34.** *Let $\mathbb{R}$ be a TRS that admits a cost–size interpretation $(\mathcal{J}_{\mathbb{B}}, \mathcal{J}_{\Sigma})$. If $[\![\ell]\!] \succ [\![r]\!]$ for all rules $\ell \to r$ in $\mathbb{R}$, then $\mathbb{R}$ is a termination model, and consequently strongly normalizing.*

Interpretation techniques are usually applied to show full termination [7, 18, 25] or as quasi-orderings for the dependency pair approach [1]. In the next example, we show that cost–size interpretations are weak enough to prove termination of call-by-value systems that do not necessarily terminate under full rewriting.

▶ **Example 35.** Let $\mathsf{a}, \mathsf{b} : \iota$, $\mathsf{g} : \iota \Rightarrow \iota \Rightarrow \iota$, and $\mathsf{f} : \iota \Rightarrow \iota \Rightarrow \iota \Rightarrow \iota$. The rewrite system introduced by Toyama [30] and defined by $\mathbb{R} = \{\mathsf{g}\, x\, y \to x,\ \mathsf{g}\, x\, y \to y,\ \mathsf{f}\, \mathsf{a}\, \mathsf{b}\, z \to \mathsf{f}\, z\, z\, z\}$ was given to show that termination is not modular for disjoint unions of TRSs. Indeed, it admits the infinite rewriting sequence $\mathsf{f}\,\mathsf{a}\,\mathsf{b}\,(\mathsf{g}\,\mathsf{a}\,\mathsf{b}) \rightarrow_{\mathbb{R}} \mathsf{f}\,(\mathsf{g}\,\mathsf{a}\,\mathsf{b})\,(\mathsf{g}\,\mathsf{a}\,\mathsf{b})\,(\mathsf{g}\,\mathsf{a}\,\mathsf{b}) \rightarrow^{+}_{\mathbb{R}} \mathsf{f}\,\mathsf{a}\,\mathsf{b}\,(\mathsf{g}\,\mathsf{a}\,\mathsf{b})$, whereas the systems $\mathbb{R}_{\mathsf{g}}$ and $\mathbb{R}_{\mathsf{f}}$ are individually terminating. If we restrict reductions to call-by-value, then the rewrite relation $\rightarrow_v$ induced by $\mathbb{R}$ is terminating.

In order to prove termination of $\mathbb{R}$, we introduce a non-numeric notion of size. Let $\mathcal{J}_{\mathbb{B}}(\iota) = \mathcal{P}(\mathsf{T}(\mathbb{F}, \mathbb{X}))$, i.e., the set of all subsets of $\mathsf{T}(\mathbb{F}, \mathbb{X})$. This set is partially ordered by inclusion, so we define $x \sqsupseteq y$ iff $x \supseteq y$ which is a quasi-order. Consider the following interpretation:

$$\mathcal{J}_{\mathsf{a}} = \Big\langle\ (0, \mathtt{u})\ , \{\mathsf{a}\}\Big\rangle \qquad \mathcal{J}_{\mathsf{g}} = \Big\langle\ (0, \boldsymbol{\lambda} x.(0, \boldsymbol{\lambda} y.(1, \mathtt{u})))\ , \boldsymbol{\lambda} xy.x \cup y\Big\rangle$$
$$\mathcal{J}_{\mathsf{b}} = \Big\langle\ (0, \mathtt{u})\ , \{\mathsf{b}\}\Big\rangle \qquad \mathcal{J}_{\mathsf{f}} = \Big\langle\ (0, \boldsymbol{\lambda} x.(0, \boldsymbol{\lambda} y.(0, \boldsymbol{\lambda} z.(H(x, y), \mathtt{u}))))\ , \boldsymbol{\lambda} xyz.\emptyset\Big\rangle\,,$$

where H is a helper function defined by $H(x, y) = \mathtt{if}\ x^{\mathsf{s}} \sqsupseteq \{\mathsf{a}\} \wedge y^{\mathsf{s}} \sqsupseteq \{\mathsf{b}\}\ \mathtt{then}\ 1\ \mathtt{else}\ 0$. Notice that H is weakly monotonic, and the size tuples for interpretation of values are sets of cardinality ≤ 1. Checking compatibility for this interpretation is straightforward: $[\![\mathsf{g}\,x\,y]\!] = \langle (1, \mathtt{u}), x \cup y\rangle \succ \langle (0, \mathtt{u}), x\rangle = [\![x]\!]$ and $[\![\mathsf{g}\,x\,y]\!] = \langle (1, \mathtt{u}), x \cup y\rangle \succ \langle (0, \mathtt{u}), y\rangle = [\![y]\!]$; and finally $[\![\mathsf{f}\,\mathsf{a}\,\mathsf{b}\,z]\!] = \langle (1, \mathtt{u}), \emptyset\rangle \succ \langle (0, \mathtt{u}), \emptyset\rangle = [\![\mathsf{f}\,z\,z\,z]\!]$, because any instantiation of z is necessarily a value, so it cannot include both a and b.

6 Complexity Analysis of Call-by-Value Rewriting

In the previous section, we showed that cost–size tuples can be used to establish termination of call-by-value rewriting. In this section, we concentrate on a quantitative analysis of such termination proofs. Hence, the goal is not merely to find tuple interpretations that prove termination but also ones that establish "good" upper bounds on the complexity of reducing terms to normal form. To start, we will extend the notion of *derivation height* to our setting:

▶ **Definition 36.** The weak call-by-value **derivation height** of a term s, notation $\mathtt{dh}_{\mathbb{R}}(s)$, is the largest number n such that $s \rightarrow_v s_1 \rightarrow_v \dots \rightarrow_v s_n$.

This notion is defined for all terms when the TRS is terminating. We will simply refer to the weak call-by-value derivation height as "derivation height".

The methodology of weakly monotonic algebras offers a systematic way to derive bounds for the derivation height of a given term:

▶ **Lemma 37.** *If $[\![s]\!] = \langle (n, F^{\mathsf{c}}), F^{\mathsf{s}}\rangle$, then $\mathtt{dh}_{\mathbb{R}}(s) \leq n$.*

Proof. By the lemmas in Section 5 we see that $[\![s]\!] \succ [\![t]\!]$ whenever $s \rightarrow t$. Since this implies $\pi_{11}(s) > \pi_{11}(t)$, the lemma follows. ◀

As an illustration of how this is used, we present the formalized examples of Section 3 and complete the interpretation of Examples 1 and 2.

Let us start with the system $\mathbb{R}_{\mathsf{add}}$ which intuitively defines addition over nat. We will use the type and constructor interpretations as given in Example 23. The rules $\mathsf{add}\,x\,\mathsf{0} \rightarrow \mathsf{0}$ and $\mathsf{add}\,x\,(\mathsf{s}\,y) \rightarrow \mathsf{s}\,(\mathsf{add}\,x\,y)$ suggest the following cost–size interpretation:

$$\mathcal{J}_{\mathsf{add}} = \Big\langle\ (0, \boldsymbol{\lambda} x.(0, \boldsymbol{\lambda} y.(y^{\mathsf{s}}, \mathtt{u})))\ , \boldsymbol{\lambda} xy.x + y\Big\rangle\,.$$

Notice that the (highlighted) cost component of $\mathcal{J}_{\mathsf{add}}$ suggest a linear cost measure for computing with add. We also set the intermediate numeric components in the cost tuple to zero. The reason for this choice is that in a cost tuple $\mathcal{C}_\sigma = \mathbb{N} \times \mathcal{F}^{\mathsf{c}}_\sigma$, the numeric component $\mathbb{N}$ captures the cost of partially applying terms, which is 0 in this case. Using the shorthand notation of Example 7), we could alternatively write $\mathcal{J}_{\mathsf{add}} = \langle (x^{\mathsf{s}}, y^{\mathsf{s}}) \mapsto y^{\mathsf{s}},\ \boldsymbol{\lambda} x^{\mathsf{s}} y^{\mathsf{s}}.x^{\mathsf{s}} + y^{\mathsf{s}}\rangle$.

Now, consider the partially applied term $s = \mathsf{add}\,(\mathsf{add}\,2\,3)$ (of type $\mathsf{nat} \Rightarrow \mathsf{nat}$). Intuitively, the cost of reducing this term to normal form, is the cost of reducing the subterm $\mathsf{add}\,2\,3$ to 5, since the partially applied term $\mathsf{add}\,5$ cannot be reduced. Hence, $\mathtt{dh}_{\mathbb{R}}(s) = 4$. This is also the bound we find through interpretation:

$$\begin{aligned} [\![s]\!] &= [\![\mathsf{add}]\!] \cdot ([\![\mathsf{add}]\!] \cdot [\![2]\!] \cdot [\![3]\!]) \\ &= [\![\mathsf{add}]\!] \cdot \langle (4, \mathtt{u}), 7 \rangle \\ &= \left\langle \ (4, \boldsymbol{\lambda} y.(y^{\mathsf{s}}, \mathtt{u})) \ , \boldsymbol{\lambda} y.7 + y \right\rangle. \end{aligned}$$

While in this case the bound we find is tight, this is not always the case; for instance $[\![\mathsf{add}\ 0\ (\mathsf{add}\ 0\ 0)]\!] = \langle (3, \mathtt{u}), 3 \rangle$, even though $\mathtt{dh}_{\mathbb{R}}(\mathsf{add}\ 0\ (\mathsf{add}\ 0\ 0)) = 2$. We could obtain a tight bound by choosing a different interpretation, but this is also not always possible.

▶ Remark 38. Intuitively, we think of the numeric component of a partially applied term $\mathsf{f}\,s_1 \dots s_n$ that cannot be reduced at the root as the "environment cost" of computing functional arguments to values. This plays an important role in the complexity analysis in our setting. Namely, when interpreting terms this is what allows us to limit interest to value substitutions, since the cost of reducing arguments to values is captured implicitly by the $\cdot$ operator. This assumption consequently allows us to limit the class of cost functions to *weakly* monotonic functions as used in Definition 14, as opposed to the *strongly* monotonic functionals used in the full rewriting setting [21, 29].

In complexity analysis of term rewriting, it is common to consider bounds on the derivation height for terms of a given size. However, it is useful to impose some limitations. Consider for example a TRS consisting *only* of the two add rules. Then, we might construct a term $(\lambda x.\mathsf{add}\,x\,x)\,((\lambda x.\mathsf{add}\,x\,x)\,(\dots(\mathsf{s}\ 0)\dots))$, with n occurrences of $(\lambda x.\mathsf{add}\,x\,x)$. The size of this term is linear in n, but its derivation height is exponential, since each contraction of a λ essentially duplicates the number of s occurrences. Hence, the traditional notion of *derivational complexity* (which maps a natural number n to the largest derivation height a term of size n can have) is arguably not so useful in a setting with λ.

Instead, we will consider the *runtime complexity* of a TRS. Following the definition in [21] for *full* higher-order runtime complexity, we define:

▶ **Definition 39.** A *data constructor* is a constructor with a type $\iota_1 \Rightarrow \dots \Rightarrow \iota_m \Rightarrow \kappa$, with κ and all ι_i base types.

A *data term* is a value of the form $\mathsf{c}\,d_1\ \dots\ d_m$ with $\mathsf{c} : \iota_1 \Rightarrow \dots \Rightarrow \iota_m \Rightarrow \kappa$ a data constructor, and each d_i a data term; that is, it is a value without any higher-order subterm.

A *basic term* is a base-type term of the form $\mathsf{f}\,d_1\ \dots\ d_m$ with $\mathsf{f} \in \Sigma^{\mathtt{def}}$ a defined symbol and all d_i data terms.

The *weak call-by-value runtime complexity* of a TRS is the function $n \mapsto rc(n)$ that maps each natural number n to the largest number h with $\mathtt{dh}_{\mathbb{R}}(s) = h$ for some term s of size n.

Note that for instance lists of functions are not data terms, and therefore not considered as viable inputs in the notion of runtime complexity. As discussed in [21] this arguably makes the notion somewhat first-order, but it can still be used to analyse higher-order programs or modules (so long as they, for instance, have a rule $\mathsf{start}\ x \to r$ where x has base type, and r is allowed to use abstractions, partial application or calls to higher-order functions).

▶ **Example 40.** Let us collect the interpretation for dbl and mult from Example 2.

$$\begin{aligned} \mathcal{J}_{\mathsf{dbl}} &= \left\langle \ (0, \boldsymbol{\lambda} x.(x^{\mathsf{s}}, \mathtt{u})) \ , \boldsymbol{\lambda} x.2x \right\rangle \\ \mathcal{J}_{\mathsf{mult}} &= \left\langle \ (0, \boldsymbol{\lambda} x.(0, \boldsymbol{\lambda} y.2x^{\mathsf{s}}y^{\mathsf{s}}, \mathtt{u})) \ , \boldsymbol{\lambda} xy.xy \right\rangle \end{aligned}$$

In the TRS of Example 2, the only basic terms have the form $\mathsf{add}\, v_1\, v_2$ or $\mathsf{dbl}\, v$ or $\mathsf{mult}\, v_1\, v_2$. Since $[\![\mathsf{s}^n\ \mathsf{0}]\!]^{\mathsf{s}} = n + 1$, Lemma 37 allows us to conclude that $rc(n) < n^2$.

Now, the size bound for data constructors introduced in Example 23 is well-behaved. However, suppose we had defined $\mathcal{J}_{\mathsf{0}} = \langle (0, \mathtt{u}), 1 \rangle$ and $\mathcal{J}_{\mathsf{s}} = \langle (0, \boldsymbol{\lambda} x.(0, \mathtt{u})), \boldsymbol{\lambda} x.2x + 1 \rangle$. In this case, for a data term $\mathsf{n} = \mathsf{s}^n\, \mathsf{0}$, we would have $[\![\mathsf{n}]\!]^{\mathsf{s}} = 2^n + n \geq 2^n$. As a result, we would only be able to derive exponential runtime complexity. Notice that this choice *is* compatible with $\mathbb{R}_{\mathsf{add}}$, and hence proves its termination; however, it induces an exponential overhead on the cost tuple of add, whose actual runtime complexity is linearly bounded as we saw in Example 40. Such a huge overestimation is not desirable in a complexity analysis setting. This behavior suggests an upper bound to the interpretation of data constructors; namely, we seek to bound the constructor's size interpretations *additively.*

Let c be a data constructor of type $\sigma = \iota_1 \Rightarrow \ldots \Rightarrow \iota_m \Rightarrow \kappa$. The size component of $(\!|\sigma|\!)$ is $\mathcal{S}_\sigma = \mathbb{N}^{K(\iota_1)} \Longrightarrow \ldots \Longrightarrow \mathbb{N}^{K(\iota_m)} \Longrightarrow \mathbb{N}^{K(\kappa)}$. The size tuple $\mathcal{J}^{\mathsf{s}}_{\mathsf{c}}$ when fully applied can be written in terms of its functional components. Hence, $\mathcal{J}^{\mathsf{s}}_{\mathsf{c}}(x_1, \ldots, x_m) = \left\langle f^{\mathsf{s}}_1(x_1, \ldots, x_m), \ldots, f^{\mathsf{s}}_{K(\kappa)}(x_1, \ldots, x_m) \right\rangle$.

▶ **Definition 41.** If $\mathsf{c} : \sigma$ is a data constructor as above, we say $\mathcal{J}^{\mathsf{s}}_{\mathsf{c}}$ is **additive** if there is a constant $a \in \mathbb{N}$ such that $\sum_{l=1}^{K(\kappa)} f^{\mathsf{s}}_l(x_1, \ldots, x_m) \leq a + \sum_{i=1}^{m} \sum_{j=1}^{K(\iota_i)} x_{ij}$.

It is easy to show that size components for nat and list in Example 23 are additive.

If data constructors are additive, and there are only finitely many of them, then there exists a constant a such that, for every data term d of size n: $[\![d]\!]^{\mathsf{s}} \leq an$. Hence, for instance the following result from [21] also extends to our setting:

▶ **Lemma 42** (From [21]; Corollary 33). *Let $\mathbb{R}$ be a TRS. If all interpretations for data constructors are additive and the interpretations for all defined symbols are polynomially bounded, then the weak call-by-value runtime complexity of $\mathbb{R}$ is polynomially bounded.*

This result provides us with a systematic approach to establishing bounds to the runtime complexity of weak call-by-value systems. The difficulty now lies in developing techniques to find suitable interpretation shapes. For instance, a first example of a higher-order function over lists is that of map. We studied the structure of its cost–size tuples in Example 17 to illustrate semantical application. We give a concrete cost–size interpretation for map below:

$$\mathcal{J}_{\mathsf{map}} = \left\langle\ (0, \boldsymbol{\lambda} F.(0, \boldsymbol{\lambda} q.(q_{\mathsf{l}} + F^{\mathsf{c}}(\mathtt{u}, q_{\mathsf{m}})q_{\mathsf{l}} + 1, \mathtt{u})))\ , \boldsymbol{\lambda} F q.(q_{\mathsf{l}}, F(q_{\mathsf{m}}))\right\rangle,$$

The highlighted cost component accounts for q_{l} possible β steps, the cost of applying the higher-order argument F over the list q is bounded by $F^{\mathsf{c}}(\mathtt{u}, q_{\mathsf{m}})q_{\mathsf{l}}$ since F^{c} is assumed to be weakly monotonic, and the unitary component is for dealing with the empty list case.

Finding such interpretations for higher-order systems can become quite challenging. In the example below we collect basic weakly monotonic combinators in order to generate more complex cost/size interpretations.

▶ **Example 43.** We list the following weakly monotonic combinators. Here, sets X, Y, Z are used generically to denote cost/size sets:

- for any X and $a \in Y$, there is a constant functional $\boldsymbol{\lambda} x.a$ in $X \Longrightarrow Y$;
- for $f : X \Longrightarrow Y$ and $g : Y \Longrightarrow Z$, we write $g \circ f : X \Longrightarrow Z$ as the composition of f and g.
- the projection function on the ith coordinate, $\pi_i : X_1 \times \cdots \times X_k \Longrightarrow X_i$;
- given $f : X \Longrightarrow Y$ and $g : X \Longrightarrow Z$, we have a function $\langle f, g\rangle : X \Longrightarrow Y \times Z$ which is defined by $\langle f, g\rangle(x) = \langle f(x), g(x)\rangle$;

- given $f : Y \times X \Longrightarrow Z$, we get a function $\boldsymbol{\lambda}_f : X \Longrightarrow (Y \Longrightarrow Z)$. For each $x \in X$ and $y \in Y$, we define $(\boldsymbol{\lambda}_f(x))(y) = f(y, x)$;
- given $f : X \Longrightarrow (Y \Longrightarrow Z)$ and $g : X \Longrightarrow Y$, we obtain $f \,.\, g : X \Longrightarrow Z$, which is defined as $(f \,.\, g)(x) = f(x)(g(x))$;
- given $f : X \Longrightarrow Y$ and $x \in X$, we have an element application functional with domain $\boldsymbol{app}_x : (X \Longrightarrow Y) \Longrightarrow Y$ which sends f to $f(x)$, i.e., $\boldsymbol{app}_x(f) = f(x)$.

Notice that we can use the combinators above with the usual monotonic functionals and operators over $\mathbb{N}$ to produce new monotonic functionals and pointwise operators over sets $X \Longrightarrow Y$. For instance, we can utilize $+, *, \lfloor\cdot\rfloor, \max, \log(\lfloor\cdot\rfloor)$, and so forth.

These basic weakly monotonic functions provide the building blocks for constructing cost–size interpretations.

▶ **Example 44.** The higher-order functions in Example 1 admit the following interpretations:

$$\begin{aligned}
\mathcal{J}_{\mathsf{app}} &= \Big\langle \; (0, \boldsymbol{\lambda} F.(2, \boldsymbol{\lambda} x.(F^{\mathsf{c}}(\mathtt{u}, x^{\mathsf{s}}), \mathtt{u}))) \;, \boldsymbol{\lambda} F x.F(x) \Big\rangle \\
\mathcal{J}_{\mathsf{comp}} &= \Big\langle \; (0, \boldsymbol{\lambda} F.(0, \boldsymbol{\lambda} G.(2, \boldsymbol{\lambda} x.(F^{\mathsf{c}}(\mathtt{u}, G^{\mathsf{s}}(x^{\mathsf{s}})) + G^{\mathsf{c}}(\mathtt{u}, x^{\mathsf{s}}), \mathtt{u})))) \;, \boldsymbol{\lambda} F G x.F(G(x)) \Big\rangle \\
\mathcal{J}_{\mathsf{rec}} &= \Big\langle \; (0, \boldsymbol{\lambda} x.(0, \boldsymbol{\lambda} y.(0, \boldsymbol{\lambda} F.(x^{\mathsf{s}} + H^{\mathsf{c}}(x, y, F), \mathtt{u})))) \;, \boldsymbol{\lambda} x y F.H^{\mathsf{s}}(x, y, F) \Big\rangle
\end{aligned}$$

In the cost component for $\mathcal{J}_{\mathsf{rec}}$, the term x^{s} computes the total number of rewriting steps using the rec symbol. Meanwhile, H^{c} is an auxiliary symbol computing the total cost of recursively applying the higher-order argument F. It can be defined as follows

$$H^{\mathsf{c}}(x, y, F) = \sum_{i=1}^{x^{\mathsf{s}}-1} \pi_1(F^{\mathsf{c}}((\mathtt{u}, i), (\mathtt{u}, H^{\mathsf{s}}(i, y^{\mathsf{s}}, F^{\mathsf{s}}))))$$

with the size helper function H^{s} given as a weakly monotonic variant of the recursor over $\mathbb{N}$:

$$H^{\mathsf{s}}(x, y, F) = \begin{cases} y & \text{if } x \leq 1 \\ \max(y, F(x-1, H^{\mathsf{s}}(x-1, y, F))) & \text{if } x > 1 \end{cases}$$

7 Conclusions and Future Work

In this paper we introduced an interpretation method for higher-order rewriting with weak call-by-value reduction. In this approach, we build on existing work defining tuple interpretations [21, 33], but restrict the evaluation strategy, and define a cost–size semantics for types and terms which generates a whole new class of cost–size termination models that can be used to reason about both termination and complexity of weak call-by-value systems. We showed that cost–size tuples correctly capture call-by-value termination and allow us to bound both the cost (number of steps to reach normal forms) and a variety of size notions for different data types. A second advantage of our approach compared to [21] is that the cost functionals are now weakly rather than strongly monotonic functionals, which simplifies the search for cost interpretations.

This is foundational work in the research direction of transposing the methodology and tools from (higher-order) term rewriting to program analysis. A first step for future work is to consider more expressive type theories, so we can capture more programs. For instance, the power of the techniques developed here would be greatly improved if polymorphic types are taken into account. A second step is to expand other complexity methods for innermost/call-by-value rewriting to the higher-order setting, such as dependency tuples [27] or polynomial path orders [3]. Also for termination analysis, it would be interesting to combine tuple interpretations with a higher-order variant of innermost dependency pairs [28], similar to what was done for full rewriting with tuple interpretations in [22].

Finally, we plan to implement this work, to automatically derive bounds to the derivation height of individual terms, as well as provide bounds for both full and call-by-value runtime complexity of higher-order term rewriting systems. The automation approach could build on the strategy for higher-order polynomial interpretations for full rewriting (not using tuples) in [14, Section 5]. While the search for tuple interpretations has more unknowns than the search for interpretations to ℕ, and will therefore likely take longer, we expect that the overall methodology can stay largely unchanged at least when it comes to an unrestricted evaluation strategy. Adapting to weak call-by-value rewriting may require some additional study, however.

References

1 T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *TCS*, 2000. doi:10.1016/S0304-3975(99)00207-8.

2 M. Avanzini and U. Dal Lago. Automating sized-type inference for complexity analysis. In *Proc. ICFP*, 2017. doi:10.1145/3110287.

3 M. Avanzini and G. Moser. Complexity analysis by rewriting. In *Proc. FLOPS*, 2008. doi:10.1007/978-3-540-78969-7_11.

4 P. Baillot and U. Dal Lago. Higher-order interpretations and program complexity. *IC*, 2016. doi:10.1016/j.ic.2015.12.008.

5 H. Barendregt, W. Dekkers, and R. Statman. *Lambda Calculus with Types*. Perspectives in Logic. Cambridge University Press, 2013. doi:10.1017/CBO9781139032636.

6 A. Ben Cherifa and P. Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9(2):137–159, 1987. doi:10.1016/0167-6423(87)90030-X.

7 G. Bonfante, A. Cichon, J.-Y. Marion, and H. Touzet. Algorithms with polynomial interpretation termination proof. *Journal of Functional Programming*, 11(1):33–53, 2001. doi:10.1017/S0956796800003877.

8 G. Bonfante, J. Marion, and J. Moyen. On lexicographic termination ordering with space bound certifications. In *Proc. PSI*, 2001. doi:10.1007/3-540-45575-2_46.

9 A. Cichon and P. Lescanne. Polynomial interpretations and the complexity of algorithms. In *CADE*, pages 139–147, 1992. doi:10.1007/3-540-55602-8_161.

10 M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, and J.F. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285(2):187–243, 2002. Rewriting Logic and its Applications. doi:10.1016/S0304-3975(01)00359-0.

11 M. Codish, I. Gonopolskiy, A. M. Ben-Amram, C. Fuhs, and J. Giesl. Sat-based termination analysis using monotonicity constraints over the integers. *Theory and Practice of Logic Programming*, 11(4-5):503–520, 2011. doi:https://doi.org10.1017/S1471068411000147.

12 E. Contejan, C. Marché, A. P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *JAR*, 34, 2005. doi:10.1007/s10817-005-9022-x.

13 N. Danner, D.R. Licata, and R. Ramyaa. Denotational cost semantics for functional languages with inductive types. In *Proc. ICFP*, 2015. doi:10.1145/2784731.2784749.

14 C. Fuhs and C. Kop. Polynomial interpretations for higher-order rewriting. In *Proc. RTA*, 2012. doi:10.4230/LIPIcs.RTA.2012.176.

15 Jürgen Giesl, Christoph Walther, and Jürgen Brauburger. Termination analysis for functional programs. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction — A Basis for Applications: Volume III Applications*, pages 135–164. Springer Netherlands, Dordrecht, 1998. doi:10.1007/978-94-017-0437-3_6.

16 N. Hirokawa and G. Moser. Automated complexity analysis based on the dependency pair method. In *Proc. IJCAR*, 2008. doi:10.1007/978-3-540-71070-7_32.

17 D. Hofbauer. Termination proofs by multiset path orderings imply primitive recursive derivation lengths. *TCS*, 1992. doi:10.1007/3-540-53162-9_50.

18 D. Hofbauer. Termination proofs by context-dependent interpretations. In *Proc. RTA*, 2001. doi:10.1007/3-540-45127-7_10.

19 D. Hofbauer and C. Lautemann. Termination proofs and the length of derivations. In *Proc. RTA*, 1989. doi:10.1007/3-540-51081-8_107.

20 D. M. Kahn and J. Hoffmann. Exponential automatic amortized resource analysis. In *Proc. FoSSaCS*, 2020. doi:10.1007/978-3-030-45231-5_19.

21 C. Kop and D. Vale. Tuple interpretations for higher-order complexity. In *Proc. FSCD*, 2021. doi:10.4230/LIPIcs.FSCD.2021.31.

22 Cynthia Kop. Cutting a Proof into Bite-Sized Chunks: Incrementally proving termination in higher-order term rewriting. In *Proc. FSCD22*, pages 1:1–1:17, 2022. doi:10.4230/LIPIcs.FSCD.2022.1.

23 F. Mitterwallner and A. Middeldorp. Polynomial Termination Over ℕ Is Undecidable. In *Proc. FSCD*, pages 27:1–27:17, 2022. doi:10.4230/LIPIcs.FSCD.2022.27.

24 G. Moser. Uniform Resource Analysis by Rewriting: Strengths and Weaknesses (Invited Talk). In *Proc. FSCD*, pages 2:1–2:10, 2017. doi:10.4230/LIPIcs.FSCD.2017.2.

25 G. Moser, A. Schnabl, and J. Waldmann. Complexity Analysis of Term Rewriting Based on Matrix and Context Dependent Interpretations. In *Proc. IARCS*, pages 304–315, 2008. doi:10.4230/LIPIcs.FSTTCS.2008.1762.

26 Y. Niu and J. Hoffmann. Automatic space bound analysis for functional programs with garbage collection. In *Proc. LPAR*, 2018. doi:10.29007/xkwx.

27 L. Noschinski, F. Emmes, and J. Giesel. Analysing innermost runtime complexity of term rewriting by dependency pairs. *JAR*, 2013. doi:10.1007/s10817-013-9277-6.

28 L. Noschinski, F. Emmes, and J. Giesl. A dependency pair framework for innermost complexity analysis of term rewrite systems. In *CADE-23*, pages 422–438, 2011. doi:10.1007/978-3-642-22438-6_32.

29 J.C. van de Pol. *Termination of Higher-order Rewrite Systems*. PhD thesis, University of Utrecht, 1996. URL: https://www.cs.au.dk/~jaco/papers/thesis.pdf.

30 Yoshihito T. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25(3):141–143, 1987. doi:10.1016/0020-0190(87)90122-0.

31 A. Weiermann. Termination proofs for term rewriting systems by lexicographic path orderings imply multiply recursive derivation lengths. *TCS*, 1995. doi:10.1016/0304-3975(94)00135-6.

32 Wiki. The International Termination Competition (TermComp). Annual International Termination Competition, 2018. URL: http://termination-portal.org/wiki/Termination_Competition.

33 A. Yamada. Multi-dimensional interpretations for termination of term rewriting. In *In. Proc. CADE28*, volume 12699 of *Lecture Notes in Computer Science*, pages 273–290, 2021. doi:10.1007/978-3-030-79876-5_16.

Categorical Coherence from Term Rewriting Systems

Samuel Mimram
LIX, École polytechnique, Palaiseau, France

Abstract

The celebrated Squier theorem allows to prove coherence properties of algebraic structures, such as MacLane's coherence theorem for monoidal categories, based on rewriting techniques. We are interested here in extending the theory and associated tools simultaneously in two directions. Firstly, we want to take in account situations where coherence is partial, in the sense that it only applies for a subset of structural morphisms (for instance, in the case of the coherence theorem for symmetric monoidal categories, we do not want to strictify the symmetry). Secondly, we are interested in structures where variables can be duplicated or erased. We develop theorems and rewriting techniques in order to achieve this, first in the setting of abstract rewriting systems, and then extend them to term rewriting systems, suitably generalized in order to take coherence in account. As an illustration of our results, we explain how to recover the coherence theorems for monoidal and symmetric monoidal categories.

2012 ACM Subject Classification Theory of computation → Rewrite systems

Keywords and phrases coherence, rewriting system, Lawvere theory

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.16

1 Introduction

A *monoidal category* consists of a category C equipped with a tensor bifunctor $\otimes : C \times C \to C$ and unit element $e : 1 \to C$ together with natural isomorphisms $\alpha_{x,y,z} : (x \otimes y) \otimes z \to x \otimes (y \otimes z)$, $\lambda_x : e \otimes x \to x$ and $\rho_x : x \otimes e \to x$, satisfying two well-known axioms. Thanks to these, the way tensor expressions are bracketed does not really matter: we can always rebracket expressions using the structural morphisms (α, λ and ρ), and any two ways of rebracketing an expression into the other are equal. In fact, there are various ways to formalize this [1]:

(C1) Every diagram in a free monoidal category made up of α, λ and ρ commutes [17, Corollary 1.6], [26, Theorem VI.2.1].

(C2) Every diagram in a monoidal category made up of α, λ and ρ commutes [27, Theorem 3.1], [26, Theorem XI.3.2].

(C3) Every monoidal category is monoidally equivalent to a strict monoidal category [17, Corollary 1.4], [26, Theorem XI.3.1].

(C4) The forgetful 2-functor from strict monoidal categories to monoidal categories has a left adjoint and the components of the unit are equivalences.

Condition (C1) implies (C2) as a particular case and the converse implication can also be shown. Condition (C4) implies (C3) as a particular case, and it can be shown that (C3) in turn implies (C2). Analogous statements hold for symmetric monoidal categories (monoidal categories equipped with a suitable symmetry $\gamma_{x,y} : x \otimes y \to y \otimes x$) although they are more subtle [2]: in (C2), we have to suppose that the diagrams are "generic enough", and in (C4) the notion of strict symmetric monoidal category does not impose that the symmetry should be an identity.

We first investigate here (in Section 2) an abstract version of this situation and formally compare the various coherence theorems: we show that quotienting a theory by a subtheory $\mathcal{W}$ gives rise to an equivalent theory if and only if $\mathcal{W}$ is coherent (or rigid), in the sense that all diagrams commute (Theorem 6). Moreover, this is the case if and only if they give rise to equivalent categories of algebras (Proposition 9), which can be thought of as a strengthened version of (C4). We also provide rewriting conditions which allow showing coherence in practice (Proposition 13). The idea of extending rewriting theory in order to take coherence in account dates back to pioneering work from people such as Power [30], Street [34] and Squier [32]. It has been generalized in higher dimensions in the context of polygraphs [33, 8], as well as homotopy type theory [19], and used to recover various coherence theorem [22, 13].

We then extend (in Section 3) our results to the 2-dimensional cartesian theories, which are able to axiomatize (symmetric) monoidal categories. Our work is based on the notion of Lawvere 2-theory [12, 35, 36], and unfortunately lead us to discover an important flaw in a main result about those [36]. The rewriting counterpart is based on a coherent extension of term rewriting systems, following [10, 5, 28]. One of the main novelties here consists in allowing for coherence with respect to a sub-theory (which is required to handle coherence for symmetric monoidal categories), building on recent works in order to work in structures modulo substructures [9, 29, 11].

2 Relative coherence and abstract rewriting systems

2.1 Quotient of categories

Suppose fixed a category $\mathcal{C}$ together with a set W of isomorphisms of $\mathcal{C}$. Although the situation is very generic, and the following explanation is only vague for now, it can be helpful to think of $\mathcal{C}$ as a theory describing a structure a category can possess and W as the morphisms we are interested in strictifying. For instance, if we are interested in the coherence theorem for symmetric monoidal categories, we can think of the objects of $\mathcal{C}$ as formal iterated tensor products, the morphisms of $\mathcal{C}$ as composites of α, λ, ρ and γ, and we would typically take W as consisting of all instances of α, λ and ρ (but not γ). This will be made formal in Section 3.

A functor $F : \mathcal{C} \to \mathcal{D}$ is *W-strict* when it sends every morphism of W to an identity. We write $\mathcal{C}/W$ for the *quotient* of $\mathcal{C}$ under W: this is the category equipped with a W-strict functor $\mathcal{C} \to \mathcal{C}/W$ such that any W-strict functor $F : \mathcal{C} \to \mathcal{D}$ extends uniquely as a functor $\tilde{F} : \mathcal{C}/W \to \mathcal{D}$. We write $\mathcal{W}$ for the subcategory of $\mathcal{C}$ generated by W (which we assimilate to a subset of the morphisms of $\mathcal{C}$). This is always a groupoid (a category in which every morphism is invertible) and it is easily shown that $\mathcal{C}/\mathcal{W} \cong \mathcal{C}/W$, so that we can always suppose that we quotient by a subgroupoid. Moreover, we can always suppose that this subgroupoid has the same objects as $\mathcal{C}$ (we can add all identities in it without changing the quotient).

We say that a groupoid $\mathcal{W}$ is *rigid* when any two morphisms $f, g : x \to y$ which are parallel (i.e. have the same source, and have the same target) are necessarily equal. Such a groupoid can be thought of as a “coherent” sub-theory of $\mathcal{C}$. It does not have non-trivial geometric structure in the following sense:

▶ **Lemma 1.** *A groupoid $\mathcal{W}$ is rigid if and only if either*

(i) *identities are the only automorphisms of $\mathcal{W}$,*

(ii) *$\mathcal{W}$ is a “set”, i.e. is equivalent to a coproduct of instances of the terminal category.*

The fact that $\mathcal{W} \subseteq \mathcal{C}$ is rigid is thought here as the fact that coherence condition (C1) holds for $\mathcal{C}$, relatively to $\mathcal{W}$.

General notions of quotients of categories are not trivial to construct (see for instance [4]), but in the case of rigid categories, we have the following simple description.

▶ **Proposition 2.** *When $\mathcal{W} \subseteq \mathcal{C}$ is rigid, the quotient category $\mathcal{C}/\mathcal{W}$ is isomorphic to the category where*

- *objects are equivalence classes $\mathcal{C}_0/\mathcal{W}$ of objects of $\mathcal{C}$ under the equivalence relation $\sim$ such that $x \sim y$ whenever there exists $w : x \to y$ in $\mathcal{W}$ (we write $[x]$ for the class of an object x),*
- *a morphism is of the form $[f] : [x] \to [y]$ for some morphism $f : x \to y$ of $\mathcal{C}$, under the equivalence relation such that $f \sim f'$ whenever there exists v and w in $\mathcal{W}$ such that $w \circ f = f' \circ v$,*
- *given $f : x \to y$ and $g : y' \to z$ with $[y] = [y']$, the composition is $[g] \circ [f] = [g \circ w \circ f]$ for the "mediating" morphism $w : y \to y'$ in $\mathcal{W}$ (uniquely determined by rigidity of $\mathcal{W}$),*
- *given $x \in \mathcal{C}$, the identity is $\mathrm{id}_{[x]} = [\mathrm{id}_x]$.*

When $\mathcal{W} \subseteq \mathcal{C}$ is not rigid, we can have a similar description, but we now have the choice between multiple mediating morphisms in the definition of the composition, and all the resulting composites in fact have to be identified in the quotient. This observation suggests that the construction of the quotient category $\mathcal{C}/\mathcal{W}$, when $\mathcal{W}$ is not rigid, is better described in two steps: we first formally make $\mathcal{W}$ rigid, and then apply Proposition 2. We say that a functor $F : \mathcal{C} \to \mathcal{D}$ is $\mathcal{W}$*-rigid* when any two parallel morphisms of $\mathcal{W}$ have the same image. The $\mathcal{W}$*-rigidification* of $\mathcal{C}$ is the category $\mathcal{C}/\!\!/\mathcal{W}$ equipped with a $\mathcal{W}$-rigid functor $\mathcal{C} \to \mathcal{C}/\!\!/\mathcal{W}$ such that any $\mathcal{W}$-rigid functor $F : \mathcal{C} \to \mathcal{D}$ extends uniquely as a functor $\mathcal{C}/\!\!/\mathcal{W} \to \mathcal{D}$.

▶ **Lemma 3.** *The category $\mathcal{C}/\!\!/\mathcal{W}$ is the category obtained from $\mathcal{C}$ by quotienting morphisms under the smallest congruence (wrt composition) identifying any two parallel morphisms of $\mathcal{W}$.*

▶ **Proposition 4.** *The quotient $\mathcal{C}/\mathcal{W}$ is isomorphic to $(\mathcal{C}/\!\!/\mathcal{W})/\tilde{\mathcal{W}}$ where $\tilde{\mathcal{W}}$ is the set of equivalence classes of morphisms in $\mathcal{W}$ under the equivalence relation of Lemma 3.*

Proof. Follows directly from the universal properties of the quotient and the rigidification, and the fact that any $\mathcal{W}$-strict functor is $\mathcal{W}$-rigid. ◀

A consequence of the preceding explicit description of the quotient is the following:

▶ **Lemma 5.** *The quotient functor $\mathcal{C} \to \mathcal{C}/\mathcal{W}$ is surjective on objects and full.*

Proof. By Proposition 4, the quotient functor is the composite of the quotient functors $\mathcal{C} \to \mathcal{C}/\!\!/\mathcal{W} \to \mathcal{C}/\mathcal{W}$. The first one is surjective on objects and full by Lemma 3 and the second one is surjective on objects and full by Proposition 2. ◀

This entails the following theorem, which is the main result of the section. Its meaning can be explained by taking the point of view given above: thinking of $\mathcal{C}$ as describing a structure and of $\mathcal{W}$ as a part of the structure we want to strictify, the structure is equivalent to its strict variant if and only if the quotiented structure does not itself bear non-trivial geometry (in the sense of Lemma 1).

▶ **Theorem 6.** *Suppose that $\mathcal{W}$ is a subgroupoid of $\mathcal{C}$. The quotient functor $[-] : \mathcal{C} \to \mathcal{C}/\mathcal{W}$ is an equivalence of categories if and only if $\mathcal{W}$ is rigid.*

Proof. Since the quotient functor is always surjective and full by Lemma 5, it remains to show that it is faithful if and only if $\mathcal{W}$ is rigid. Suppose that the quotient functor is faithful. Given $w, w' : x \to y$ in $\mathcal{W}$, by Lemma 3 and Proposition 4 we have $[w] = [w']$ and thus $w = w'$ by faithfulness. Suppose that $\mathcal{W}$ is rigid. The category $\mathcal{C}/\mathcal{W}$ then admits the description given in Proposition 2. Given $f, g : x \to y$ in $\mathcal{C}$ such that $[f] = [g]$, there is $v : x \to x$ and $w : y \to y$ such that $w \circ f = g \circ v$. By rigidity, both v and w are identities and thus $f = g$. ◀

▶ **Example 7.** As a simple example, consider the groupoid $\mathcal{C}$ freely generated by the graph $\begin{tikzcd} x \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & y \end{tikzcd}$. The subgroupoid generated by $W = \{g\}$ is rigid, so that $\mathcal{C}$ is equivalent to the quotient category $\mathcal{C}/W$, which is the groupoid generated by $\begin{tikzcd} x \arrow[loop right, "f"] \end{tikzcd}$. However, the groupoid generated by $W = \{f, g\}$ is not rigid (since we don't have $f = g$). And indeed, $\mathcal{C}$ is not equivalent to the quotient category $\mathcal{C}/W$, which is the terminal category.

2.2 Coherence for algebras

Given a category $\mathcal{C}$, we consider here a functor $\mathcal{C} \to \mathbf{Cat}$ as an *algebra* for $\mathcal{C}$. Namely, if we think of the category $\mathcal{C}$ as describing an algebraic structure (e.g. the one of monoidal categories), an algebra can be thought of as a category actually possessing this structure (an actual monoidal category).

We write $\operatorname{Alg}(\mathcal{C})$ for the category of algebras, with natural transformations as morphisms. Any functor $F : \mathcal{C} \to \mathcal{C}'$ induces, by precomposition, a functor $\operatorname{Alg}(F) : \operatorname{Alg}(\mathcal{C}') \to \operatorname{Alg}(\mathcal{C})$. We can characterize situations where two categories give rise to the same algebras:

▶ **Proposition 8.** *Suppose given a functor $F : \mathcal{C} \to \mathcal{C}'$ between categories. The functor F is an equivalence if and only if the induced functor $\operatorname{Alg}(F) : \operatorname{Alg}(\mathcal{C}') \to \operatorname{Alg}(\mathcal{C})$ is an equivalence.*

Proof. Given a 2-category $\mathcal{K}$, one can define a Yoneda functor $Y_{\mathcal{K}} : \mathcal{K}^{\mathrm{op}} \to [\mathcal{K}, \mathbf{Cat}]$, where $\mathbf{Cat}$ is the 2-category of categories, functors and natural transformations, and $[\mathcal{K}, \mathbf{Cat}]$ denotes the 2-category of 2-functors $\mathcal{K} \to \mathbf{Cat}$, transformations and modifications. In particular, given 0-cells $x \in \mathcal{K}^{\mathrm{op}}$ and $y \in \mathcal{K}$, we have $Y_{\mathcal{K}} x y = \mathcal{K}(x, y)$. The Yoneda lemma states that this functor is a local isomorphism (this is a particular case of the Yoneda lemma for bicategories detailed for instance in [16, chapter 8]). In particular, taking $\mathcal{K} = \mathbf{Cat}$ (and ignoring size issues), the Yoneda functor sends a category $\mathcal{C} \in \mathcal{K}^{\mathrm{op}}$ to $Y_{\mathcal{K}}\mathcal{C} = \operatorname{Alg}(\mathcal{C})$, and the result follows from the Yoneda lemma. ◀

As a particular application, given a category $\mathcal{C}$ and a subgroupoid $\mathcal{W}$, we have by Theorem 6 that $\mathcal{W}$ is rigid if and only if the quotient functor $\mathcal{C} \to \mathcal{C}/\mathcal{W}$ is an equivalence. By Proposition 8, we thus have the following property which can be interpreted as the equivalence of coherence conditions (C1) and a strengthened variant of (C4).

▶ **Proposition 9.** *Given a category $\mathcal{C}$ and a subgroupoid $\mathcal{W}$, the morphism $\operatorname{Alg}(\mathcal{C}/\mathcal{W}) \to \operatorname{Alg}(\mathcal{C})$ induced by the quotient functor is an equivalence of categories if and only if $\mathcal{W}$ is rigid.*

2.3 Coherent abstract rewriting systems

We now explain how the theory rewriting can be used to show the rigidity of a groupoid in practice. In the same way the theory of rewriting can be studied abstractly [15, 3, 6], i.e. without taking in consideration the structure of the objects getting rewritten, we first develop the coherence theorems of interest in this article in an abstract setting. Although the terminology is different, the formalization given here is based on the notion of polygraph [33, 8].

Extended abstract rewriting systems. An *abstract rewriting system*, or ARS, $\mathsf{P} = (\mathsf{P}_0, s_0, t_0, \mathsf{P}_1)$ consists of a set P_0, a set P_1 and two functions $s_0, t_0 : \mathsf{P}_1 \to \mathsf{P}_0$. The elements of P_0 are generally thought as the *objects* of interest, the elements of P_1 as *rewriting rules*, and the function s_0 (resp. t_0) associating to a rewriting rule its *source* (resp. *target*). We write $a : x \to y$ for a rewriting rule a with $s_0(a) = x$ and $t_0(a) = y$. We write P_1^* for the set of *rewriting paths* in the ARS: its elements are (possibly empty) finite sequences $a_1, \ldots, a_n$

of rewriting steps, which are composable in the sense that $t_0(a_i) = s_0(a_{i+1})$ for $1 \leq i < n$. The source (resp. target) of such a rewriting path is $s_0(a_1)$ (resp. $t_0(a_n)$); we sometimes write $p : x \xrightarrow{*} y$ to indicate that p is a rewriting path with x as source and y as target. Given two composable paths $p : x \xrightarrow{*} y$ and $q : y \xrightarrow{*} z$, we write $p \cdot q$ for their concatenation.

A morphism $f : \mathsf{P} \to \mathsf{Q}$ of ARS is a pair of functions $f_0 : \mathsf{P}_0 \to \mathsf{Q}_0$ and $f_1 : \mathsf{P}_1 \to \mathsf{Q}_1$ such that $s_0 \circ f_1 = f_0 \circ s_0$ and $t_0 \circ f_1 = f_0 \circ t_0$, and we write $\mathbf{Pol}_1$ for the resulting category. There is a forgetful functor $\mathbf{Cat} \to \mathbf{Pol}_1$, sending a category C to the ARS whose objects are those of C and whose rewriting steps are the morphisms of C. This functor admits a left adjoint $-^* : \mathbf{Pol}_1 \to \mathbf{Cat}$ sending an ARS to the category with P_0 as objects and P_1^* as morphisms (composition is given by concatenation of paths and identities are the empty paths).

As a variant of the preceding situation, we can consider the forgetful functor $\mathbf{Gpd} \to \mathbf{Pol}_1$, from the category of groupoids. It also admits a left adjoint $-^{\sim} : \mathbf{Pol}_1 \to \mathbf{Gpd}$, and we write $\mathsf{P}_1^{\sim}$ for the set of morphisms of the groupoid generated by an ARS. The elements of P_1 are *rewriting zig-zags* in the ARS: they consist in finite sequences $a_1^{\epsilon_1}, \ldots, a_n^{\epsilon_n}$ with $a_i \in \mathsf{P}_1$ and $\epsilon_i \in \{-, +\}$ for $1 \leq i \leq n$, which are

- composable: $t_0(a_i^{\epsilon_i}) = s_0(a_{i+1}^{\epsilon_{i+1}})$ for $1 \leq i < n$,
  by convention $s_0(a_i^+) = s_0(a_i)$, $t_0(a_i^+) = t_0(a_i)$, $s_0(a_i^-) = t_0(a_i)$, $t_0(a_i^-) = s_0(a_i)$, and
- reduced: if $a_i = a_{i+1}$ then $\epsilon_i = \epsilon_{i+1}$ for $1 \leq i < n$.

The intuition is that a zig-zag is a "non-directed" rewriting path, consisting of rewriting steps, some of which are taken backward (i.e. formally inverted: those for which the exponent is "$-$"). The source (resp. target) of a zig-zag as above is $s_0(a_1^{\epsilon_1})$ (resp. $t_0(a_n^{\epsilon_n})$) and we write $p : x \xrightarrow{\sim} y$ to indicate that p is a zig-zag from x to y. Composition $p \cdot q$ of composable zig-zags $p : x \xrightarrow{\sim} y$ and $q : y \xrightarrow{\sim} y$ is given by taking their concatenation and iteratively removing the subpaths of the form $a^- \cdot a^+$ or $a^+ \cdot a^-$ at the interface, which ensures that the composite is reduced. Given a zig-zag p, we write p^- for its inverse, obtained by inverting the polarity of the exponents in p (we exchange "+" and "$-$"): it satisfies $p \cdot p^- = \mathrm{id}$ and $p^- \cdot p = \mathrm{id}$, where id denotes an empty zig-zag. Note that there is a canonical inclusion $\mathsf{P}_1^* \to \mathsf{P}_1^{\sim}$, which adds a "+" exponent to every step of a rewriting path, witnessing for the fact that rewriting paths are particular zig-zags.

An *extended abstract rewriting system*, or 2-ARS, P consists of an ARS as above, together with a set P_2 and two functions $s_1, t_1 : \mathsf{P}_2 \to \mathsf{P}_1^{\sim}$, such that $s_0 \circ s_1 = s_0 \circ t_1$ and $t_0 \circ s_1 = t_0 \circ t_1$. The elements of P_2 are *coherence relations* and the functions respectively describe their source and target (which are rewriting paths). We sometimes write $A : p \Rightarrow q$ to indicate that $A \in \mathsf{P}_2$ admits p (resp. q) as source (resp. target), which can be thought of as a 2-cell

$$\begin{tikzcd} x \arrow[r, bend left, "p"{name=U}] \arrow[r, bend right, "q"'{name=D}] & y \arrow[from=U, to=D, Rightarrow, "A"'] \end{tikzcd}$$

where x (resp. y) is the common source (resp. target) of p and q. The notion of 2-ARS is a groupoidal variant of the one of 2-*computad* [33] aka 2-*polygraph* [8], which generalizes in arbitrary dimension. The *groupoid presented* by a 2-ARS P, denoted by $\overline{\mathsf{P}}$, is the groupoid obtained from the free groupoid generated by the underlying ARS by quotienting morphisms under the smallest congruence identifying the source and the target of any element of P_2. The groupoid $\overline{\mathsf{P}}$ thus has P_0 as set of objects, the set $\mathsf{P}_1^{\sim}$ of rewriting zig-zags as morphisms, quotiented by the smallest equivalence relation $\equiv$ such that $p \cdot q \cdot r \equiv p \cdot q' \cdot r$ for every rewriting zig-zags p and r and coherence relation $A : q \Rightarrow q'$, which are suitably composable. Given a rewriting zig-zag $p \in \mathsf{P}_1^{\sim}$, we write $\overline{p}$ for the corresponding morphism in $\overline{\mathsf{P}}$ (i.e. its equivalence class under $\equiv$).

Rewriting properties. Now, suppose fixed a 2-ARS P together with a set $W \subseteq \mathsf{P}_1$. We can think of W as inducing a rewriting subsystem W of P, with P_0 as objects, W as rewriting steps and $\mathsf{W}_2 = \{A \in \mathsf{P}_2 \mid s_1(A) \in W^* \text{ and } t_1(A) \in W^*\}$ as coherence relations, and formulate the various traditional rewriting concepts with respect to it. We are considering here the quotient of a category $\mathcal{C} = \overline{\mathsf{P}}$ presented by a 2-ARS P by the subgroupoid $\mathcal{W}$ generated by W, with the aim of showing coherence results wrt the strictification of $\mathcal{W}$ as previously.

We say that P is *W-terminating* if there is no infinite sequence $a_1, a_2, \ldots$ of elements of W such that every finite prefix is a rewriting path (i.e. belongs to W^*). An element $x \in \mathsf{P}_0$ is a *W-normal form* when there is no rewriting step in W with x as source. We say that P is *weakly W-normalizing* when for every $x \in \mathsf{P}_0$ there exists a normal form $\hat{x}$ and a rewriting path $n_x : x \overset{*}{\to} \hat{x}$. We necessarily have $\hat{\hat{x}} = \hat{x}$ and we always suppose that $n_{\hat{x}} = \mathrm{id}_{\hat{x}}$.

▶ **Lemma 10.** *If P is W-terminating then it is weakly W-normalizing.*

Proof. Traditional rewriting argument: a maximal path (wrt prefix order) starting from x exists (because W is terminating) and its target is necessarily a normal form. ◀

A *W-branching* is a pair of rewriting steps $a_1 : x \to y_1$ and $a_2 : x \to y_2$ in W which are coinitial, i.e. have the same source. Such a branching is *confluent* when there is a pair of cofinal (with the same target) rewriting paths $p_1 : y_1 \to z$ and $p_2 : y_2 \to z$ in W^* such that $\overline{a_1 \cdot p_1} = \overline{a_2 \cdot p_2}$ (as morphisms of $\overline{\mathsf{P}}$, or, equivalently, of $\overline{\mathcal{W}}$). We say that P is *locally W-confluent* when W-branching is confluent. This condition is in particular satisfied when there exists a coherence relation $A : a_1 \cdot p_1 \Rightarrow a_2 \cdot p_2$, or $A : a_2 \cdot p_2 \Rightarrow a_1 \cdot p_1$ in P_2. Note that, here, not only we require that we can close a span of rewriting steps by a cospan of rewriting paths (as in the traditional definition of confluence), but also that the confluence square can be filled coherence relations. Similarly, P is *W-confluent* when for every $p_1 : x \overset{*}{\to} y_1$ and $p_2 : x \overset{*}{\to} y_2$ in W^*, there exist $q_1 : y_1 \overset{*}{\to} z$ and $q_2 : y_2 \overset{*}{\to} z$ in W^* such that $\overline{p_1 \cdot q_1} = \overline{p_2 \cdot q_2}$. We say that P is *W-convergent* when it is both W-terminating and W-confluent.

The celebrated Newman's lemma (also sometimes called the diamond lemma) along with its traditional proof [6, Theorem 1.2.1 (ii)] easily generalizes to our setting:

▶ **Proposition 11.** *If P is W-terminating and locally W-confluent then it is W-confluent.*

Proof. Classical argument, by well-founded induction on x, using local W-confluence. ◀

We say that P is *W-coherent* if for any parallel paths $p, q : x \overset{\sim}{\to} y$ in $W^\sim$, we have $\overline{p} = \overline{q}$. In other words, P is W-coherent precisely when $\overline{\mathsf{W}}$ is a rigid subgroupoid of $\overline{\mathsf{P}}$. The traditional Church-Rosser property [6, Theorem 1.2.2] generalizes as follows in our setting:

▶ **Proposition 12.** *If P is weakly W-normalizing and W-confluent then for any zig-zag $p : x \overset{\sim}{\to} y$ in $W^\sim$, we have $\overline{p \cdot n_y} = \overline{n_x}$.*

Proof. By confluence, given a rewriting path $p : x \overset{*}{\to} y$ in W^*, we have $\hat{x} = \hat{y}$ and $\overline{p \cdot n_y} = \overline{n_x}$ (where n_x and n_y are paths to a normal form given by the weak normalization property), and thus $\overline{p^+ \cdot n_y} = \overline{n_x}$ and $\overline{n_x \cdot p^-} = \overline{n_y}$. Any zig-zag $p : x \overset{\sim}{\to} y$ in $W^\sim$ decomposes as $p = p_1^- q_1^+ p_2^- p_2^+ \ldots p_n^- p_n^+$ for some $n \in \mathbb{N}$ and paths p_i and q_i in W^*. We thus have $\overline{p \cdot n_y} = \overline{n_x}$, since all the squares of the following diagram commute in $W^\sim$ by the preceding remark:

$$\begin{tikzcd}
x \arrow[r, "p_1^-"] \arrow[d, "n_x"'] & y_1 \arrow[r, "q_1^+"] \arrow[d, dashed, "n_{y_1}"] & x_2 \arrow[r] \arrow[d, dashed, "n_{x_2}"] & \cdots \arrow[r] & x_n \arrow[r, "p_n^-"] \arrow[d, dashed, "n_{x_n}"] & y_n \arrow[r, "q_n^-"] \arrow[d, dashed, "n_{y_n}"] & y \arrow[d, dashed, "n_y"] \\
\hat{x} \arrow[r, equal] & \hat{x} \arrow[r, equal] & \hat{x} \arrow[r, equal] & \cdots \arrow[r, equal] & \hat{x} \arrow[r, equal] & \hat{x} \arrow[r, equal] & \hat{x}
\end{tikzcd}$$

which allows us to conclude. ◀

This implies the following "abstract" variant of Squier's homotopical theorem [32, 21, 14]:

▶ **Proposition 13.** *If* P *is weakly W-normalizing and is W-confluent then it is W-coherent.*

Proof. Given two parallel zig-zags $p, q : x \overset{\sim}{\to} y$ in $W^\sim$, we have $\overline{p} = \overline{q}$, since the following diagram commutes in $\overline{\mathsf{P}}$:

$$\begin{tikzcd}
& y \arrow[dr, "n_y"] \arrow[drr, bend left, "\mathrm{id}_y"] & & \\
x \arrow[ur, "p"] \arrow[rr, "n_x"] \arrow[dr, "q"'] & & \hat{y} \arrow[r, "n_y^-"] & y \\
& y \arrow[ur, "n_y"'] \arrow[urr, bend right, "\mathrm{id}_y"'] & &
\end{tikzcd}$$

Namely, we have $\hat{x} = \hat{y}$ by confluence, the two triangles above commute by Proposition 12, and the two triangles below do because n_y^- is an inverse for n_y. ◀

▶ **Example 14.** As a variant of Example 7, consider the 2-ARS P with $\mathsf{P}_0 = \{x, y\}$, $\mathsf{P}_1 = \{a, b : x \to y\}$ and $\mathsf{P}_2 = \emptyset$, i.e. $x \overset{a}{\underset{b}{\rightrightarrows}} y$. With $W = \{a\}$, we have that P is W-terminating and locally W-confluent, thus W-confluent by Proposition 11, and thus W-coherent by Lemma 10 and Proposition 13. With $W = \{a, b\}$, we have seen in Example 7 that the groupoid $\overline{W}$ is not rigid and, indeed, P is not W-confluent because $\overline{a} \neq \overline{b}$ (because $\mathsf{P}_2 = \emptyset$).

In a situation as above, we write $\mathrm{N}(\overline{\mathsf{P}})$ for the full subcategory of $\overline{\mathsf{P}}$ whose objects are W-normal forms. When P is weakly W-normalizing, we have that every object x of $\overline{\mathsf{P}}$ is isomorphic to one in the image by $\overline{n_x}$, and thus the inclusion functor $\mathrm{N}(\overline{\mathsf{P}}) \to \overline{\mathsf{P}}$ is an equivalence of categories. This equivalence is precisely the one with the quotient category when P is W-convergent:

▶ **Proposition 15.** *If* P *is W-convergent, the quotient category is isomorphic to the category of normal forms:* $\overline{\mathsf{P}}/W \cong \mathrm{N}(\overline{\mathsf{P}})$.

Proof. Since P is W-convergent, by Proposition 13, the groupoid generated by W is rigid and we thus have the description of the quotient $\overline{\mathsf{P}}/W$ given by Proposition 2. We have a canonical functor $\mathrm{N}(\overline{\mathsf{P}}) \to \overline{\mathsf{P}}/W$, obtained as the composite of the inclusion functor $\mathrm{N}(\overline{\mathsf{P}}) \to \overline{\mathsf{P}}$ with the quotient functor $\overline{\mathsf{P}} \to \overline{\mathsf{P}}/W$. By convergence, an equivalence class $[x]$ of objects contains a unique normal form (which is $\hat{x}$), and the functor is bijective on objects. By weak normalization (Lemma 10), any morphism $f : x \to y$ is equivalent to one with both normal source and target, namely $n_y \circ f \circ n_x^- : \hat{x} \to \hat{y}$, hence the functor is full. Suppose given two morphisms $f, g : \hat{x} \to \hat{y}$ in $\hat{\mathsf{P}}$ with the same image $[f] = [g]$: there exist morphisms $v : \hat{x} \to \hat{x}$ and $w : \hat{y} \to \hat{y}$ in $W^\sim$ such that $w \circ f = g \circ v$. By the Church-Rosser property (Proposition 13), we have $v = n_{\hat{x}} \circ n_{\hat{x}}^-$ and thus $v = \mathrm{id}_x$ (since $n_{\hat{x}} = \mathrm{id}_{\hat{x}}$ by hypothesis), and similarly $w = \mathrm{id}_y$. Hence $f = g$ and the functor is faithful. ◀

We would now like to provide an explicit description of $\mathrm{N}(\overline{\mathsf{P}})$. Since the rules in W are quotiented out, we can expect that they can simply be removed from P. This is not the case in general, but we provide here conditions which ensure that it holds, see also [9, 29] for alternative conditions. We write $\mathsf{P} \setminus W$ for the 2-ARS where $(\mathsf{P} \setminus W)_0 = \mathsf{P}_0$, $(\mathsf{P} \setminus W)_1 = \mathsf{P}_1 \setminus W$ and $(\mathsf{P} \setminus W)_2$ is P_2 restricted to rewriting rules whose source and target both belong to $(\mathsf{P}_1 \setminus W)^\sim$.

▶ **Proposition 16.** *Writing* $\mathsf{P}' = \mathsf{P} \setminus W$, *suppose that*

1. P *is W-convergent,*
2. *every rule* $a : x \to y$ *in* P'_1 *has a normal target* $\hat{y} = y$,
3. *for every coinitial rules* $a : x \to y$ *in* P'_1 *and* $w : x \to x'$ *in* W, *there is a path* $p : x' \overset{*}{\to} y$ *in* P'^*_1 *such that* $\overline{a} = \overline{w \cdot p}$,

4. *for every $A : p \Rightarrow q : x \to y$ in P_2 and rule $w : x \to x'$ in W, there is a relation $A' : p' \Rightarrow q' : x' \to y$ in P'_2 (or, more generally, $\overline{p}' = \overline{q}'$ in $\overline{\mathsf{P}}'$) where $p', q' : x' \to y$ are paths such that $\overline{p} = \overline{w \cdot p'}$ and $\overline{q} = \overline{w \cdot q'}$.*

Then $\mathrm{N}(\overline{\mathsf{P}})$ is isomorphic to $\mathrm{N}(\overline{\mathsf{P}}')$.

Proof. We claim that for every zig-zag $p : x \overset{\sim}{\to} y$ in $\mathsf{P}_1^{\sim}$ there is zig-zag $q \in \mathsf{P}_1'^{\sim}$ such that $\overline{p} = \overline{n_x \cdot q \cdot n_y^-}$. We have that p is of the form $p = w_0 \cdot a_1 \cdot w_1 \cdot a_2 \cdot w_2 \cdot \ldots \cdot a_n \cdot w_n$ where the a_i are rules in P'_1 (possibly taken backward) and the w_i are in $W^{\sim}$. For instance, consider the case $n = 1$ and a path p of the form $p = v \cdot a \cdot w$ with $a \in \mathsf{P}'_1$ and $v, w \in W^{\sim}$ (the case where a is reversed is similar, and the general case follows by induction):

$$\begin{tikzcd}
x \arrow[rr, "v"] \arrow[dr, "n_x"'] & & x' \arrow[r, "a"] \arrow[dl, "n_{x'}"] & y' \arrow[rr, "w"] \arrow[dr, "n_{y'}"'] & & y \arrow[dl, "n_y"] \\
& \hat{x} \arrow[rrr, dotted, "q"'] & & & \hat{y} &
\end{tikzcd}$$

By hypothesis 1 and Proposition 12, we have $\overline{v} = \overline{n_x \cdot n_{x'}^-}$ and $\overline{w} = \overline{n_{y'} \cdot n_y^-}$. By hypothesis 2, $n_{y'}$ is the empty path. By iterated use of hypothesis 3, there is $q \in \mathsf{P}_1'^{*}$ such that $\overline{a} = \overline{n_{x'} \cdot q}$.

The canonical functor $F : \mathrm{N}(\overline{\mathsf{P}}') \to \mathrm{N}(\overline{\mathsf{P}})$ is the identity on objects. The above reasoning shows that it is full. It can also be shown to be faithful by iterated use of hypothesis 4. ◀

We can finally summarize the results obtained in this section as follows. Given a 2-ARS P and a set $W \subseteq \mathsf{P}_1$, we have the following possible reasonable definitions of the fact that P is *coherent* wrt W:

(1) Every parallel zig-zags with edges in W are equal
(i.e. the subgroupoid of $\overline{P}$ generated by W is rigid).
(2) The quotient map $\overline{\mathsf{P}} \to \overline{\mathsf{P}}/W$ is an equivalence of categories.
(3) The canonical morphism $\mathrm{N}(\mathsf{P}) \to \overline{\mathsf{P}}$ is an equivalence.
(4) The inclusion $\mathrm{Alg}(\overline{\mathsf{P}}/W) \to \mathrm{Alg}(\overline{\mathsf{P}})$ is an equivalence of categories.

▶ **Theorem 17.** *If P is W-convergent then all the above coherence properties hold.*

Proof. (1) is given by Proposition 13, (2) is given by (1) and Theorem 6, (3) is given by Proposition 15, and (4) is given by (1) and Proposition 9. ◀

3 Relative coherence and term rewriting systems

In order to use the previous developments in concrete situations, such as (symmetric) monoidal categories, we need to consider a more structured notion of theory. For this reason, we consider here Lawvere 2-theories, as well as the adapted notion of rewriting, which is a coherent extension of the traditional notion of string rewriting systems.

3.1 Lawvere 2-theories

A *Lawvere theory* $\mathcal{T}$ is a cartesian category, with $\mathbb{N}$ as set of objects, and cartesian product given on objects by addition [25] (for simplicity, we restrict here to unsorted theories). In such a theory, we usually restrict our attention to morphisms with 1 as codomain, since $\mathcal{T}(n, m) \cong \mathcal{T}(n, 1)^m$ by cartesianness. A morphism between Lawvere theories is a product-preserving functor and we write $\mathbf{Law}_1$ for the category of Lawvere theories.

A $(2,1)$-*category* is a 2-category in which every 2-cell is invertible (i.e. a category enriched in groupoids). A *Lawvere 2-theory* $\mathcal{T}$, as introduced in [12, 35, 36] (as well as [31] for the enriched point of view), is a cartesian $(2,1)$-category with $\mathbb{N}$ as objects, and cartesian product given on objects by addition. A *morphism* $F : \mathcal{T} \to \mathcal{U}$ between 2-theories is a functor which preserves products. We write $\mathbf{Law}_2$ for the resulting category (which can be extended to a 3-category by respectively taking natural transformations and modifications as 2- and 3-cells).

3.2 Coherence for 2-theories

We can reuse the properties developed in Section 2 by working "hom-wise" as follows. Suppose fixed a 2-theory $\mathcal{T}$ together with a subset W of the 2-cells. We write $\mathcal{W}$ for the sub-2-theory of $\mathcal{T}$, with the same 0- and 1-cells, and whose 2-cells contain W (we often assimilate this 2-theory to its set of 2-cells). The *quotient* 2*-theory* $\mathcal{T}/W$ is the one representing the morphisms from $\mathcal{T}$ sending 2-cells in W to identities; it comes equipped with a quotient 2-functor $\mathcal{T} \to \mathcal{T}/W$. We have $\mathcal{T}/W \cong \mathcal{T}/\mathcal{W}$, so that we can always assume that we are quotienting by a sub-2-theory. On hom-categories, the quotient corresponds to the one introduced in Section 2.1: for every $m, n \in \mathbb{N}$, we have $(\mathcal{T}/\mathcal{W})(m,n) = \mathcal{T}(m,n)/\mathcal{W}(m,n)$.

We say that a morphism $F : \mathcal{T} \to \mathcal{U}$ is a *local equivalence* when for every objects $m, n \in \mathcal{T}$, the induced functor $F_{m,n} : \mathcal{T}(m,n) \to \mathcal{U}(m,n)$ between hom-categories is an equivalence. We say that $\mathcal{W}$ is 2*-rigid* when any two parallel 2-cells are equal, i.e. the category $\mathcal{W}(m,n)$ is rigid for every 0-cells m and n. By direct application of Theorem 6, we have

▶ **Theorem 18.** *The quotient* 2*-functor* $\mathcal{T} \to \mathcal{T}/\mathcal{W}$ *is a local equivalence iff* $\mathcal{W}$ *is* 2*-rigid.*

3.3 Extended rewriting systems

We briefly recall here the categorical setting for term rewriting systems. A more detailed presentation can be found in [10, 5, 28].

A *signature* consists of a set S_1 of *symbols* together with a function $s_0 : \mathsf{S}_1 \to \mathbb{N}$ associating to each symbol an *arity* and we write $a : n \to 1$ for a symbol a of arity n. A morphism of signatures is a function between the corresponding sets of symbols which preserves arity, and we write $\mathbf{Pol}_1^\times$ for the corresponding category. There is a forgetful functor $\mathbf{Law}_1 \to \mathbf{Pol}_1^\times$, sending a theory $\mathcal{T}$ on the set $\bigsqcup_{n \in \mathbb{N}} \mathcal{T}(n,1)$ with first projection as arity. This functor admits a left adjoint $-^* : \mathbf{Pol}_1^\times \to \mathbf{Law}_1$. Given a signature S_1, and $n \in \mathbb{N}$, $\mathsf{S}_1^*(n,1)$ is the set of *terms* formed using operations, with variables in $\{x_1^n, x_2^n, \ldots, x_n^n\}$ (the superscript is necessary to unambiguously recover the type of a variable, i.e. $x_i^n : n \to 1$, but for simplicity we will often omit it in the following). More generally, a morphism in $\mathsf{S}_1^*(n,m)$ is an m-uple $\langle t_1, \ldots, t_m \rangle$ of terms with variables in $\{x_1^n, \ldots, x_n^n\}$, which can be thought of as a formal *substitution*, and composition is given by parallel substitution:

$$\langle u_1, \ldots, u_k \rangle \circ \langle t_1, \ldots, t_m \rangle = \langle u_1[t_1/x_1, \ldots, t_n/x_n], \ldots, u_k[t_1/x_1, \ldots, t_m/x_m] \rangle$$

(the term on the right is obtained by substituting each occurrence of a variable x_i in a term u_j by t_i). By abuse of notation, we write S_1^* for the set of substitutions and $s_0, t_0 : \mathsf{S}_1^* \to \mathbb{N}$ for the source and target maps.

A *term rewriting system*, or TRS, consists of a signature S_1 together with a set S_2 of *rewriting rules* and functions $s_1, t_1 : \mathsf{S}_2 \to \mathsf{S}_1^*$ which indicate the source and target of each rewriting rule, and are supposed to satisfy $s_0 \circ s_1 = s_0 \circ t_1$ and $t_0 \circ s_1 = t_0 \circ t_1 = 1$ (i.e. the source and target of a rewriting rule are parallel terms). We sometimes write $\rho : t \Rightarrow u$ for a rule ρ with t as source and u as target. A *context* C of arity n is a term with variables in $\{x_1, \ldots, x_n, \square\}$ where the variable $\square$ is a particular variable, the *hole*, occurring exactly once. Given a term t of arity n, we write $C[t]$ for the term obtained from C by replacing $\square$ by t. The composition of contexts C and D is given by substitution $D \circ C = D[C]$. A *bicontext* is a pair (C, f) consisting of a context C and a substitution f. A *rewriting step* $C[\rho \circ f]$ of arity n is a triple consisting of a rewriting rule $\rho : t \Rightarrow u$, with t and u of arity k, together with a hole C of arity n, as well as a substitution $f : n \to k$ in S_1^*: a rewriting step can thus be thought of as a rewriting rule in a bicontext. Its source is the term $C[t \circ f]$ and its target is the term $C[u \circ f]$. We write $\mathsf{S}_2^{[]}$ for the set of rewriting steps. As in Section 2.3, we write S_2^* for the set of rewriting paths, which consist of composable sequence of rewriting steps, and $\mathsf{S}_2^{\sim}$ for the set of rewriting zig-zags in a TRS, and use associated notations. Every term rewriting system S freely generates a 2-Lawvere theory, with S_1^* as 1-cells and $\mathsf{S}_2^{\sim}$ as 2-cells. Given a rewriting step $C[\rho \circ f]$, a context D and a substitution g of suitable types, we have $D[C[\rho \circ f] \circ g] = (D \circ C)[\rho \circ (f \circ g)]$ so that bicontexts act on rewriting steps, and this action extends to rewriting paths and zig-zags by functoriality, i.e. $C[(p \cdot q) \circ f] = C[p \circ f] \cdot C[q \circ f]$.

An *extended term rewriting system*, or 2-TRS, consists of a term rewriting system as above, together with a set S_3 of *coherence relations* and functions $s_2, t_2 : \mathsf{S}_3 \to \mathsf{S}_2^{\sim}$, indicating their source and target, such that $s_1 \circ s_2 = s_1 \circ t_2$ and $t_1 \circ s_2 = t_1 \circ t_2$. The Lawvere 2-theory *presented* by a 2-TRS S is the $(2,1)$-category noted $\overline{\mathsf{S}}$, with $\mathbb{N}$ as 0-cells, S_1^* as 1-cells and, as 2-cells the quotient of $\mathsf{S}_2^{\sim}$ under the smallest congruence identifying the source and target of any elements of S_3.

▶ **Example 19.** The extended rewriting system Mon for monoids has symbols and rules

$$\mathsf{Mon}_1 = \{m : 2 \to 1, e : 0 \to 1\}$$
$$\mathsf{Mon}_2 = \{\alpha : m(m(x_1, x_2), x_3) \Rightarrow m(x_1, m(x_2, x_3)), \lambda : m(e, x_1) \Rightarrow x_1, \rho : m(x_1, e) \Rightarrow x_1\}$$

There are coherence relations A, B, C, D and E, respectively corresponding to a confluence for the five critical branchings of the rewriting system, whose 0-sources are

$$m(m(m(x_1, x_2), x_3), x_4) \qquad m(m(e, x_1), x_2) \qquad m(m(x_1, e), x_2) \qquad m(m(x_1, x_2), e) \qquad m(e, e)$$

Those coherence relations can be pictured as follows:

$$\begin{tikzcd}
m(m(m(x_1,x_2),x_3),x_4) \arrow[r, "\alpha"] \arrow[dd, "\alpha"'] & m(m(x_1,m(x_2,x_3)),x_4) \arrow[dr, "\alpha"] & \\
 & \overset{A}{\Rightarrow} & m(x_1,m(m(x_2,x_3),x_4)) \arrow[d, "\alpha"] \\
m(m(x_1,x_2),m(x_3,x_4)) \arrow[rr, "\alpha"'] & & m(x_1,m(x_2,m(x_3,x_4)))
\end{tikzcd}
\qquad
\begin{tikzcd}
m(m(e,x_1),x_2) \arrow[rr, "\alpha"] \arrow[dr, "\lambda"'] & \overset{B}{\Rightarrow} & m(e,m(x_1,x_2)) \arrow[dl, "\lambda"] \\
 & m(x_1,x_2) &
\end{tikzcd}$$

$$\begin{tikzcd}
m(m(x_1,e),x_2) \arrow[rr, "\alpha"] \arrow[dr, "\rho"'] & \overset{C}{\Rightarrow} & m(x_1,m(e,x_2)) \arrow[dl, "\lambda"] \\
 & m(x_1,x_2) &
\end{tikzcd}
\quad
\begin{tikzcd}
m(m(x_1,x_2),e) \arrow[rr, "\alpha"] \arrow[dr, "\rho"'] & \overset{D}{\Rightarrow} & m(x_1,m(x_2,e)) \arrow[dl, "\rho"] \\
 & m(x_1,x_2) &
\end{tikzcd}
\quad
\begin{tikzcd}
m(e,e) \arrow[d, bend right, "\lambda"'] \arrow[d, bend left, "\rho"] \arrow[d, phantom, "\overset{E}{\Rightarrow}"] \\
te
\end{tikzcd}$$

For concision, for each arrow, we did not indicate the proper rewriting step, but only the rewriting rule of the rewriting step (hopefully, the reader will easily be able to reconstruct it). For instance, the coherence relation C has target $\alpha(x_1, e, x_2) \cdot m(x_1, \lambda(x_2))$.

3.4 Rewriting properties

Suppose fixed a 2-TRS S together with $W \subseteq \mathsf{S}_2$. The 2-TRS S induces an 2-ARS in each hom-set: this point of view will allow reusing the work done on 2-ARS on Section 2.

▶ **Definition 20.** *Given a 2-TRS S and $m, n \in \mathbb{N}$, we write $\mathsf{S}(m, n)$ for the 2-ARS with $\mathsf{S}(m, n)_0 = \mathsf{S}_1^*(n, m)$, $\mathsf{S}(m, n)_1 = \mathsf{S}_2^{[]}$ and $\mathsf{S}(m, n)_2 = \mathsf{S}_3$.*

Similarly, a set W induces a set $W(m,n) \subseteq \mathsf{S}(m,n)_1 = \mathsf{S}_2^{\square}$, where $W(m,n)$ is the set of rewriting steps whose rewriting rule belongs to W (they are of the form $C[\alpha \circ f]$ with $\alpha \in W$). We say that a 2-TRS S is *W-terminating* / *locally W-confluent* / *W-confluent* / *W-coherent* when each $\mathsf{S}(m,n)$ is with respect to $W(m,n)$. We say that S is *confluent* when it is W-confluent for $W = \mathsf{S}_2$ (and similarly for other properties).

A *W-branching* (α_1, α_2) is a pair of rewriting steps $\alpha_1 : t \Rightarrow u_1$ and $\alpha_2 : t \Rightarrow u_2$ in $\mathcal{W}^{\square}$ with the same source:

$$u_1 \overset{\alpha_1}{\Longleftarrow} t \overset{\alpha_2}{\Longrightarrow} u_2$$

It is *W-confluent* when there are cofinal rewriting paths $\pi_1 : u_1 \Rightarrow v$ and $\pi_2 : u_2 \Rightarrow v$ in W^* such that $\overline{\alpha_1 \cdot \pi_1} = \overline{\alpha_2 \cdot \pi_2}$, which is depicted on the left.

$$\begin{tikzcd}
& t \arrow[dl, Rightarrow, "\alpha_1"'] \arrow[dr, Rightarrow, "\alpha_2"] & \\
u_1 \arrow[dr, Rightarrow, dotted, "\pi_1"'] & & u_2 \arrow[dl, Rightarrow, dotted, "\pi_2"] \\
& v &
\end{tikzcd}$$

$$\begin{tikzcd}
& C[t \circ f] \arrow[dl, Rightarrow, "C[\alpha_1 \circ f]"'] \arrow[dr, Rightarrow, "C[\alpha_2 \circ f]"] & \\
C[u_1 \circ f] \arrow[dr, Rightarrow, dotted, "C[\pi_1 \circ f]"'] & & C[u_2 \circ f] \arrow[dl, Rightarrow, dotted, "C[\pi_2 \circ f]"] \\
& C[v \circ f] &
\end{tikzcd}$$

By extension of Proposition 11, we have

▶ **Proposition 21.** *If S is W-terminating and locally W-confluent then it is W-confluent.*

In practice, termination can be shown as follows [3, Section 5.2]. A *reduction order* $>$ is a well-founded partial order on terms in S_1^* which is compatible with context extension: given terms $t, u \in \mathsf{S}_1^*$, $t > u$ implies $C[t \circ f] > C[u \circ f]$ for every context C and substitution $f \in \mathsf{S}_1^*$ (whose types are such that the expressions make sense).

▶ **Proposition 22.** *A 2-TRS S equipped with a reduction order such that $t > u$ for any rule $\alpha : t \Rightarrow u$ in W is W-terminating.*

In order to construct a reduction order one can use the following "interpretation method" [3, Section 5.3]. Suppose given a well-founded poset $(X, <)$ and an interpretation $[\![a]\!] : X^n \to X$ of each symbol $a \in \mathsf{S}_1$ of arity n as a function which is strictly decreasing in each argument. This induces, by composition and product, an interpretation $[\![t]\!]$ of every term. We define an order on functions $f, g : X^n \to X$ by $f \succ g$ whenever $f(x_1, \ldots, x_n) \succ g(x_1, \ldots, x_n)$ for every $x_i \in X$; this order is well-founded because the order on X is. By extension, we define an order on terms $t, u \in \mathsf{S}_1^*(n, 1)$ by $t \succ u$ whenever $[\![t]\!] \succ [\![u]\!]$: this order is always a reduction order. By Proposition 22, if we have $t \succ u$ for every rule $\alpha : t \Rightarrow u$ the 2-TRS is thus W-terminating.

▶ **Example 23.** Consider the 2-TRS Mon of Example 19. We interpret the symbols as $[\![m(x_1, x_2)]\!] = 2x_1 + x_2$ and $[\![e]\!] = 1$ on $X = \mathbb{N} \setminus 0$. All the rules are decreasing since we have

$$[\![m(m(x_1, x_2), x_3)]\!] = 4x_1 + 2x_2 + x_3 > 2x_1 + 2x_2 + x_3 = [\![m(x_1, m(x_2, x_3))]\!]$$

$$[\![m(e, x_1)]\!] = 2 + x_1 > x_1 = [\![x_1]\!] \qquad [\![m(x_1, e)]\!] = 2x_1 + 1 > x_1 = [\![x_1]\!]$$

and the rewriting system is terminating.

We now briefly recall the notion of *critical branching*, see [28] for a more detailed presentation. We say that a branching (α_1, α_2) is *smaller* than a branching (β_1, β_2) when the second can be obtained from the first by "extending the context", i.e. when there exists a context C and a morphism f of suitable types such that $\beta_i = C[\alpha_i \circ f]$ for $i = 1, 2$. In this case, the confluence of the first branching implies the confluence of the second one (see the diagram on the right above). The notion of context can be generalized to define the notion of a binary context C, with two holes, each of which occurs exactly once: we write $C[t,u]$ for the context where the holes have respectively been substituted with terms t and u. A branching is *non-overlapping* when it consists of two rewriting steps at disjoint positions, i.e. when it is of the form

$$C[u_1 \circ f_1, t_2 \circ f_2] \xLeftarrow{C[\alpha_1 \circ f_1, t_2 \circ f_2]} C[t_1 \circ f_1, t_2 \circ f_2] \xRightarrow{C[t_1 \circ f_1, \alpha_2 \circ f_2]} C[t_1 \circ f_1, u_2 \circ f_2]$$

for some binary context C, rewriting rules $\alpha_i : t_i \Rightarrow u_i$ in S_2 and morphisms f_i in S_1^* of suitable types. A branching is *critical* when it is not non-overlapping and minimal (wrt the above order). A TRS with a finite number of rewriting rules always have a finite number of critical branchings and those can be computed efficiently [3].

▶ **Lemma 24.** *A 2-TRS S is locally W-confluent when all its critical W-branchings are W-confluent.*

Proof. Suppose that all critical W-branchings are confluent. A non-overlapping W-branching is easily shown to be W-confluent. A non-minimal W-branching is greater than a minimal one, which is W-confluent by hypothesis, and is thus itself also W-confluent. ◀

We write $W_3 \subseteq \mathsf{S}_3$ for the set of coherence relations $A : \pi \Rightarrow \rho$ such that both π and ρ belong to $W^\sim$. As a useful particular case, we have the following variant of the Squier theorem:

▶ **Lemma 25.** *If 2-TRS S has a coherence relation in W_3 corresponding to a choice of confluence for every critical W-branching then it is locally W-confluent.*

▶ **Example 26.** The 2-TRS Mon of Example 19. By definition, every critical branching is confluent and Mon is thus locally confluent. From Example 23 and Proposition 21, we deduce that it is confluent.

As a direct adaptation of Proposition 13, we have

▶ **Lemma 27.** *If S is W-terminating and locally W-confluent then it is W-coherent.*

From Examples 23 and 26, we deduce that the 2-TRS Mon is coherent, thus showing the coherence property (C1) for monoidal categories.

Suppose given a W-convergent 2-TRS S. By Lemma 27, $\overline{\mathsf{S}}$ is W-coherent, by Theorem 18, the quotient functor $\overline{\mathsf{S}} \to \overline{\mathsf{S}}/W$ is a local equivalence, and by Proposition 15, $\overline{\mathsf{S}}/W$ is obtained from $\overline{\mathsf{P}}$ by restricting to 1-cells in normal form. Moreover, in good situations, we can provide a description of the quotient category $\overline{\mathsf{S}}/W$ by applying Proposition 16 hom-wise.

3.5 Algebras for Lawvere 2-theories

The notion of algebra for 2-theories was extensively studied by Yanofsky [35, 36], we refer to his work for details.

An *algebra* for a Lawvere 2-theory $\mathcal{T}$ is a 2-functor $C : \mathcal{T} \to \mathbf{Cat}$ which preserves products. By abuse of notation, we often write C instead of $C1$ and suppose that products are strictly preserved, so that $Cn = C^n$. A *pseudo-natural transformation* $F : C \Rightarrow D$ between algebras C and D consists in a functor $F : C \to D$ together with a family $\phi_f : Df \circ F^n \Rightarrow F \circ Cf$ of natural transformations indexed by 1-cells $f : n \to 1$ in $\mathcal{T}$, which is compatible with products, composition and 2-cells of $\mathcal{T}$. A *modification* $\mu : F \Rrightarrow G : C \Rightarrow D$ between two pseudo-natural transformations is a natural transformation $\mu : F \Rrightarrow G$ which is compatible with 2-cells of $\mathcal{T}$. We write $\mathrm{Alg}(\mathcal{T})$ for the 2-category of algebras, pseudo-natural transformations and modifications. We write $\mathrm{Alg}(\mathcal{T})$ for the category of algebras of a 2-theory $\mathcal{T}$.

▶ **Example 28.** Consider the 2-TRS Mon of Example 19. The category $\mathrm{Alg}(\overline{\mathsf{Mon}})$ of algebras of the presented 2-theory is isomorphic to the category **MonCat** of monoidal categories, strong monoidal functors and monoidal natural transformations. It might be surprising that Mon has five coherence relations whereas the traditional definition of monoidal categories only features two axioms (which correspond to the coherence relations A and C). There is no contradiction here: the commutation of two axioms can be shown to imply the one of the three other [18, 13].

A morphism $F : \mathcal{T} \to \mathcal{U}$ of 2-theories is a *biequivalence* when there is a morphism $G : \mathcal{U} \to \mathcal{T}$ and natural transformations $\eta : \mathrm{Id}_{\mathcal{T}} \Rightarrow G \circ f$ and $\varepsilon : F \circ G \Rightarrow \mathrm{Id}_{\mathcal{U}}$ whose components are equivalences. A generalization of Proposition 8 is shown in [36, Proposition 7]:

▶ **Proposition 29.** *Suppose given a morphism $F : \mathcal{T} \to \mathcal{T}'$ between theories. It is a biequivalence if and only if the functor $\mathrm{Alg}(F) : \mathrm{Alg}(\mathcal{T}') \to \mathrm{Alg}(\mathcal{T})$ induced by precomposition is a biequivalence (in a suitable sense).*

In particular, in the case where $\mathcal{W}$ is 2-rigid, it seems that we can deduce from Theorem 18 that the projection functor $\mathcal{T} \to \mathcal{T}/\mathcal{W}$ is a local equivalence, and thus a biequivalence, and thus that the categories $\mathrm{Alg}(\mathcal{T})$ and $\mathrm{Alg}(\mathcal{T}/\mathcal{W})$ are biequivalent (for instance, it is claimed that the categories of monoidal and strict monoidal categories are equivalent). However, the claim that any local equivalence is a biequivalence [36, Proposition 6] is wrong: given a local equivalence $F : \mathcal{C} \to \mathcal{D}$ between 2-categories, one can in general construct a pseudo-functor $G : \mathcal{D} \to \mathcal{C}$ satisfying suitable properties, but not a strict one, see [20, Example 3.1] for a counter-example. Intuitively, in the case where $\mathcal{C} = \mathcal{T}$ and $\mathcal{D} = \mathcal{T}/\mathcal{W}$ with rewriting properties as in Section 3.4, G will send a 1-cell to a normal form in its equivalence class, but the composite of two normal forms is not itself a normal form in general, we can only expect that it is isomorphic to a normal form.

We however conjecture that one can generalize the classical proof that any monoidal category is monoidally equivalent to a strict one [26, Theorem XI.3.1] to show the following general (C3) coherence theorem, as well as its (C4) generalization:

▶ **Conjecture 30.** *When $\mathcal{W}$ is 2-rigid, every $\mathcal{T}$-algebra is equivalent to a $\mathcal{T}/\mathcal{W}$ algebra.*

▶ **Conjecture 31.** *When $\mathcal{W}$ is 2-rigid, the 2-functor $\mathrm{Alg}(\mathcal{T}/\mathcal{W}) \to \mathrm{Alg}(\mathcal{T})$ induced by precomposition with the quotient functor $\mathcal{T} \to \mathcal{T}/W$ has a left adjoint such that the components of the unit are equivalences.*

This is left for future works. Note that, apart from informal explanations, we could not find a proof of Conjectures 30 and 31 for symmetric or braided monoidal categories in the literature, e.g. in [27, 17, 26] (in [17, Theorem 2.5] the result is only shown for free braided monoidal categories).

3.6 Symmetric monoidal categories

A *symmetric monoidal category* is a monoidal category equipped with a natural isomorphism $\gamma_{x,y} : x \otimes y \to y \otimes x$, called *symmetry*, satisfying three classical axioms. A symmetric monoidal category is *strict* when the structural isomorphisms α, λ and ρ are identities (but we do not require γ to be an identity). We write **SMonCat** (resp. $\mathbf{SMonCat}_{\mathrm{str}}$) for the category of symmetric monoidal categories (resp. strict ones). Using the same method as above, we can show the coherence theorems for symmetric monoidal categories [17]. This example illustrates more the interest of the previous developments since we are quotienting by a $(2,1)$-category $\mathcal{W}$ which is not the whole category.

We write SMon for the 2-TRS obtained from Mon (see Example 19) by adding a rewriting rule $\gamma : m(x_1, x_2) \Rightarrow m(x_2, x_1)$ (corresponding to symmetry), together with a coherence relation

$$\begin{tikzcd}
m(x_1,x_2) \arrow[r, "\gamma"] \arrow[d, equal] \arrow[dr, phantom, "\overset{F}{\Rrightarrow}"] & m(x_2,x_1) \arrow[d, "\gamma"] \\
m(x_1,x_2) \arrow[r, equal] & m(x_1,x_2)
\end{tikzcd}$$

as well as four relations corresponding to the critical branchings between the rule γ and the rules α, λ or ρ:

$$\begin{tikzcd}
m(m(x_1,x_2),x_3) \arrow[r, "\gamma"] \arrow[d, swap, "\alpha"] & m(x_3,m(x_1,x_2)) \arrow[r, "\alpha"] \arrow[d, phantom, "\overset{G}{\Rrightarrow}"] & m(m(x_3,x_1),x_2) \arrow[d, "\gamma"] \\
m(x_1,m(x_2,x_3)) \arrow[r, swap, "\gamma"] & m(m(x_2,x_3),x_1) \arrow[r, swap, "\alpha"] & m(x_2,m(x_3,x_1))
\end{tikzcd}
\qquad
\begin{tikzcd}
m(e,x_1) \arrow[rr, "\gamma"] \arrow[dr, swap, "\lambda"] & \arrow[d, phantom, "\overset{I}{\Rrightarrow}"] & m(x_1,e) \arrow[dl, "\rho"] \\
& x_1 &
\end{tikzcd}$$

$$\begin{tikzcd}
m(m(x_1,x_2),x_3) \arrow[rr, "\gamma"] \arrow[d, swap, "\alpha"] & \arrow[d, phantom, "\overset{H}{\Rrightarrow}"] & m(m(x_2,x_2),x_3) \arrow[d, "\gamma"] \\
m(x_1,m(x_2,x_3)) \arrow[r, swap, "\gamma"] & m(m(x_2,x_3),x_1) \arrow[r, swap, "\gamma"] & m(m(x_3,x_2),x_1)
\end{tikzcd}
\qquad
\begin{tikzcd}
m(x_1,e) \arrow[rr, "\gamma"] \arrow[dr, swap, "\rho"] & \arrow[d, phantom, "\overset{J}{\Rrightarrow}"] & m(e,x_1) \arrow[dl, "\lambda"] \\
& x_1 &
\end{tikzcd}$$

The category $\mathrm{Alg}(\overline{\mathsf{SMon}})$ is isomorphic to the category **SMonCat**. The traditional definition of symmetric monoidal categories only features axioms corresponding to F, G and I, but it can be shown that they implies the commutation of the axiom corresponding to H (by using G twice) and J (by using F and I). We write $W = \{\alpha, \lambda, \rho\}$. The category $\mathrm{Alg}(\mathsf{SMon}/W)$ is isomorphic to $\mathbf{SMonCat}_{\mathrm{str}}$. We have that the 2-TRS is W-terminating by Example 23 and W-locally confluent by definition (Example 19), it is thus W-coherent by Lemma 27. From Conjecture 30, we would deduce that any symmetric monoidal category is monoidally equivalent to a strict one.

Note that the above reasoning only depends on the convergence of the subsystem induced by W, i.e. on the fact that every diagram made of α, λ and ρ commutes, but it does not require anything on diagrams containing γ's. In particular, if we removed the compatibility relations G, H, I and J, the strictification theorem would still hold. The resulting notion of strict symmetric monoidal category would however be worrying since, for instance, in absence of I, the morphism $\gamma_{e,x_1} : m(e, x_1) \to m(x_1, e)$ would induce, in the quotient, a non-trivial automorphism $\gamma_{e,x_1} : x_1 \to x_1$ of each object x_1. The following variant of the coherence theorem is "stronger" in the sense that it requires these axioms to hold.

We have seen that for the theory of monoidal categories "every diagram commutes", in the sense that $\overline{\mathsf{Mon}}$ is a 2-rigid $(2, 1)$-category. For symmetric monoidal categories, we do not expect this to hold since γ_{x_1,x_1} and $\mathrm{id}_{m(x_1,x_1)}$ both are rewriting paths from $m(x_1, x_1)$ to itself, and are not equal in general (one can easily find an example of a symmetric monoidal category in which the symmetry is not the identity, this is in fact the case for most usual examples). It can however be shown that it holds for a subclass of 2-cells whose source and target are affine terms: a term is *affine* if no variable occurs twice. We now explain this, thus recovering a well-known property [27, Theorem 4.1] using rewriting techniques. Note that the property of being affine, as well as the variables occurring in terms, are preserved by rewriting steps. By inspection of critical branchings (Lemma 24), we have

▶ **Lemma 32.** *The 2-TRS SMon is locally confluent.*

It is not terminating, even when restricted to affine terms, since we for instance have the loop

$$m(x_1, x_2) \xrightarrow{\gamma(x_1,x_2)} m(x_2, x_1) \xrightarrow{\gamma(x_2,x_1)} m(x_1, x_2)$$

In order to circumvent this problem, we are going to formally "remove" the second morphism above and only keep instances of γ which tend to make variables in decreasing order. Note that the coherence relation F ensures that $\gamma(x_2, x_1) = \gamma(x_1, x_2)^-$ so that this removal does not change the presented $(2,1)$-category.

Given a term t, we write $\|t\|$ the list of variables occurring in it, from left to right, e.g. $\|m(m(x_2, e), x_1)\| = x_2 x_1$. We order variables by $x_i \succeq x_j$ whenever $i \leq j$ and extend it to lists of variables by lexicographic ordering (which is well-founded since we compare only words of the same length). Given a rewriting step $C[\gamma(t_1, t_2)]$ involving the rule γ is *decreasing* when $\|t_1\|\|t_2\| \succ \|t_2\|\|t_1\|$. Fix $n \in \mathbb{N}$, consider the 2-ARS $\mathsf{P}' = \mathsf{SMon}(n, 1)$, and write P for the 2-ARS obtained from P' by

- removing from P'_1 all non-decreasing rewriting steps involving γ,
- replacing in the source or target of a relation in P'_2 all non-decreasing steps $C[\gamma(t_1, t_2)]$ by $C[\gamma(t_2, t_1)^-]$.

▶ **Lemma 33.** *The 2-ARS* P *is locally confluent.*

Proof. The above reasoning shows that we have $\overline{\mathsf{P}}' = \overline{\mathsf{P}}$. Since SMon is locally confluent (Lemma 32), P' is locally confluent and thus also P. ◀

Since we restricted ourselves to decreasing symmetries, P is "almost" terminating since rewriting rules tend to put variables in decreasing order. However, it sill does not prevent loops when there is no variable: for instance, $\gamma(e, e)$ is a rewriting step from $m(e, e)$ to itself. Fortunately, we can always remove units by restricting to terms which are in normal form wrt λ and ρ. Namely, P satisfies the hypothesis of Proposition 16 with W consisting of all rewriting steps generated by λ and ρ (condition 3. uses the compatibility relations G, H, I and J). We can thus restrict to terms in which the unit e does not occur and for those the system is terminating. Any two rewriting paths between affine terms are equals and the algebras thus satisfy:

▶ **Proposition 34.** *In a symmetric monoidal category, every diagram whose* 0*-source is a tensor product of distinct objects commutes.*

4 Future works

We believe that the developed framework applies to a wide variety of algebraic structures, which will be explored in subsequent work. In fact, the full generality of the framework was not needed for (symmetric) monoidal categories, since the rules of the corresponding theory never need to duplicate or erase variables (and, in fact, those can be handled by traditional polygraphs [22, 13]). This is however, needed for the case of rig categories [24], which feature two monoidal structures $\oplus$ and $\otimes$, and natural isomorphisms such as $\delta_{x,y,z} : x \otimes (y \otimes z) \to (x \otimes y) \oplus (x \otimes z)$ (note that x occurs twice in the target), generalizing the laws for rings. Those were a motivating example for this work, and we will develop elsewhere a proof of coherence of those structures based on our rewriting framework, as well as related approaches on the subject [7, Appendix G].

A notion of Tietze transformation for term rewriting system, which are transformations allowing one to navigate between the various presentations of a given Lawvere theory, were given in [28]. It would be interesting to develop an analogous notion for 2-TRS, presenting a given Lawvere 2-theory: this would allow us to formalize reasoning about superfluous generators or relations (such as in Example 28).

Finally, the importance of the notion of polygraph can be explained by the fact that they are the cofibrant objects in a model structure on ω-categories [23]. It would be interesting to develop a similar point of view for higher term rewriting systems: a first step in this direction is the model structure developed in [36].

References

1 Coherence theorem for monoidal categories. URL: `https://ncatlab.org/nlab/show/coherence+theorem+for+monoidal+categories`.

2 Coherence theorem for symmetric monoidal categories. URL: `https://ncatlab.org/nlab/show/coherence+theorem+for+symmetric+monoidal+categories`.

3 Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge university press, 1999.

4 Marek A. Bednarczyk, Andrzej M. Borzyszkowski, and Wieslaw Pawlowski. Generalized congruences – Epimorphisms in Cat. *Theory and Applications of Categories*, 5(11):266–280, 1999.

5 Tibor Beke. Categorification, term rewriting and the Knuth-Bendix procedure. *Journal of Pure and Applied Algebra*, 215(5):728–740, 2011.

6 Marc Bezem, Jan Willem Klop, and Roel de Vrijer. *Term rewriting systems*. Cambridge University Press, 2003.

7 Filippo Bonchi, Alessandro Di Giorgio, and Alessio Santamaria. Tape diagrams for rig categories with finite biproducts. Technical report, Università di Pisa, 2022.

8 Albert Burroni. Higher-dimensional word problems with applications to equational logic. *Theoretical computer science*, 115(1):43–62, 1993.

9 Florence Clerc and Samuel Mimram. Presenting a category modulo a rewriting system. In *26th International Conference on Rewriting Techniques and Applications (RTA 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

10 Jonathan Asher Cohen. *Coherence for rewriting 2-theories*. PhD thesis, Australian National University, 2008. `arXiv:0904.0125`.

11 Benjamin Dupont and Philippe Malbos. Coherent confluence modulo relations and double groupoids. *Journal of Pure and Applied Algebra*, 226(10):107037, 2022. `arXiv:1810.08184`.

12 John W. Gray. 2-algebraic theories and triples. *Cahiers de Topologie et Géométrie Différentielle*, 14:178–180, 1973.

13 Yves Guiraud and Philippe Malbos. Coherence in monoidal track categories. *Mathematical Structures in Computer Science*, 22(6):931–969, 2012.

14 Yves Guiraud and Philippe Malbos. Polygraphs of finite derivation type. *Mathematical Structures in Computer Science*, 28(2):155–201, 2018.

15 Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM (JACM)*, 27(4):797–821, 1980.

16 Niles Johnson and Donald Yau. *2-dimensional categories*. Oxford University Press, USA, 2021.

17 André Joyal and Ross Street. Braided tensor categories. *Advances in Math*, 102:20–78, 1993.

18 Gregory Maxwell Kelly. On MacLane's conditions for coherence of natural associativities, commutativities, etc. *Journal of Algebra*, 1(4):397–402, 1964.

19 Nicolai Kraus and Jakob von Raumer. A rewriting coherence theorem with applications in homotopy type theory. *Mathematical Structures in Computer Science*, 32(7):982–1014, 2022. `arXiv:2107.01594`.

20 Stephen Lack. A Quillen model structure for 2-categories. *K-theory*, 26(2):171–205, 2002.

21 Yves Lafont. A new finiteness condition for monoids presented by complete rewriting systems (after Craig C. Squier). *Journal of Pure and Applied Algebra*, 98(3):229–244, 1995.

22 Yves Lafont. Towards an algebraic theory of boolean circuits. *Journal of Pure and Applied Algebra*, 184(2-3):257–310, 2003.

23 Yves Lafont, François Métayer, and Krzysztof Worytkiewicz. A folk model structure on omega-cat. *Advances in Mathematics*, 224(3):1183–1231, 2010.

24 Miguel L. Laplaza. Coherence for distributivity. In *Coherence in categories*, pages 29–65. Springer, 1972.

25 F. William Lawvere. Functorial semantics of algebraic theories. *Proceedings of the National Academy of Sciences*, 50(5):869–872, 1963.

26 Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5. Springer Science & Business Media, 1998.

27 Saunders MacLane. Natural associativity and commutativity. *Rice Institute Pamphlet-Rice University Studies*, 49(4), 1963.

28 Philippe Malbos and Samuel Mimram. Homological computations for term rewriting systems. In *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*, volume 52, pages 27–1, 2016.

29 Samuel Mimram and Pierre-Louis Curien. Coherent Presentations of Monoidal Categories. *Logical Methods in Computer Science*, 13, 2017.

30 John Power. An abstract formulation for rewrite systems. In *Category Theory and Computer Science*, pages 300–312, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg. doi:10.1007/BFb0018358.

31 John Power. Enriched Lawvere theories. *Theory and Applications of Categories*, 6(7):83–93, 1999.

32 Craig C. Squier, Friedrich Otto, and Yuji Kobayashi. A finiteness condition for rewriting systems. *Theoretical Computer Science*, 131(2):271–294, 1994.

33 Ross Street. Limits indexed by category-valued 2-functors. *Journal of Pure and Applied Algebra*, 8(2):149–181, 1976.

34 Ross Street. Categorical structures. *Handbook of algebra*, 1:529–577, 1996.

35 Noson S. Yanofsky. The syntax of coherence. *Cahiers de topologie et géométrie différentielle catégoriques*, 41(4):255–304, 2000.

36 Noson S. Yanofsky. Coherence, Homotopy and 2-Theories. *K-Theory*, 23(3):203–235, 2001.

Convolution Products on Double Categories and Categorification of Rule Algebras

Nicolas Behr
CNRS, Université Paris Cité, IRIF, France

Paul-André Melliès
CNRS, Université Paris Cité, INRIA, France

Noam Zeilberger
École Polytechnique, LIX, Palaiseau, France

Abstract

Motivated by compositional categorical rewriting theory, we introduce a convolution product over presheaves of double categories which generalizes the usual Day tensor product of presheaves of monoidal categories. One interesting aspect of the construction is that this convolution product is in general only oplax associative. For that reason, we identify several classes of double categories for which the convolution product is not just oplax associative, but fully associative. This includes in particular framed bicategories on the one hand, and double categories of compositional rewriting theories on the other. For the latter, we establish a formula which justifies the view that the convolution product categorifies the rule algebra product.

2012 ACM Subject Classification Theory of computation → Categorical semantics

Keywords and phrases Categorical rewriting, double pushout, sesqui-pushout, double categories, convolution product, presheaf categories, framed bicategories, opfibrations, rule algebra

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.17

1 Introduction

Our main motivation in this work is to categorify notions coming from *compositional rewriting theory* in the sense of [1–5, 8, 9] and more specifically the concepts of *rule algebra* [1, 3, 8, 9] and of *tracelet* [2, 6]. There, a *rewriting theory* is specified by a base category $\mathbf{C}$ together with a specific categorical description of *direct derivations*, defined as rewriting steps $s : X \rightharpoonup Y$ obtained by applying a rewriting rule $r : A \rightharpoonup B$ to a given object $X \in \mathbf{C}$ of the base category. Typical descriptions include *double-pushout (DPO)* [13] and *sesqui-pushout (SqPO)* [10] formalisms. A rewriting theory defined in this way is called *compositional* when it satisfies a technical property of two- and three-step derivation traces, ensuring that the two theorems below are satisfied:

- the *concurrency theorem* [1, 4, 5, 7, 13] states that every two-step derivation trace may be (essentially uniquely) characterized by a one-step trace (i.e., a direct derivation) along a *composite rule* capturing the causal interactions between the two rules,
- the *associativity theorem* [1, 4, 5, 7, 9] states that whenever the concurrency theorem is applied twice in order to convert a three-step trace into a one-step trace along a composite rule, either possible nesting order of two-step rule composition operations yields essentially the same one-step trace (i.e., up to universal isomorphisms).

One important benefit of compositionality is that every compositional rewriting theory gives rise to a *rule algebra* defined as a vector space $\mathcal{R}$ (over a suitable field $\mathbf{k}$ such as $\mathbf{k} = \mathbb{R}$) with a basis indexed by (isomorphism classes of) rules, and equipped with a bilinear product that maps a pair of basis elements to a sum over basis elements indexed by composite rules. More

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 17; pp. 17:1–17:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

explicitly, letting $\delta(r)$ denote the basis vector of $\mathcal{R}$ indexed by (the isomorphism class of) a rule r, writing $\cdot \star \cdot$ for the aforementioned binary product, $\mathcal{M}_r(s)$ for admissible matches of rule r into rule s, we have

$$\delta(r) \star \delta(s) = \sum_{\mu \in \mathcal{M}_r(s)} \delta(r_\mu s) \tag{1}$$

where $r_\mu s$ denotes one possible way to obtain a composite rule from r and s. Another natural idea when reasoning about compositional rewriting systems is to study *sets of direct derivations* as follows: we introduce a vector space $\mathcal{S}$ together with a notation $|X\rangle$ for a basis vector of $\mathcal{S}$ indexed by an (isomorphism class of an) object X of the underlying category. We then define the algebra morphism (or representation) $\rho : \mathcal{R} \to \mathit{Endo}_{\mathbf{k}}(\mathcal{S})$ as follows:

$$\rho(\delta(r))\,|X\rangle := \sum_{m \in M_r(X)} |r_m(X)\rangle\ , \tag{2}$$

where the right-hand side of the equation ranges over possible matches $m \in M_r(X)$ of the rule r into the object X, and where $|r_m(X)\rangle$ is the vector indexed by the isomorphism class of the outcome of applying r to X via m. The crucial fact that ρ satisfies the equation

$$\rho(\delta(r))\rho(\delta(s)) = \rho(\delta(r) \star \delta(s)) \tag{3}$$

and thus defines a *representation in $\mathcal{S}$ of the rule algebra* $(\mathcal{R}, \star)$ is far from trivial, and comes from a subtle interplay between the concurrency and the associativity theorems [1, 4, 5, 7, 9].

In the present paper, our primary purpose is to begin to *categorify* the rule algebra formalism, starting from the observation that the traditional frameworks for categorical rewriting (including the double-pushout and sesqui-pushout formalisms) can be neatly expressed using double categories. The idea is to associate to any such categorical rewriting framework a specific double category $\mathbb{D}$ whose objects are the objects of the original base category $\mathbf{C}$ and whose horizontal 1-cells $X \rightharpoonup Y$ are transformations typically defined as spans $X \leftarrow S \rightarrow Y$ in $\mathbf{C}$, defined in such a way that they include both the rewriting rules $r : A \rightharpoonup B$ as well as the derivation traces $s : X \rightharpoonup Y$ of the underlying rewriting theory. The double category $\mathbb{D}$ is then carefully designed in such a way that a direct derivation θ applying the rewriting rule $r : A \rightharpoonup B$ to define a rewriting step $s : X \rightharpoonup Y$ is the same thing as a double cell $\theta : r \to s$ of the form below, in the double category $\mathbb{D}$.

$$\begin{tikzcd} B \arrow[d, tail, "g"'] & & A \arrow[ll, harpoon', "r"'] \arrow[d, tail, "f"] \\ Y & & X \arrow[ll, harpoon, "s"] \arrow[from=1-2, to=2-2, Rightarrow, "\theta"] \end{tikzcd} \tag{4}$$

Here, the vertical maps f and g of the double category $\mathbb{D}$ indicate how the source A and the target B of the rewriting rule $r : A \rightharpoonup B$ are “embedded” in the objects X and Y, respectively, in order to define the direct derivation $\theta : r \to s$ exhibiting $s : X \rightharpoonup Y$ as an instance of the rewriting rule r. Given a rewriting rule $r : A \rightharpoonup B$ and a horizontal 1-cell $s : X \rightharpoonup Y$, it makes sense to look at the set $\hat{\Delta}_r(s)$ of double cells $\theta : r \to s$ of the form (4), which describes all the possible embeddings $f : A \rightarrowtail X$ and $g : Y \rightarrowtail B$ and all the possible ways $\theta : r \to s$ the horizontal 1-cell $s : X \rightharpoonup Y$ can be seen as an instance of the rewriting rule r. An important observation is that $\hat{\Delta}_r$ defines a covariant presheaf $\hat{\Delta}_r : \mathbb{D}_1 \to \mathsf{Set}$ over the vertical cell category $\mathbb{D}_1$ whose objects are horizontal 1-cells, and whose morphisms $\varphi : s \to s'$ are double cells of the form

$$\begin{tikzcd} Y \arrow[d, tail, "h_Y"'] & & X \arrow[ll, harpoon', "s"'] \arrow[d, tail, "h_X"] \\ Y' & & X" \arrow[ll, harpoon, "s'"] \arrow[from=1-2, to=2-2, Rightarrow, "\varphi"] \end{tikzcd} \tag{5}$$

Note that the covariant presheaf $\hat{\Delta}_r : \mathbb{D}_1 \to \mathsf{Set}$ is the representable presheaf $\hat{\Delta}_r = \mathbb{D}_1(r, -)$ which associates to every 1-cell $s : X \rightharpoonup Y$ the set $\hat{\Delta}_r(s) = \mathbb{D}_1(r,-)(s)$ of all morphisms (4) from r to s in the category $\mathbb{D}_1$. One main intuition guiding us in the process of categorification is that the representable presheaf $\hat{\Delta}_r : \mathbb{D}_1 \to \mathsf{Set}$ should play the role of the basis vector $\delta(r)$ of rule algebra $(\mathcal{R}, \star)$.

This fundamental intuition brought us to develop a larger picture, and to associate to any double category $\mathbb{D}$ the category $\widehat{\mathbb{D}}$ of its vertical presheaves, simply defined as the category $\widehat{\mathbb{D}} = [\mathbb{D}_1, \mathsf{Set}]$ of covariant presheaves $G, F : \mathbb{D}_1 \to \mathsf{Set}$ over the vertical cell category $\mathbb{D}_1$ with natural transformations $G \Rightarrow F$ between them. One main contribution of the paper is the discovery of a convolution product $* : \widehat{\mathbb{D}} \times \widehat{\mathbb{D}} \to \widehat{\mathbb{D}}$ which generalizes the usual Day convolution product of presheaves over a monoidal category, and is of interest in its own right. In particular, we explain in §3 that, somewhat unexpectedly, the convolution product is only *oplax associative* in general. We then examine in the paper a number of additional *fibrational properties* of the double category $\mathbb{D}$ in order to recover strong associativity. We establish in §4 that strong associativity is guaranteed for framed bicategories and then study in §5 how the story unfolds for the case of double categories coming from rewriting frameworks. Finally, we establish in §6 that the convolution product $\hat{\Delta}_{r_2} * \hat{\Delta}_{r_1}$ of two representable presheaves $\hat{\Delta}_{r_1}$ and $\hat{\Delta}_{r_2}$ can in certain situations be decomposed as a finite sum of representables, categorifying equation (1).

2 Double categories

Throughout this paper, we consider double categories as weakly internal categories in CAT [14, Ch. 12.3]. This means that a (weak) *double category* $\mathbb{D}$ consists of a pair of categories $\mathbb{D}_0$ and $\mathbb{D}_1$ and a collection of functors

$$S, T : \mathbb{D}_1 \longrightarrow \mathbb{D}_0\,, \qquad U : \mathbb{D}_0 \longrightarrow \mathbb{D}_1\,, \qquad \diamond_h : \mathbb{D}_1 \times_{\mathbb{D}_0} \mathbb{D}_1 \longrightarrow \mathbb{D}_1\,, \tag{6}$$

(where $\mathbb{D}_1 \times_{\mathbb{D}_0} \mathbb{D}_1$ denotes the pullback of S and T) making the diagrams

$$\begin{tikzcd}
\mathbb{D}_1 \arrow[d, "T"'] & \mathbb{D}_1 \times_{\mathbb{D}_0} \mathbb{D}_1 \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\diamond_h"'] & \mathbb{D}_1 \arrow[d, "S"] \\
\mathbb{D}_0 & \mathbb{D}_1 \arrow[l, "T"] \arrow[r, "S"'] & \mathbb{D}_0
\end{tikzcd}
\qquad
\begin{tikzcd}
& \mathbb{D}_0 \arrow[dl, equal] \arrow[dr, equal] \arrow[d, "U"'] & \\
\mathbb{D}_0 & \mathbb{D}_1 \arrow[l, "T"] \arrow[r, "S"'] & \mathbb{D}_0
\end{tikzcd}$$

commute, together with natural isomorphisms $(r \diamond_h s) \diamond_h t \xrightarrow{\sim} r \diamond_h (s \diamond_h t)$ and $U \diamond_h r \xrightarrow{\sim} r \xrightarrow{\sim} r \diamond_h U$ expressing associativity and neutrality of the structure up to isomorphism, and satisfying a number of coherence axioms.

We refer to the objects of $\mathbb{D}_0$ as 0-*cells*, to the morphisms of $\mathbb{D}_0$ as *vertical 1-cells*, to the objects of $\mathbb{D}_1$ as *horizontal 1-cells*, and to the morphisms of $\mathbb{D}_1$ as *double cells.* We employ a slightly non-standard convention in writing horizontal 1-cells from right to left, using $\leftharpoonup$ arrows, and we reserve the arrow type $\rightarrowtail$ for vertical 1-cells. With these conventions, horizontal composition, denoted $\diamond_h$, reads as follows:

$$\begin{tikzcd}
Z \arrow[d, tail, "h"'] \arrow[dr, phantom, "\overset{\Vert}{\underset{\Downarrow}{\beta}}"] & Y \arrow[l, harpoon', "s"'] \arrow[d, tail, "g"] & & Y \arrow[d, tail, "g"'] \arrow[dr, phantom, "\overset{\Vert}{\underset{\Downarrow}{\alpha}}"] & X \arrow[l, harpoon', "r"'] \arrow[d, tail, "f"] & & Z \arrow[d, tail, "h"'] \arrow[dr, phantom, "\overset{\Vert}{\underset{\Downarrow}{\beta \diamond_h \alpha}}"] & X \arrow[l, harpoon', "s \diamond_h r"'] \arrow[d, tail, "f"] \\
Z' & Y' \arrow[l, harpoon, "s'"] & \diamond_h & Y' & X' \arrow[l, harpoon, "r'"] & = & Z' & X' \arrow[l, harpoon, "s' \diamond_h r'"]
\end{tikzcd}$$

We emphasize that horizontal composition is only associative up to isomorphism. On the other hand, *vertical composition*, denoted $\diamond_v$, is a strictly associative operation, corresponding to composition of morphisms in the category $\mathbb{D}_1$ and of their images along the functors S and T in the category $\mathbb{D}_0$.

▶ **Example 2.1.** A prototypical example of a double category is the *double category of spans* $\mathbb{S}\mathsf{pan}(\mathbf{C})$ in some category $\mathbf{C}$ with chosen pullbacks, where:

- 0-cells and vertical 1-cells of $\mathbb{S}\mathsf{pan}(\mathbf{C})$ are given by objects and morphisms of $\mathbf{C}$;
- horizontal 1-cells $Y \leftharpoonup X$ are given by spans $Y \leftarrow Z \rightarrow X$ in $\mathbf{C}$;
- double cells are given by morphisms of spans in the sense of a pair of commuting squares

$$\begin{tikzcd}
Y \arrow[d, tail] & X \arrow[l, harpoon'] \arrow[d, tail] \arrow[dl, phantom, "\Downarrow"] & {=} & Y \arrow[d, tail] & Z \arrow[l] \arrow[r] \arrow[d, tail] & X \arrow[d, tail] \\
Y' & X' \arrow[l, harpoon'] & & Y' & Z' \arrow[l] \arrow[r] & X'
\end{tikzcd}$$

- horizontal composition of spans is defined by pullback, with unit $U_X = X \overset{id_X}{\leftarrow} X \overset{id_X}{\rightarrow} X$.

Observe that horizontal composition in $\mathbb{S}\mathsf{pan}(\mathbf{C})$ is indeed only associative up to isomorphism. For that reason, the notion of double category we have just introduced is sometimes called weak double category.

▶ **Example 2.2.** In order to describe term rewriting in the language of double categories, we consider the *double category* $\mathbb{T}\mathsf{RS}[\Sigma]$ associated to a fixed signature Σ of operations, defined as follows:

- 0-cells are lists of terms $\mathbf{t} = t_1, \dots, t_n$ over Σ with set of variables denoted $\mathsf{Var}(\mathbf{t})$;
- vertical 1-cells $\mathbf{t} \rightarrowtail \mathbf{u}$ represent subterm matchings, given by a pair $(C \mid \sigma)$ of a multi-hole context C and a substitution σ such that $\mathbf{u} = C[\mathbf{t}\sigma]$;
- there is a unique horizontal 1-cell $\mathbf{t} \rightharpoonup \mathbf{t}'$ for every pair of lists of terms of the same length $|\mathbf{t}| = |\mathbf{t}'|$ such that $\mathsf{Var}(\mathbf{t}') \subseteq \mathsf{Var}(\mathbf{t})$;
- double cells

$$\begin{tikzcd}
\mathbf{t}' \arrow[d, tail] & \mathbf{t} \arrow[l, harpoon'] \arrow[d, tail] \arrow[dl, phantom, "\Downarrow"] \\
\mathbf{u}' & \mathbf{u} \arrow[l, harpoon']
\end{tikzcd}$$

are given by a pair $(C \mid \sigma)$ of a multi-hole context C and a substitution σ such that $\mathbf{u} = C[\mathbf{t}\sigma]$ and $\mathbf{u}' = C[\mathbf{t}'\sigma]$.

The idea is that the horizontal 1-cells of $\mathbb{T}\mathsf{RS}[\Sigma]$ describe all possible shapes of (potentially parallel) rewriting rules, and double cells close those rules under context extension and substitution.

Any bicategory may be seen as a double category in which all vertical 1-cells are identities, i.e., such that $\mathbb{D}_0$ is a discrete category. (As a special case, any monoidal category may be seen as a double category with $\mathbb{D}_0 = 1$ the terminal category.) Conversely, every double category $\mathbb{D}$ has an underlying *horizontal bicategory* $\mathbb{D}_\bullet$, defined as the double category with the same 0-cells and horizontal 1-cells as $\mathbb{D}$, but restricted to double cells whose vertical components are identities (i.e., morphisms α in $\mathbb{D}_1$ such that $S(\alpha)$ and $T(\alpha)$ are identity morphisms), also said to be *globular*.

It will be convenient to also consider an "unbiased" (in the sense of [16, §3.1]) definition of double category, starting from a pair of categories $\mathbb{D}_0$ and $\mathbb{D}_1$ and a family of functors $(h_n : \mathbb{D}_n \to \mathbb{D}_1)_{n \geqslant 0}$ where $\mathbb{D}_n := \underbrace{\mathbb{D}_1 \times_{\mathbb{D}_0} \dots \times_{\mathbb{D}_0} \mathbb{D}_1}_{n \text{ times}}$ is the limit in Set of the "zig-zag" diagram of functors:

$$\begin{tikzcd}
& \mathbb{D}_1 \arrow[dl, "T"'] \arrow[dr, "S"] & & & & \mathbb{D}_1 \arrow[dl, "T"'] \arrow[dr, "S"] & \\
\mathbb{D}_0 & & \mathbb{D}_0 & \cdots & \mathbb{D}_0 & & \mathbb{D}_0
\end{tikzcd}$$

where the category $\mathbb{D}_1$ appears n times and $\mathbb{D}_0$ appears $n+1$ times. The objects (respectively morphisms) of $\mathbb{D}_n$ may be seen as sequences $(s_n, \dots, s_1)$ of n composable horizontal 1-cells (resp. double cells), with the functor

$$h_n \;=\; (s_n, \dots, s_1) \;\mapsto\; h_n(s_n, \dots, s_1) \in \mathsf{obj}(\mathbb{D}_1)$$

performing the horizontal composition “all at once”. In this *presentation*, both associativity and neutrality are represented by a single family of natural isomorphisms

$$h_n \circ (h_{i_1}, \dots, h_{i_n}) \quad \cong \quad h_{i_1 + \dots + i_n}$$

satisfying a number of coherence axioms.

We will go back and forth between the biased and unbiased definitions of a double category, which are equivalent. In particular, given a double category with unit U and (binary) horizontal composition $\diamond_h$, we can obtain a family of n-ary composition functors h_n by taking $h_0 = U$, $h_1 = id$, and h_n to be any bracketing of $n-1$ $\diamond_h$'s for $n \geqslant 2$.

3 A convolution product of presheaves over double categories

3.1 Presheaves over double categories and the convolution product

One starting point for our work was the observation that the Day convolution product on presheaves over monoidal categories [11] may be extended to a convolution product for *vertical presheaves* over double categories. As explained in the introduction, a *vertical presheaf over a double category* $\mathbb{D}$ is simply defined as a covariant Set-valued presheaf over the category $\mathbb{D}_1$ whose objects are the horizontal 1-cells and whose morphisms are the double cells of $\mathbb{D}$. We write $\widehat{\mathbb{D}} = [\mathbb{D}_1, \mathsf{Set}]$ for the category of vertical presheaves over $\mathbb{D}$ and natural transformations between them. A vertical presheaf F over a double category $\mathbb{D}$ thus assigns a set $F(r)$ to every horizontal 1-cell $r : Y \leftharpoonup X$, and a function $F(\alpha) : F(r) \to F(r')$ to every double cell $\alpha : r \to r'$. As also explained in the introduction, an important example is provided by *representable presheaves*, which we notate $\hat{\Delta}_r := \mathbb{D}_1(r, -)$.

▶ **Example 3.1.** Term rewriting systems may be modeled as vertical presheaves over the double category $\mathbb{T}\mathsf{RS}[\Sigma]$. For example, suppose Σ contains a binary operation m and a constant e, and consider the rewriting rule $r : m(e, x) \rightharpoonup x$. The presheaf $\hat{\Delta}_r$ in a sense encapsulates all ways of applying r once to a subterm. For instance, $\hat{\Delta}_r(r' : m(e, m(e, x)) \rightharpoonup m(e, x))$ contains exactly two elements, corresponding to the double cells $\alpha_1 : r \to r'$ and $\alpha_2 : r \to r'$ defined by the context/substitution pairs $(C_1 = - \mid \sigma_1 = m(e, x)/x)$ and $(C_2 = m(e, -) \mid \sigma_2 = x/x)$ respectively.

At this stage, we explain how we extend the Day convolution product to double categories. We find it instructive to begin by recalling the usual definition of the Day convolution product on presheaves over a category C equipped with a monoidal product $\otimes : \mathsf{C} \times \mathsf{C} \to \mathsf{C}$. Given a pair of presheaves G and F over C, the convolution product $G * F$ is defined as the left Kan extension of the presheaf over the product category $\mathsf{C} \times \mathsf{C}$

$$\begin{tikzcd} \mathsf{C} \times \mathsf{C} \arrow[r, "G \times F"] & \mathsf{Set} \times \mathsf{Set} \arrow[r, "\times"] & \mathsf{Set} \end{tikzcd}$$

along the monoidal product functor $\otimes$:

$$\begin{tikzcd} \mathsf{C} \times \mathsf{C} \arrow[r, "G \times F"] \arrow[dr, "\otimes"'] & \mathsf{Set} \times \mathsf{Set} \arrow[r, "\times"] \arrow[d, phantom, "\Downarrow"] & \mathsf{Set} \\ & \mathsf{C} \arrow[ur, "G * F"'] & \end{tikzcd}$$

Equivalently, $G * F$ may be defined by the following well-known coend formula:

$$G * F = a \mapsto \int^{(c,b) \in \mathbf{C} \times \mathbf{C}} \mathbf{C}(c \otimes b, a) \times G(c) \times F(b) \tag{7}$$

Recall that in general, the coend $\int^{x \in \mathsf{C}} M(x,x)$ of a functor $M : \mathsf{C}^{op} \times \mathsf{C} \to \mathsf{Set}$ may be explicitly computed as a quotient of the coproduct $\coprod_{x \in \mathsf{C}} M(x,x)$ modulo an equivalence relation induced by the co- and contravariant actions of M [15, IX.6].

This definition of convolution product based on a left Kan extension and a coend formula can be adapted to double categories in the following way. Given two presheaves G and F over the category $\mathbb{D}_1$, the *convolution product* $G * F$ is the presheaf over the category $\mathbb{D}_1$ defined as the left Kan extension of the presheaf

$$\begin{tikzcd}
\mathbb{D}_1 \times_{\mathbb{D}_0} \mathbb{D}_1 \arrow[r, "proj"] & \mathbb{D}_1 \times \mathbb{D}_1 \arrow[r, "G \times F"] & \mathsf{Set} \times \mathsf{Set} \arrow[r, "\times"] & \mathsf{Set}
\end{tikzcd}$$

along the horizontal composition functor $\diamond_h$:

$$\begin{tikzcd}
\mathbb{D}_1 \times_{\mathbb{D}_0} \mathbb{D}_1 \arrow[r, "proj"] \arrow[drr, "\diamond_h"'] & \mathbb{D}_1 \times \mathbb{D}_1 \arrow[rr, "G \times F"] & {} \arrow[d, Rightarrow] & \mathsf{Set} \times \mathsf{Set} \arrow[r, "\times"] & \mathsf{Set} \\
 & & \mathbb{D}_1 \arrow[urr, "G * F"'] & &
\end{tikzcd}$$

As in the case of the convolution product for monoidal categories, the left Kan extension can be also neatly expressed as a coend formula:

$$G * F = r \mapsto \int^{(s_2, s_1) \in \mathbb{D}_2} \mathbb{D}_1(s_2 \diamond_h s_1, r) \times G(s_2) \times F(s_1) \tag{8}$$

As in the case of the Day convolution product, it follows from the definition that

▶ **Proposition 3.2.** *The convolution product* $* : \widehat{\mathbb{D}} \times \widehat{\mathbb{D}} \to \widehat{\mathbb{D}}$ *preserves colimits component-wise.*

Before proceeding further, let us consider an example from term rewriting that illustrates the motivation for the definition of the convolution product.

▶ **Example 3.3.** Let $r : m(e,x) \rightharpoonup x$ as in Example 3.1, and consider the convolution product $\hat{\Delta}_r * \hat{\Delta}_r$ of $\hat{\Delta}_r$ with itself. Intuitively, this presheaf encapsulates all ways of applying the rewriting rule r twice, possibly in parallel. One can verify that $\hat{\Delta}_r * \hat{\Delta}_r$ decomposes as a sum of representables:

$$\hat{\Delta}_r * \hat{\Delta}_r \cong \hat{\Delta}_{r_1} + \hat{\Delta}_{r_1} + \hat{\Delta}_{r_2} + \hat{\Delta}_{r_3} \tag{9}$$

where

$$r_1 : m(e, m(e,x)) \rightharpoonup x\,, \quad r_2 : m(m(e,e),x) \rightharpoonup x\,, \quad r_3 : m(e,x), m(e,y) \rightharpoonup x, y\,.$$

The convolution product therefore allows us to express neatly in algebraic form by the formula (9) the fact that there are four canonical ways of applying the rule $r : m(e,x) \rightharpoonup x$ twice, corresponding to the two ways of deriving the r_1 rule and the unique derivation of r_2, as well as the r_3 rule corresponding to two parallel applications of r. In Section 6 we will give a more general analysis of this phenomenon.

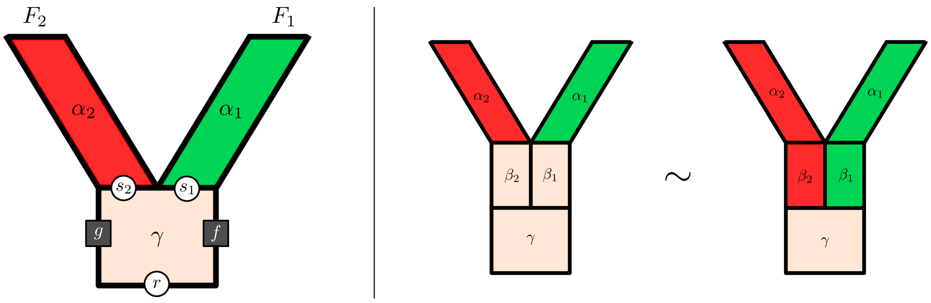


Figure 1 Left: diagram illustrating an element of the convolution product $F_2 * F_1$ evaluated at a horizontal cell r, where $\alpha_1 \in F_1(s_1)$ and $\alpha_2 \in F_2(s_2)$, and $\gamma : s_2 \diamond_h s_1 \to r$. Right: equivalence relation on diagrams induced by the coend formula.

3.2 Oplax associativity of the convolution product

The binary convolution product naturally generalizes to an n-ary convolution product of presheaves, defined in terms of the functors $h_n : \mathbb{D}_n \to \mathbb{D}_1$ discussed in Section 2.

▶ **Definition 3.4.** *Let $F_1, \ldots, F_n : \mathbb{D}_1 \to \mathsf{Set}$ be an n-tuple of covariant presheaves over $\mathbb{D}_1$. We define their* convolution product *by the coend formula*

$$F_n * \ldots * F_1 = r \mapsto \int^{(s_n,\ldots,s_1)\in\mathbb{D}_n} \mathbb{D}_1(h_n(s_n,\ldots,s_1),r) \times F_n(s_n) \times \cdots \times F_1(s_n) \qquad (10)$$

with the understanding that the formula specializes to $r \mapsto \int^{X\in\mathbb{D}_0} \mathbb{D}_1(U_X, r)$ in the nullary case $n = 0$. Equivalently, $F_n * \ldots * F_1$ is defined by the following left Kan extension diagram:

$$\begin{tikzcd}
\mathbb{D}_n \arrow[r, "proj"] \arrow[drr, "h_n"'] & \mathbb{D}_1^n \arrow[rr, "F_n\times\cdots\times F_1"] & {} \arrow[d, Rightarrow] & \mathsf{Set}^n \arrow[r, "\times"] & \mathsf{Set} \\
& & \mathbb{D}_1 \arrow[urr, "F_n*\ldots*F_1"'] & &
\end{tikzcd} \qquad (11)$$

For convenience, we write $\circledast_n : \widehat{\mathbb{D}}^n \to \widehat{\mathbb{D}}$ for the resulting n-ary convolution product, and we also sometimes write $\bar{U}$ for the nullary case $\bar{U} = \circledast_0$.

We find it evocative to visualize the elements of the convolution product $F_n * \ldots * F_1$ evaluated at a horizontal 1-cell r by a kind of "rabbit diagram", as illustrated on the left side of Figure 1 for the case $n = 2$. In the diagram, α_1 and α_2 represent elements of $F_1(s_1)$ and $F_2(s_2)$ respectively, where s_1 and s_2 are arbitrary horizontal 1-cells, and γ represents a double cell $s_2 \diamond_h s_1 \to r$ with $S(\gamma) = f$ and $T(\gamma) = g$. (We will sometimes omit the labels of the various cells in diagrams when they are unimportant.) On the right side of the figure, we also depict the equivalence relation on tuples

$$(\gamma \diamond_v (\beta_1 \diamond_h \beta_2), \alpha_2, \alpha_1) \sim (\gamma, F_2(\beta_2)(\alpha_1), F_1(\beta_1)(\alpha_1)) \qquad (12)$$

that is forced by the coend formula (8).

Perhaps surprisingly, it turns out that in general convolution only defines an *oplax* monoidal product on the presheaf category $\widehat{\mathbb{D}}$.

▶ **Theorem 3.5.** *The convolution product on the category $\widehat{\mathbb{D}}$ of vertical presheaves is oplax associative and oplax unital in the sense that there exists a family of natural transformations*

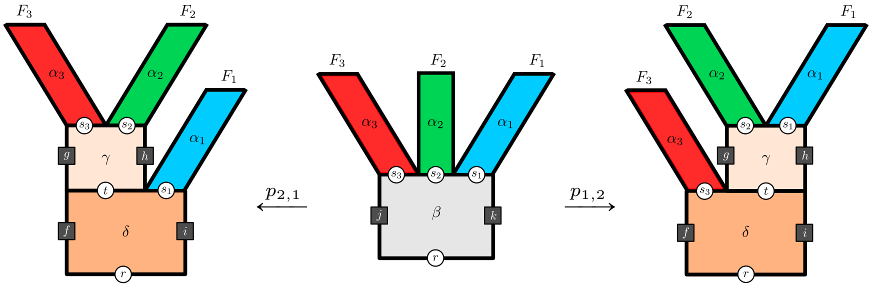
$$\begin{tikzcd}
\widehat{\mathbb{D}}^{n_1+\cdots+n_k} \arrow[rr, "*_{n_1+\cdots+n_k}"] \arrow[dr, "*_{n_1}\times\cdots\times *_{n_k}"'] & \arrow[d, phantom, "\Downarrow p_{n_1,\ldots,n_k}"] & \widehat{\mathbb{D}} \\
& \widehat{\mathbb{D}}^k \arrow[ur, "*_k"'] &
\end{tikzcd}$$

satisfying the coherence laws of an oplax monoidal product. In other words, $\widehat{\mathbb{D}}$ is an oplax monoidal category under the convolution product functors $*_n : \widehat{\mathbb{D}}^n \to \widehat{\mathbb{D}}$.

An illustrative example is given by the natural transformations

$$(F_3 * F_2) * F_1 \xleftarrow{p_{2,1}} F_3 * F_2 * F_1 \xrightarrow{p_{1,2}} F_3 * (F_2 * F_1)$$

for any triple (F_3, F_2, F_1) of presheaves of $\widehat{\mathbb{D}}$. To understand these natural transformations, let us consider the following diagrams, which depict generic elements of $(F_3 * F_2) * F_1$, $F_3 * F_2 * F_1$, and $F_3 * (F_2 * F_1)$ evaluated at a horizontal 1-cell r:



(13)

Here s_1, s_2, s_3, t are horizontal 1-cells, β, γ, and δ are double cells, and each α_i is an element of $F_i(s_i)$, while $f, \ldots, k$ are vertical 1-cells corresponding to the projections of the respective double cells $f = T(\delta), g = T(\gamma), h = S(\gamma), i = S(\delta), j = T(\beta), k = S(\beta)$. The diagram in the middle of (13) may be seen as a degenerate case of the ones on the outside, in the sense that it corresponds to taking γ to be the identity double cell (on $s_3 \diamond_h s_2$ and $s_2 \diamond_h s_1$ respectively) and taking $\delta = \beta$. This simple observation yields the natural transformations $p_{2,1}$ and $p_{1,2}$. On the other hand, these natural transformations need not be invertible in general for an arbitrary double category, since the diagrams on the outside of (13) cannot necessarily be transformed into the one in the middle, in particular if the vertical 1-cells g and h are non-trivial.

One special case where the oplaxity maps are easily seen to be invertible is when the underlying vertical category $\mathbb{D}_0$ is discrete, i.e., when $\mathbb{D}$ is a bicategory.

▶ **Definition 3.6.** *We say the convolution product on $\widehat{\mathbb{D}}$ is* strongly associative *if the natural transformations $p_{n_1,\ldots,n_k}$ of Theorem 3.5 are invertible for all $n_1, \ldots, n_k \geqslant 1$. We say that it is strongly associative and unital if the $p_{n1,\ldots,n_k}$ are invertible for all $n_1, \ldots, n_k \geqslant 0$.*

▶ **Proposition 3.7.** *If $\mathbb{D}_0$ is a discrete category then the convolution product on $\widehat{\mathbb{D}}$ is strongly associative and unital.*

Proposition 3.7 covers in particular the case of the usual Day convolution product on presheaves over a monoidal category, seen as a double category over the terminal category $\mathbb{D}_0 = 1$. In Section 5 we will establish sufficient conditions under which the convolution product is strongly associative and unital. First, though, let us take a bit of time to consider a well-known class of double categories for which it turns out that the convolution product is strongly associative, but not strongly unital.

4 Non-unital associativity in a framed bicategory

A special situation in which the convolution product becomes strongly associative is when the horizontal 1-cells of the underlying double category may be "pushed" along vertical 1-cells independently with respect to their source and target, while leaving the other end fixed. Such situations are captured precisely by the notion of framed bicategory [18].

▶ **Definition 4.1** (Shulman [18, Definition 4.2]). *A double category $\mathbb{D}$ is said to be a* framed bicategory *if the pairing of the source and target functors $(T, S) : \mathbb{D}_1 \to \mathbb{D}_0 \times \mathbb{D}_0$ is an opfibration (or if it satisfies any of the equivalent conditions of [18, Theorem 4.1]).*

The double category $\mathbb{S}\mathsf{pan} = \mathbb{S}\mathsf{pan}(\mathsf{Set})$ is an example of a framed bicategory. Indeed, the pushforward of a span of sets $Y \xleftarrow{a} Z \xrightarrow{b} X$ along a pair of functions $(g, h) : (Y, X) \to (Y', X')$ is given by composing the legs of the span with the two functions $Y' \xleftarrow{g} Y \xleftarrow{a} Z \xrightarrow{b} X \xrightarrow{h} X'$.

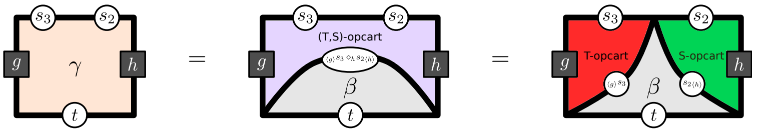
One important property of framed bicategories is that pushing forward along a pair of vertical cells may be decomposed into a pair of pushforward operations with respect to the source and the target. Given a horizontal 1-cell $r : Y \leftharpoondown X$ and a pair of vertical 1-cells $g : Y \to Y'$ and $h : X \to X'$, we thus write ${}_{\langle g \rangle}r_{\langle h \rangle} : Y' \leftharpoondown X'$ for the pushforward of r along (g, h), which may be equivalently read as $({}_{\langle g \rangle}r)_{\langle h \rangle}$ (push r along g relative to T and then along h relative to S) or as ${}_{\langle g \rangle}(r_{\langle h \rangle})$ (push r along h relative to S and then along g relative to T). In particular, the pushforward operations are compatible with horizontal composition, in the following sense.

▶ **Proposition 4.2** ([18, Corollary 4.3]). *In a framed bicategory, ${}_{\langle g \rangle}(t \diamond_h s)_{\langle h \rangle} \cong ({}_{\langle g \rangle}t) \diamond_h (s_{\langle h \rangle})$.*

This property is crucial in the proof of associativity of the convolution product for presheaves on framed bicategories.

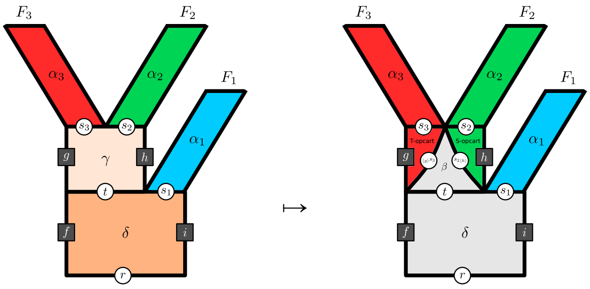
▶ **Theorem 4.3.** *If $\mathbb{D}$ is a framed bicategory then the convolution product on $\widehat{\mathbb{D}}$ is strongly associative.*

Proof sketch. The idea is summarized in the following series of diagrams:



(14)

Here, γ is the double cell appearing on the left side of (13), in the depiction of a generic element of $(F_3 * F_2) * F_1$. In the middle, by pushing $s_3 \diamond_h s_2$ forward along (g, h), we have factored γ as an op-Cartesian double cell followed by a globular cell β. On the right side of (14), by applying Proposition 4.2, we have factored this op-Cartesian double cell as the horizontal composition of a pair of op-Cartesian "triangles" (i.e., double cells with one side being an identity vertical cell). Finally, using this factorization, we can turn a generic element of $(F_3 * F_2) * F_1$ into an element of $F_3 * F_2 * F_1$:



(15)

Note that here we rely on the covariant action of the presheaves F_3 and F_2 to extend the elements $\alpha_3 \in F(s_3)$ and $\alpha_2 \in F(s_2)$ by the respective op-Cartesian triangles to obtain elements of $F_3({}_{\langle g \rangle}s_3)$ and $F_2(s_{2\langle h \rangle})$. It is routine to verify that the transformation (15) defines an inverse to the natural transformation $p_{2,1} : F_3 * F_2 * F_1 \to (F_3 * F_2) * F_1$: the equation $q_{2,1} \circ p_{2,1} = id$ is trivial, while the equation $p_{2,1} \circ q_{2,1} = id$ holds because the two sides of (15) are equivalent modulo equation (12) (cf. right side of Figure 1). This argument generalizes easily to inverting $p_{n_1,\ldots,n_k}$ for any $n_1, \ldots, n_k \geqslant 1$. ◀

However, the proof of the invertibility of the natural transformations $p_{n_1,\ldots,n_k}$ does *not* extend to arbitrary sequences of non-negative integers $n_1, \ldots, n_k \geqslant 0$, and indeed in general the convolution product on presheaves over framed bicategories is not strongly unital. Before demonstrating this, we first state an easy observation about representability of the nullary convolution product in the presence of an initial object.

▶ **Proposition 4.4.** *If $\mathbb{D}_0$ has an initial object 0, then $\bar{U} \cong \hat{\Delta}_{U_0}$.*

▶ **Example 4.5** (Counterexample to unitality of convolution over framed bicategories)**.** Consider the framed bicategory $\mathbb{S}\mathsf{pan}$, and let us write $\mathsf{Span} = \mathbb{S}\mathsf{pan}_1$ for its underlying category of spans and double cells between them. Since Set has an initial object given by the empty set $\varnothing$, the nullary convolution is representable (Prop. 4.4) by $U_\varnothing = \varnothing \leftarrow \varnothing \to \varnothing$. Note that the functor $U : \mathsf{Set} \to \mathsf{Span}$ has a right adjoint given by the functor sending a span $Y \leftarrow Z \to X$ to its underlying carrier Z, which implies (and indeed it is easy to verify) that $\varnothing \leftarrow \varnothing \to \varnothing$ is itself an initial object in Span. This entails that $\bar{U}$ is isomorphic to the terminal presheaf on Span.

Now, let $F = \hat{\Delta}_{U_1}$ be the presheaf represented by the identity span over the one-element set. By the aforementioned adjunction, for an arbitrary span $r = Y \leftarrow Z \to X$, elements of $F(r) \cong \mathsf{Span}(U_1, r)$ are in bijection with elements of Z, that is $F(r) \cong Z$. In particular, $F(r)$ is empty if Z is empty. On the other hand, since $U_Y \diamond_h r \xrightarrow{\sim} r$ and since $\bar{U}(r) \cong \{*\}$, every element of $F(U_Y) \cong Y$ induces an element of $(F * \bar{U})(r)$. So $(F * \bar{U})(r)$ is non-empty if Y is non-empty. But this implies that there is no natural transformation $q_{1,0} : F * \bar{U} \to F$, and hence the oplax unitor $p_{1,0} : F \to F * \bar{U}$ is not invertible.

5 Associativity from the positive cylindrical decomposition property

5.1 Cylindrical Decomposition Property

At this point, it must be stressed that our motivating examples of double categories coming from rewriting theory are *not* framed bicategories. For many of these examples, each of the source and target functors $S, T : \mathbb{D}_1 \to \mathbb{D}_0$ separately has some kind of opfibrational structure (or multi-opfibrational structure, see [5]), yet the pairing $(T, S) : \mathbb{D}_1 \to \mathbb{D}_0 \times \mathbb{D}_0$ is typically not an opfibration, because pushing one end of a horizontal 1-cell along a vertical 1-cell will not leave the other end fixed. Nevertheless, we will see that such double categories admit a strongly associative convolution product.

As an illustration, consider the *double category* $\mathbb{D}\mathsf{PO} = \mathbb{D}\mathsf{PO}(\mathsf{Set})$ defined as a sub-double category of $\mathbb{S}\mathsf{pan} = \mathbb{S}\mathsf{pan}(\mathsf{Set})$ with the same 0-cells and horizontal 1-cells, but restricting vertical 1-cells to injections, and restricting double cells to pairs of pushout squares:

$$\begin{tikzcd}
Y \arrow[d, tail] & X \arrow[l, harpoon'] \arrow[d, tail] \\
Y' & X' \arrow[l, harpoon']
\arrow[from=1-1, to=2-2, phantom, "\Downarrow"]
\end{tikzcd}
=
\begin{tikzcd}
Y \arrow[d, tail] & Z \arrow[l] \arrow[r] \arrow[d, tail] \arrow[dl, phantom, very near end, "\urcorner"] \arrow[dr, phantom, very near end, "\ulcorner"] & X \arrow[d, tail] \\
Y' & Z' \arrow[l] \arrow[r] & X'
\end{tikzcd}$$

($\mathbb{D}\mathsf{PO}$ and similar double categories play a role in DPO-rewriting, see [5].) Now suppose that we want to push a span of the form $Y \leftarrow \varnothing \rightarrow X$ along a pair of injective functions $Y \rightarrow Y'$ and $X \rightarrow X'$:

$$\begin{tikzcd}
Y \arrow[d, tail] & \varnothing \arrow[l] \arrow[r] & X \arrow[d, tail] \\
Y' & & X'
\end{tikzcd}$$

Such a horn may be completed to a double cell in $\mathbb{D}\mathsf{PO}$ just in case there exists a set Z' such that $Y' \cong Y + Z'$ and $X' \cong X + Z'$. But it is easy to construct examples for which no such set exists, for instance taking $|X| = |Y| = 1$, $|Y'| = 2$, and $|X'| = 3$. So the functor $(T, S) : \mathsf{DPO} \rightarrow \mathsf{Inj} \times \mathsf{Inj}$ is not an opfibration.

However, despite $\mathbb{D}\mathsf{PO}$ not being a framed bicategory, let us observe that it enjoys a similar factorization property to the one used in the proof of Theorem 4.3, and which is sufficient for proving associativity.

▶ **Definition 5.1.** *We say that a double category $\mathbb{D}$ has the n-*cylindrical decomposition property *if for every globular cell $\rho : h_n(r_n, \dots, r_1) \rightarrow r$ and for every double cell $\varphi : r \rightarrow s$ there exists a family of n double cells $(\varphi_n, \dots, \varphi_1) : (r_n, \dots, r_1) \rightarrow (s_n, \dots, s_1)$ in $\mathbb{D}_n$ and a globular cell $\sigma : h_n(s_n, \dots, s_1) \rightarrow s$ such that*

$$\varphi \diamond_v \rho \quad = \quad \sigma \diamond_v h_n(\varphi_n, \dots, \varphi_1) \tag{16}$$

which means that the double cell $\varphi \diamond_v \rho$ factors as the vertical composition $\sigma \diamond_v h_n(\varphi_n, \dots, \varphi_1)$ of the globular cell σ after the horizontal composition $h_n(\varphi_n, \dots, \varphi_1)$, as depicted below:

$$\begin{tikzcd}
\cdot \arrow[r, "r_n"] \arrow[d, equal] & \cdot & \cdots \arrow[d, phantom, "\Downarrow \rho"] & \cdot \arrow[r, "r_1"] & \cdot \arrow[d, equal] \\
\cdot \arrow[rrrr, "r"] \arrow[d, tail] & & \arrow[d, phantom, "\Downarrow \varphi"] & & \cdot \arrow[d, tail] \\
\cdot \arrow[rrrr, "s"'] & & & & \cdot
\end{tikzcd}
=
\begin{tikzcd}
\cdot \arrow[r, "r_n"] \arrow[d, tail] \arrow[dr, phantom, "\Downarrow \varphi_n"] & \cdot \arrow[d, tail] & \cdots & \cdot \arrow[r, "r_1"] \arrow[d, tail] \arrow[dr, phantom, "\Downarrow \varphi_1"] & \cdot \arrow[d, tail] \\
\cdot \arrow[r, "s_n"'] \arrow[d, equal] & \cdot & \cdots \arrow[d, phantom, "\Downarrow \sigma"] & \cdot \arrow[r, "s_1"'] & \cdot \arrow[d, equal] \\
\cdot \arrow[rrrr, "s"'] & & & & \cdot
\end{tikzcd}$$

Moreover, the family of n cells $(\varphi_n, \dots, \varphi_1)$ and the globular cell σ are universal, in the sense that for every family of n double cells $(\chi_n, \dots, \chi_1) : (r_n, \dots, r_1) \rightarrow (t_n, \dots, t_1)$ in $\mathbb{D}_n$, for every globular cell $\tau : h_n(t_n, \dots, t_1) \rightarrow t$ and for every double cell $\psi : s \rightarrow t$ such that the equation

$$\psi \diamond_v \varphi \diamond_v \rho \quad = \quad \tau \diamond_v h_n(\chi_n, \dots, \chi_1) \tag{17}$$

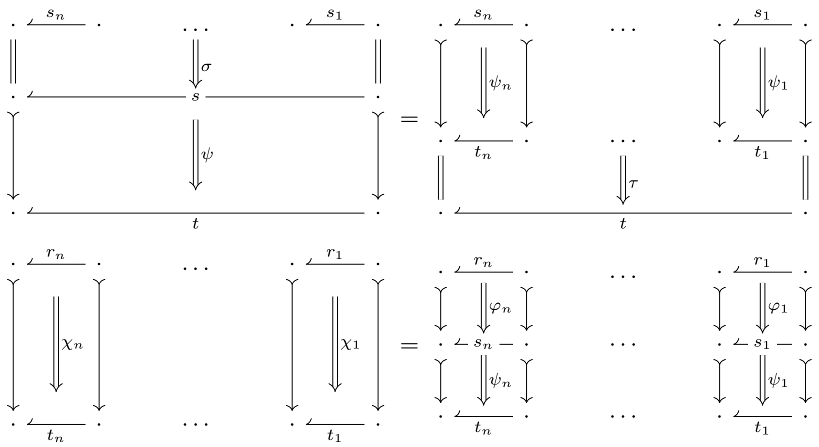
holds, as depicted below

$$\begin{tikzcd}
\cdot \arrow[r, "r_n"] \arrow[d, equal] & \cdot & \cdots \arrow[d, phantom, "\Downarrow \rho"] & \cdot \arrow[r, "r_1"] & \cdot \arrow[d, equal] \\
\cdot \arrow[rrrr, "r"] \arrow[d, tail] & & \arrow[d, phantom, "\Downarrow \varphi"] & & \cdot \arrow[d, tail] \\
\cdot \arrow[rrrr, "s"] \arrow[d, tail] & & \arrow[d, phantom, "\Downarrow \psi"] & & \cdot \arrow[d, tail] \\
\cdot \arrow[rrrr, "t"'] & & & & \cdot
\end{tikzcd}
=
\begin{tikzcd}
\cdot \arrow[r, "r_n"] \arrow[d, tail] \arrow[dr, phantom, "\Downarrow \chi_n"] & \cdot \arrow[d, tail] & \cdots & \cdot \arrow[r, "r_1"] \arrow[d, tail] \arrow[dr, phantom, "\Downarrow \chi_1"] & \cdot \arrow[d, tail] \\
\cdot \arrow[r, "t_n"'] \arrow[d, equal] & \cdot & \cdots \arrow[d, phantom, "\Downarrow \tau"] & \cdot \arrow[r, "t_1"'] & \cdot \arrow[d, equal] \\
\cdot \arrow[rrrr, "t"'] & & & & \cdot
\end{tikzcd}$$

there exists a unique family of n double cells $(\psi_n,\ldots,\psi_1):(s_n,\ldots,s_1)\to(t_n,\ldots,t_1)$ in $\mathbb{D}_n$ such that the two equations

$$\begin{aligned}\psi\diamond_v\sigma&=\tau\diamond_v h_n(\psi_n,\ldots,\psi_1)\\(\chi_n,\ldots,\chi_1)&=(\psi_n,\ldots,\psi_1)\diamond_v(\varphi_n,\ldots,\varphi_1)\end{aligned}\qquad(18)$$

are satisfied, as depicted below:



(19)

We say that $\mathbb{D}$ has the cylindrical decomposition property *(or CDP) if it has the n-CDP for all $n\geqslant 0$, and the* positive CDP *if it has the n-CDP for all $n\geqslant 1$.*

▶ **Example 5.2.** Any framed bicategory has the positive CDP.

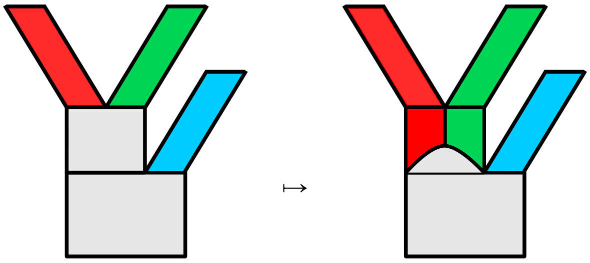
▶ **Example 5.3** ([5], Prop. 6.5)**.** $\mathbb{D}$PO has the positive CDP.

▶ **Example 5.4.** $\mathbb{T}$RS$[\Sigma]$ has the n-CDP for all $n\geqslant 0$, in the even stronger sense that any double cell $\varphi:h_n(r_n,\ldots,r_1)\to s$ factors uniquely as a horizontal composition $\varphi=h_n(\alpha_n,\ldots,\alpha_1)$.

One can establish that

▶ **Theorem 5.5.** *If $\mathbb{D}$ has the positive CDP then the convolution product on $\widehat{\mathbb{D}}$ is strongly associative. If it also has the 0-CDP then convolution is strongly associative and unital.*

Momentarily putting aside the full motivation for the universality condition in Definition 5.1 (which will become clearer in Section 5.2 below), we can already give an intuitive explanation for why the positive CDP entails strong associativity. Indeed, the factorization (16) applied to the trivial globular cell $\gamma:s_3\circ s_2\to s_3\circ s_2$ generalizes equation (14), allowing us to reuse essentially the same procedure to invert the natural transformations $p_{n_1,\ldots,n_k}$ for any positive $n_1,\ldots,n_k\geqslant 1$. For example, we can define an inverse $q_{2,1}:(F_3*F_2)*F_1\to F_3*F_2*F_1$ to $p_{2,1}$ in an analogous way to the map (15) we used in proving strong associativity of the convolution product over framed bicategories:



(20)

Once again, the reason $q_{2,1}$ defines an inverse to $p_{2,1}$ boils down to the fact that the two sides of (20) are equated by the coend formula defining the convolution product (recall Figure 1). A similar argument can also be used to establish strong unitality when the underlying double category $\mathbb{D}$ also has the 0-CDP.

Note however that neither example of $\mathbb{S}\mathsf{pan}$ nor $\mathbb{D}\mathsf{PO}$ has the 0-CDP. It turns out nevertheless that convolution over the latter is strongly unital, as we will briefly address in Section 5.4.

5.2 Relative opfibrations

Here we explain how the cylindrical decomposition property can be reformulated in fibrational terms, for a natural common generalization of the notions of Grothendieck opfibration and of Street opfibration.

Let $F : \mathsf{E} \to \mathsf{B}$ be a functor, and let $\mathsf{G} \xrightarrow{\iota} \mathsf{B}$ be a wide subcategory of B (i.e., ι is bijective on objects). We define a new category $F[\mathsf{G}]$ as the category whose objects are given by triples (e, b, γ) of an object $e \in \mathsf{E}$, an object $b \in \mathsf{B}$, and an arrow $\gamma : Fe \twoheadrightarrow b$ in G, and whose morphisms

$$(e, b, \gamma) \xrightarrow{(\varepsilon,\beta)} (e', b', \gamma')$$

are given by pairs of morphisms $\varepsilon : e \to e'$ and $\beta : b \to b'$ such that $\beta \circ \gamma = \gamma' \circ F\varepsilon$. This category comes equipped with evident forgetful functors $\pi_{\mathsf{E}} : F[\mathsf{G}] \to \mathsf{E}$ and $\pi_{\mathsf{B}} : F[\mathsf{G}] \to \mathsf{B}$. Moreover, there is a functor $in_{\mathsf{E}} : \mathsf{E} \to F[\mathsf{G}]$ defined on objects by $in_{\mathsf{E}}(e) = (e, Fe, id_{Fe})$, in such a way that $F = \pi_{\mathsf{B}} \circ in_{\mathsf{E}}$. We remark that the functor in_{E} is a left adjoint to the functor π_{E}, since

$$\begin{aligned} Hom_{F[\mathsf{G}]}(in_{\mathsf{E}}(e), (e', b', \gamma')) &= Hom_{F[\mathsf{G}]}((e, Fe, id_{Fe}), (e', b', \gamma')) \\ &= \{(\varepsilon, \beta) \mid \varepsilon : e \to e', \beta : Fe \to b', \beta \circ id_{Fe} = \gamma' \circ F\varepsilon\} \qquad (21) \\ &\cong Hom_{\mathsf{E}}(e, e') = Hom_{\mathsf{E}}(e, \pi_{\mathsf{E}}((e', b', \gamma')))\,. \end{aligned}$$

The construction generalizes to every wide subcategory $\mathsf{G} \xrightarrow{\iota} \mathsf{B}$ the construction of the free opfibration $\pi_{\mathsf{B}} : F[\mathsf{B}] \to \mathsf{B}$ associated to a functor $F : \mathsf{E} \to \mathsf{B}$, which one recovers when G is the category B itself. This leads us to the following definition:

▶ **Definition 5.6.** *Let $F : \mathsf{E} \to \mathsf{B}$ be a functor, and let G be a wide subcategory of B. We say that F is a* G-relative opfibration *if $\pi_{\mathsf{B}} : F[\mathsf{G}] \to \mathsf{B}$ is a Grothendieck opfibration.*

▶ **Example 5.7.** The definition above subsumes three standard notions of functor $F : \mathsf{E} \to \mathsf{B}$:

- When $\mathsf{G} = |\mathsf{B}|$ is the discrete wide subcategory of B, a G-relative opfibration is the same thing as a Grothendieck opfibration $F : \mathsf{E} \to \mathsf{B}$,
- When $\mathsf{G} = \mathsf{core}(\mathsf{B})$ is the wide subcategory of reversible maps in B, a G-relative opfibration is the same thing as a Street opfibration $F : \mathsf{E} \to \mathsf{B}$,
- When $\mathsf{G} = \mathsf{B}$ is the category B itself, a G-relative opfibration $F : \mathsf{E} \to \mathsf{B}$ is the same thing a general functor $F : \mathsf{E} \to \mathsf{B}$.

Moreover, one can readily verify that the cylindrical decomposition property we introduced above (Def. 5.1) corresponds to the following particular instance of G-relative opfibration.

▶ **Proposition 5.8.** *A double category $\mathbb{D}$ satisfies the n-CDP precisely when the functor $h_n : \mathbb{D}_n \to \mathbb{D}_1$ is a* globular opfibration*, that is, a G-relative opfibration where $\mathsf{G} = \mathbb{D}_1^\bullet$ is the wide subcategory of $\mathbb{D}_1$ of globular double cells.*

The reason is that the category $h_n[\mathsf{G}]$ can be neatly described as the category of *cylindric maps* whose objects the tuples $(s_1, \ldots, s_n, s, \sigma)$ where $\sigma : h_n(s_n, \ldots, s_1) \to s$ is a globular double cell, and whose morphisms

$$(\psi_n, \ldots, \psi_1, \psi) \quad : \quad (s_n, \ldots, s_1, s, \sigma) \longrightarrow (t_n, \ldots, t_1, t, \tau)$$

consist of a family of n double cells $(\psi_n, \ldots, \psi_1) : (s_n, \ldots, s_1) \to (t_n, \ldots, t_1)$ in $\mathbb{D}_n$ and of a double cell $\psi : s \to t$ in $\mathbb{D}_1$ satisfying the equation

$$\psi \diamond_v \sigma \quad = \quad \tau \diamond_v h_n(\psi_n, \ldots, \psi_1).$$

depicted in (19).

5.3 Kan extensions along relative opfibrations

We now state a basic result on left Kan extensions along G-relative opfibrations, analogous to a standard result about Kan extension along Grothendieck opfibrations that is extremely useful in practice:

▶ **Lemma 5.9** (cf. e.g. [17], Cor. 5.8)**.** *Let $F : \mathsf{E} \to \mathsf{B}$ be a Grothendieck opfibration between small categories, and let C be cocomplete and locally small. Then the (point-wise) left Kan extension of a functor $G : \mathsf{E} \to \mathsf{C}$ along F at $b \in \mathsf{obj}(\mathsf{B})$ can be computed as a colimit over the fiber $F^{-1}(b)$,*

$$\mathsf{Lan}_F G(b) \cong \underset{e \in F^{-1}(b)}{\mathsf{colim}} G(e)\,. \tag{22}$$

To develop a variant and generalization of this result for G-relative opfibrations, let us first recall a few standard constructions and facts from category theory.

▶ **Definition 5.10** ("Global" definition of left Kan extensions)**.** *Let $p : \mathsf{C} \to \mathsf{C}'$ be a functor, and D a category. If $p^* := - \circ p : [\mathsf{C}', \mathsf{D}] \to [\mathsf{C}, \mathsf{D}]$ has a left adjoint $p_! : [\mathsf{C}, \mathsf{D}] \to [\mathsf{C}', \mathsf{D}]$, i.e., if $p_! \dashv p^*$, then for all functors $F : \mathsf{C} \to \mathsf{D}$ the left Kan extension $\mathsf{Lan}_p F$ exists and is given by $\mathsf{Lan}_p F = p_! F$.*

▶ **Lemma 5.11.** *Let $L \dashv R$ be a pair of adjoint functors, with $L : \mathsf{C} \to \mathsf{C}'$ and $R : \mathsf{C}' \to \mathsf{C}$. Then for every category D, there is an induced pair of adjoint functors $- \circ R \dashv - \circ L : [\mathsf{C}', \mathsf{D}] \to [\mathsf{C}, \mathsf{D}]$. Therefore, for any functor $F : \mathsf{C} \to \mathsf{D}$, the left Kan extension $\mathsf{Lan}_L F$ of F along L is given by $\mathsf{Lan}_L F = F \circ R$.*

▶ **Theorem 5.12.** *Let $F : \mathsf{E} \to \mathsf{B}$ be a G-relative opfibration for some wide subcategory G of B, and let $P : \mathsf{E} \to \mathsf{Set}$ be a covariant presheaf (for small categories E and B). Then for every object b of B, we find that*

$$\mathsf{Lan}_F P(b) \cong \mathsf{Lan}_{\pi_{\mathsf{B}}} P \circ \pi_{\mathsf{E}}(b) \cong \underset{(e,b,\gamma) \in \pi_{\mathsf{B}}^{-1}(b)}{\mathsf{colim}} P(e) \cong \left(\coprod_{e \in \mathsf{E}} \mathsf{G}(\bar{F}(e), \bar{b}) \times P(e) \right) \diagup_{\sim_{\mathsf{G}}}\,. \tag{23}$$

Here, we used convenient shorthand notations $\bar{F}(e) := \iota^{-1} \circ F(e)$ and $\bar{b} := \iota^{-1}(b)$, and the equivalence relation $\sim_{\mathsf{G}}$ is defined as

$$\begin{aligned} (e, (\gamma, p)) \sim_{\mathsf{G}} (e', (\gamma', p')) :\Leftrightarrow \exists e - \varepsilon \to e' \in \mathsf{E},\ (\delta, q) \in \mathsf{G}(\bar{F}(e'), \bar{b}) \times P(e) : \\ (\gamma, p) = (\delta \circ \iota^{-1} \circ F(\varepsilon) \circ \iota, q) \ \wedge \ (\gamma', p') = (\delta, P(\varepsilon) q)\,. \end{aligned} \tag{24}$$

Proof. Recall that F factorizes uniquely as $F = \pi_{\mathsf{B}} \circ in_{\mathsf{E}}$, and that $in_{\mathsf{E}} \dashv \pi_{\mathsf{E}}$. Let us then compute $\mathsf{Lan}_F P$ step-wise via $\mathsf{Lan}_F P = \mathsf{Lan}_{\pi_{\mathsf{B}} \circ in_{\mathsf{E}}} P = \mathsf{Lan}_{\pi_{\mathsf{B}}} \mathsf{Lan}_{in_{\mathsf{E}}} P$:

$$\begin{tikzcd}
\mathsf{E} \arrow[d, bend left, "in_{\mathsf{E}}"] \arrow[dr, bend left, "P"] \arrow[dd, bend right=60, "F"'] & \\
F[\mathsf{G}] \arrow[u, bend left, "\pi_{\mathsf{E}}"] \arrow[u, phantom, "\pi_{\mathsf{E}} \vdash in_{\mathsf{E}}"] \arrow[r, "\mathsf{Lan}_{in_{\mathsf{E}}} P"] \arrow[d, "\pi_{\mathsf{B}}"] \arrow[ur, phantom, "\Leftarrow \lambda_{in_{\mathsf{E}}}"] \arrow[dr, phantom, "\Swarrow \lambda_{\pi_{\mathsf{B}}}"] & \mathsf{Set} \\
\mathsf{B} \arrow[ur, bend right, "\mathsf{Lan}_{\pi_{\mathsf{B}}}(\mathsf{Lan}_{in_{\mathsf{E}}} P) = \mathsf{Lan}_F P"'] &
\end{tikzcd} \tag{25}$$

By Lemma 5.11, since $in_{\mathsf{E}} \dashv \pi_{\mathsf{E}}$, $\mathsf{Lan}_{in_{\mathsf{E}}} P = P \circ \pi_{\mathsf{E}}$. Since F by assumption is a G-relative opfibration, π_{B} is in particular a Grothendieck opfirbation, hence according to Lemma 5.9,

$$\mathsf{Lan}_{\pi_{\mathsf{B}}} P \circ \pi_{\mathsf{E}}(b) \cong \operatorname*{colim}_{(e,b,\gamma) \in \pi_{\mathsf{B}}^{-1}(b)} P(e) \cong \left(\coprod_{(e,b,\gamma) \in \pi_{\mathsf{B}}^{-1}(b)} P(e) \right) \big/ {\sim_{F[\mathsf{G}]}} \; .$$

Here, the equivalence relation $\sim_{F[\mathsf{G}]}$ is the least equivalence relation such that

$$((e, b, \gamma), p) \sim_{F[\mathsf{G}]} ((e', b, \gamma'), p') :\Leftrightarrow \exists (e, b, \gamma) \xrightarrow{(\varepsilon, id_b)} (e', b, \gamma') \in \pi_{\mathsf{B}}^{-1}(b) : p' = P(\varepsilon)p \, .$$

Finally, according to the definition of morphisms in $F[\mathsf{G}]$, for a morphism (ε, id_b) in the above equation exists only if $\gamma = \gamma' \circ F(\varepsilon)$, which explains the last isomorphism in (23). ◀

We will now demonstrate the utility of these results for evaluating convolution products. Invoking Theorem 5.12 yields the following results:

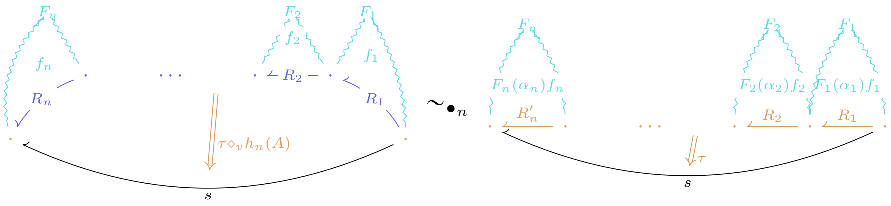
▶ **Lemma 5.13.** *Let $\mathbb{D}$ be a double category such that for all $n > 1$, the functors $h_n : \mathbb{D}_n \to \mathbb{D}_1$ are globular opfibrations (i.e., $\mathbb{D}$ has the positive CDP property). Denote by $\iota : \mathbb{D}_1^\bullet \to \mathbb{D}_1$ the inclusion functor from the wide subcategory of globular morphisms into $\mathbb{D}_1$, and define $\mathbb{D}_n^\bullet$ as the wide subcategory of $\mathbb{D}_n$ whose morphisms satisfy $h_n(\mathbb{D}_n) \in \mathbb{D}_1^\bullet$. Let $F_n, \ldots, F_1 : \mathbb{D}_1 \to \mathsf{Set}$ be covariant presheaves, and denote by $\mathbb{F}_n^\bullet : \mathbb{D}_n \to \mathsf{Set}$ the restriction of $F_n \times \ldots \times F_1$ to $\mathbb{D}_n^\bullet$. Then the convolution product formula simplifies as follows:*

$$(F_n * \ldots * F_1)(r) \cong \left(\coprod_{R \in \mathbb{D}_n^\bullet} \mathbb{D}_1^\bullet(h_n(R), r) \times \mathbb{F}_n^\bullet(R) \right) \big/ {\sim_{\bullet_n}} \tag{26}$$

where $\sim_{\bullet_n}$ is the least equivalence relation that satisfies

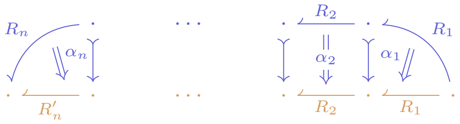
$$\begin{aligned} (R, (\sigma, f)) \sim_{\bullet_n} (R', (\sigma', f')) \Leftrightarrow \exists R - A \to R' \in \mathbb{D}_n^\bullet, (\gamma, g) \in \mathbb{D}_1^\bullet(h_n(R'), r) \times \mathbb{F}_n^\bullet(R) : \\ (\sigma, f) = (\gamma \circ h_n(A), g) \ \wedge \ (\sigma', f') = (\gamma, \mathbb{F}_n^\bullet(A)g) \, . \end{aligned} \tag{27}$$

In order to provide some intuition for the structure of convolution products within the refined framework, we provide below a graphical illustration of $\sim_{\bullet_n}$ (where $\sigma = \tau \diamond_v h_n(A)$):



(28)

$A =$



$\in \mathbb{D}_n^\bullet$

The preceding discussion allows us to give a fully rigorous proof of Theorem 5.5, which may be found in the long version of the paper (to appear).

5.4 A brief analysis of unitality

As already mentioned, neither the framed bicategory $\mathbb{S}\mathsf{pan}$ nor the double category $\mathbb{D}\mathsf{PO}$ modeling DPO-rewriting have the 0-CDP, in the sense that the functor $h_0 = U : \mathbb{D}_0 \to \mathbb{D}_1$ is not a globular opfibration. Nevertheless, the convolution product over $\mathbb{D}\mathsf{PO}$ is in fact strongly unital. One way to establish unitality is by showing that $\mathbb{D}\mathsf{PO}$ and similar double categories do satisfy a weakened version of the 0-CDP, equivalent to saying that h_0 is an opfibration relative to both the subcategory of S-vertical maps (i.e., double cells α such that $S(\alpha)$ is an identity in $\mathbb{D}_0$) and the subcategory of T-vertical maps. We leave a more detailed analysis of this phenomenon to future work.

6 Categorification of rule algebras

As a presheaf, the convolution product $\hat{\Delta}_s * \hat{\Delta}_r$ of two representable presheaves $\hat{\Delta}_r$ and $\hat{\Delta}_s$ is isomorphic to a *colimit of representables* by general considerations on categories of presheaves. Moreover, the fact that the convolution product $* : \widehat{\mathbb{D}} \times \widehat{\mathbb{D}} \to \widehat{\mathbb{D}}$ preserves colimits componentwise (Prop. 3.2) implies that it is entirely determined by its restriction $\mathbb{D}_1 \times \mathbb{D}_1 \to \widehat{\mathbb{D}}$ to representable presheaves. In the introduction, we recalled how the rule algebra product was typically defined as a *sum over admissible matchings* between two rules, see equation (1). We now categorify this formula by showing that in many situations the convolution product of representable presheaves is isomorphic to a *sum of representables* of the following form

$$\hat{\Delta}_{r_2} * \hat{\Delta}_{r_1} \quad \cong \quad \sum_{j \in J} \quad \hat{\Delta}_{s_j}$$

where the family of horizontal cells noted

$$r_2 \circledast r_1 \quad = \quad \big(s_j : A_j \rightharpoonup B_j\big)_{j \in J}$$

can be effectively computed from r_1 and r_2. The property means that, in a certain sense, the family $r_2 \circledast r_1 = (s_j)_{j \in J}$ *classifies* the convolution product of the presheaves $\hat{\Delta}_{r_2}$ with $\hat{\Delta}_{r_1}$, thus providing a categorified version of what is known as the concurrency theorem in rewriting theory (compare [1,4,5,7,13]). The intuition is that the composition of two rewrite rules can be classified into a number of different, disjoint cases, induced by all possible ways of matching the source of one rule with the target of the other.

In order to formalize this intuition in the language of double categories, we make from now on the assumption that our double category $\mathbb{D}$ satisfies the following property:

(i) the vertical category $\mathbb{D}_0$ has multi-sums.

We find useful to recall at this stage the notion of *multi-sum* due to Diers [12]. Suppose that A and B are objects in a category. A *multi-sum* (or multi-coproduct) of A and B is a family of cospans

$$\Big(A \xrightarrow{a_i} U_i \xleftarrow{b_i} B\Big)_{i \in I} \tag{29}$$

such that for any cospan $A \xrightarrow{f} X \xleftarrow{g} B$ there exists a unique $i \in I$ and a unique morphism $[f, g] : U_i \to X$ such that $f = [f, g] \circ a_i$ and $g = [f, g] \circ b_i$. The multi-sum generalizes the standard notion of coproduct $A \xrightarrow{a} A + B \xleftarrow{b} B$ to situations in which there may not necessarily be a single universal cospan through which all other cospans $A \xrightarrow{f} X \xleftarrow{g} B$ factor, but there is nonetheless a universal family (29) of such cospans. As with the ordinary coproduct of two objects, when it exists the multi-sum of A and B is unique up to unique isomorphism.

Notation: given two horizontal 1-cells $r_1 : A \rightharpoonup B$ and $r_2 : C \rightharpoonup D$ of the double category $\mathbb{D}$, we find sometimes convenient to write $\Sigma^*_{(r_2,r_1)}$ for the set of cospans $(m_2, m_1) = (c_i, b_i)$ appearing in the multi-sum of B and C. This notation is used in particular in §6.2.

6.1 A first easy version of categorification

We start by establishing a categorification of equation (1) under the general assumption that

(ii) the source and target functors $S, T : \mathbb{D}_1 \to \mathbb{D}_0$ are Grothendieck opfibrations.

This assumption holds for framed bicategories since it is weaker than the assumption that the pairing (T, S) is an opfibration. It also holds for the double category $\mathbb{T}\mathsf{RS}[\Sigma]$ of term rewriting which is not a framed bicategory. On the other hand, this assumption is too strong for the double category $\mathbb{D}\mathsf{PO}$, and we will see further below how to weaken it to prove a more general formula that also applies in that case. We establish that

▶ **Theorem 6.1.** *Assume $\mathbb{D}$ is a small double category satisfying assumptions (i) and (ii) and suppose that $r_1 : A \rightharpoonup B$ and $r_2 : C \rightharpoonup D$ are horizontal 1-cells in $\mathbb{D}$. In that case, the convolution product of two representable presheaves is isomorphic to the sum of representables*

$$\hat{\Delta}_{r_2} * \hat{\Delta}_{r_1} \quad\cong\quad \sum_{i\in I} \hat{\Delta}_{r_2\langle c_i\rangle \diamond_h \langle b_i\rangle r_1} \tag{30}$$

where the multi-sum of B and C is given by a family of cospans $(B \xrightarrow{b_i} U_i \xleftarrow{c_i} C)_{i\in I}$, and where $r_2\langle c_i\rangle$ denotes the S-pushforward of r_2 along c_i and $\langle b_i\rangle r_1$ denotes the T-pushforward of r_1 along b_i.

Proof. By definition, an element of $\hat{\Delta}_{r_2} * \hat{\Delta}_{r_1}$ evaluated at a generic horizontal 1-cell r consists of three double cells of the following shape:

$$\begin{tikzcd}
D \arrow[d, tail] & \arrow[d, Rightarrow, "\alpha_2"] & C \arrow[ll, harpoon', "r_2"'] \arrow[dr, tail, "g"] & & B \arrow[dl, tail, "f"'] & \arrow[d, Rightarrow, "\alpha_1"'] & A \arrow[ll, harpoon', "r_1"'] \arrow[d, tail] \\
\cdot \arrow[d, tail] & & & X \arrow[lll, harpoon', "s_2"] \arrow[d, Rightarrow, "\beta"] & & & \cdot \arrow[lll, harpoon', "s_1"] \arrow[d, tail] \\
\cdot & & & \, & & & \cdot \arrow[llllll, harpoon', "r"]
\end{tikzcd}$$

Since $(B \xrightarrow{b_i} U_i \xleftarrow{c_i} C)_{i\in I}$ is the multi-sum of B and C, there exists a unique $i \in I$ and a morphism $[f, g] : U_i \to X$ such that $f = [f, g] \circ b_i$ and $g = [f, g] \circ c_i$. By the assumption that S and T are opfibrations, the double cells α_1 and α_2 therefore factor as follows:

$$\begin{tikzcd}
D \arrow[d, tail] & \arrow[d, Rightarrow, "\alpha_2"] & C \arrow[ll, harpoon', "r_2"'] \arrow[dr, tail, "g"] & & B \arrow[dl, tail, "f"'] & \arrow[d, Rightarrow, "\alpha_1"'] & A \arrow[ll, harpoon', "r_1"'] \arrow[d, tail] & & D \arrow[d, tail] & \arrow[d, Rightarrow, "\text{S-opcart}"] & C \arrow[ll, harpoon', "r_2"'] \arrow[dr, tail, "c_i"] & & B \arrow[dl, tail, "u_i"'] & \arrow[d, Rightarrow, "\text{T opcart}"'] & A \arrow[ll, harpoon', "r_1"'] \arrow[d, tail] \\
\cdot & & & X \arrow[lll, harpoon', "s_2"] & & & \cdot \arrow[lll, harpoon', "s_1"] & = & \cdot \arrow[d, tail] & \arrow[d, Rightarrow, "\tilde{\alpha_2}"'] & & U_i \arrow[lll, "r_2\langle c_i\rangle"] \arrow[d, tail, "{[f,g]}"] & & \arrow[d, Rightarrow, "\tilde{\alpha}_1"] & \cdot \arrow[lll, "r_1\langle b_i\rangle"] \arrow[d, tail] \\
 & & & & & & & & \cdot & \, & & X \arrow[lll, harpoon', "s_2"] & & \, & \cdot \arrow[lll, harpoon', "s_1"]
\end{tikzcd}$$

Observe that the double cell $(\tilde{\alpha}_2 \diamond_h \tilde{\alpha}_1) \diamond_v \beta$ is an element of the representable $\hat{\Delta}_{r_2 \langle c_i \rangle \diamond_h \langle b_i \rangle r_1}$ evaluated at r. This defines a natural transformation from $\hat{\Delta}_{r_2} * \hat{\Delta}_{r_1}$ to $\sum_{i \in I} \hat{\Delta}_{r_2 \langle c_i \rangle \diamond_h \langle b_i \rangle r_1}$, which is invertible by the universal properties of the multi-sum and the pushforward. ◀

▶ **Example 6.2.** In Example 3.3 we saw how the convolution product $\hat{\Delta}_r * \hat{\Delta}_r$ of the representable presheaf for the rewrite rule $r : m(e,x) \rightharpoonup x$ decomposes as the sum (9) of representables. This decomposition may be seen as a consequence of Theorem 6.1. First, note that the multi-sum of x and $m(e,x)$ exists in $\mathbb{T}\mathsf{RS}[\Sigma]_0$, and is given by the minimal set of four unifying context/substitution pairs depicted below (with the last corresponding to the *disjoint matching* of the two terms, up to variable renaming).

$$\begin{tikzcd}
m(e,x) \arrow[dr, equal] & & x \arrow[dl, tail, "(-|m(e,x)/x)"] \\
& m(e,x) &
\end{tikzcd}
\qquad
\begin{tikzcd}
m(e,x) \arrow[dr, equal] & & x \arrow[dl, tail, "(m(-,x)|e/x)"] \\
& m(e,x) &
\end{tikzcd}$$

$$\begin{tikzcd}
m(e,x) \arrow[dr, equal] & & x \arrow[dl, tail, "(m(e,-)|x/x)"] \\
& m(e,x) &
\end{tikzcd}
\qquad
\begin{tikzcd}
m(e,x) \arrow[dr, tail, "(-,y|id)"'] & & x \arrow[dl, tail, "(m(e,x),-|y/x)"] \\
& m(e,x),y &
\end{tikzcd}$$

Moreover, observe that the functors $S, T : \mathbb{T}\mathsf{RS}[\Sigma]_1 \to \mathbb{T}\mathsf{RS}[\Sigma]_0$ are Grothendieck opfibrations, indeed even discrete opfibrations: the S-pushforward of a rule $\mathbf{t} \rightharpoonup \mathbf{t}'$ along a vertical 1-cell $\mathbf{t} \rightarrowtail \mathbf{u} = C[\mathbf{t}\sigma]$ is the rule $C[\mathbf{t}\sigma] \rightharpoonup C[\mathbf{t}'\sigma]$, and similarly for the T-pushforward along a vertical 1-cell $\mathbf{t}' \rightarrowtail \mathbf{u}' = C[\mathbf{t}'\sigma]$. Instantiating (30), we recover (9).

6.2 A more advanced version of categorification

In this subsection, we refine the assumptions of the previous subsection in order to establish in a more general framework that the convolution product of two representable presheaves is a sum of representables. One main motivation is to include among our examples the double category $\mathbb{D}\mathsf{PO}$ and other double categories of interest in graph rewriting theory. From now on, we thus make the following two assumptions (iia) and (iib) on the double category $\mathbb{D}$, which generalize the assumption (ii) just made in the previous subsection:

(iia) the source functor $S : \mathbb{D}_1 \to \mathbb{D}_0$ is a multi-opfibration;

(iib) the target functor $T : \mathbb{D}_1 \to \mathbb{D}_0$ is a residual multi-opfibration.

The three assumptions (i), (iia) and (iib) are part of the definition of *compositional rewriting double category (crDC)* formulated in [5] where the interested reader will find the notion of (residual) multi-opfibration. We establish that

▶ **Theorem 6.3.** *Assume $\mathbb{D}$ is a small double category satisfying assumptions (i), (iia) and (iib) and suppose that $r_1 : A \rightharpoonup B$ and $r_2 : C \rightharpoonup D$ are horizontal 1-cells in $\mathbb{D}$. In that case, the convolution product of two representable presheaves is isomorphic to the sum of representables*

$$\hat{\Delta}_{r_2} * \hat{\Delta}_{r_1} \cong \sum_{(m_2,m_1) \in \Sigma^*_{(r_2,r_1)}} \ \sum_{(m_{1\ \star j}, \beta_{1,j}) \in T^*(r_1; m_1)} \ \sum_{\beta_{2,j,k} \in S^*(r_2, m_{1\ \star j} \circ m_2)} \hat{\Delta}_{\beta_{2,j,k}(r_2) \diamond_h \beta_{1,j}(r_1)} \tag{31}$$

where $\Sigma^*_{(r_2,r_1)}$ denotes the set of cospans appearing in the multi-sum of B and C, and where a choice of cleavages S^*, T^* for S, T on $\mathbb{D}_0$, respectively.

For illustration, every element of the presheaf $\hat{\Delta}_{r_2} * \hat{\Delta}_{r_1}$ at instance the horizontal arrow r may be factored in the following way:

$$\begin{tikzcd}
\cdot \arrow[dd, tail] & \cdot \arrow[l, "r_2"'] \arrow[dr, tail, "m_2"'] & & \cdot \arrow[dl, tail, "m_1"] & \cdot \arrow[l, "r_1"'] \arrow[dd, tail] \\
& & \cdot \arrow[d, tail, "m_{1\star\, j}"] & & \\
\cdot \arrow[d, tail] & & \cdot \arrow[ll] & & \cdot \arrow[ll] \arrow[d, tail] \\
\cdot & & & & \cdot \arrow[llll, "r"]
\arrow[from=1-2, to=3-2, Rightarrow, "\beta_{2,j,k}"']
\arrow[from=1-4, to=3-4, Rightarrow, "\beta_{1,j}"]
\arrow[from=3-3, to=4-3, Rightarrow, "\alpha"]
\end{tikzcd} \tag{32}$$

where the pair (m_2, m_1) of vertical arrows $m_1 : B \rightarrowtail U_i$ and $m_2 : C \rightarrowtail U_i$ is an element of the set $\Sigma^*_{(r_2,r_1)}$ of cospans appearing in the multi-sum of B and C.

7 Conclusion

In this paper, we explain how our original project of categorifying the rule algebra $(\mathcal{R}, \star)$ associated to a compositional rewriting theory brought us to formulate a very general notion of convolution product $* : \widehat{\mathbb{D}} \times \widehat{\mathbb{D}} \to \widehat{\mathbb{D}}$ for vertical presheaves over a double category $\mathbb{D}$. The convolution product is only oplax associative in general, and we thus investigate in the paper sufficient conditions on the double category $\mathbb{D}$ for the convolution product to be strongly associative. We start by establishing that the convolution product is strongly associative in the case of framed bicategories, but not necessarily strongly unital. We then extend this result by formulating a more general cylindrical decomposition property for double categories (as an instance of the more general notion of relative opfibration) which, we show, implies that the convolution product is strongly associative under the assumption of n-CDP for all $n > 0$. The question of the strong unitality of the convolution product appears to be very subtle and interesting: it fails for the framed bicategory $\mathbb{S}\mathsf{pan}$ (Example 4.5), it holds for $\mathbb{T}\mathsf{RS}[\Sigma]$ as a consequence of 0-CDP, and it holds for $\mathbb{D}\mathsf{PO}$ *despite* the failure of 0-CDP.

One main achievement of the paper is to justify the view that the convolution product $* : \widehat{\mathbb{D}} \times \widehat{\mathbb{D}} \to \widehat{\mathbb{D}}$ categorifies the product $\star : \mathcal{R} \otimes_{\mathbf{k}} \mathcal{R} \to \mathcal{R}$ of the rule algebra, thanks to formulas (30) and (31) which play the role of formula (1). We see this as a foundation for developing a deeper understanding of the rule algebra representation $\rho : \mathcal{R} \to \mathit{Endo}_{\mathbf{k}}(\mathcal{S})$ defined by formula (2) in the introduction, as well as formula (3). A strong benefit of categorification which we will clarify in future work is that it unifies, thanks to the Yoneda embedding, the rule algebra $\mathcal{R}$ with its action on states in $\mathcal{S}$ through the representation ρ, following a healthy analogy with the well-known principle of Cayley theorem in algebra.

References

1 Nicolas Behr. Sesqui-Pushout Rewriting: Concurrency, Associativity and Rule Algebra Framework. In *Proceedings of GCM 2019*, volume 309 of *EPTCS*, pages 23–52, 2019. doi:10.4204/eptcs.309.2.

2 Nicolas Behr. Tracelets and tracelet analysis of compositional rewriting systems. In *Proceedings of ACT 2019*, volume 323 of *EPTCS*, pages 44–71, 2020. doi:10.4204/EPTCS.323.4.

3 Nicolas Behr, Vincent Danos, and Ilias Garnier. Stochastic mechanics of graph rewriting. In *Proceedings of LiCS '16*. ACM Press, 2016. doi:10.1145/2933575.2934537.

4 Nicolas Behr, Russ Harmer, and Jean Krivine. Concurrency theorems for non-linear rewriting theories. In Proceedings of ICGT 2021, volume 12741 of *LNCS*, pages 3–21. Springer, 2021. doi:10.1007/978-3-030-78946-6_1.

5 Nicolas Behr, Russ Harmer, and Jean Krivine. Fundamentals of Compositional Rewriting Theory, 2022. doi:10.48550/ARXIV.2204.07175.

6 Nicolas Behr and Joachim Kock. Tracelet Hopf Algebras and Decomposition Spaces (Extended Abstract). In Proceedings of ACT 2021, volume 372 of *EPTCS*, pages 323–337, 2022. doi:10.4204/EPTCS.372.23.

7 Nicolas Behr and Jean Krivine. Compositionality of Rewriting Rules with Conditions. *Compositionality*, 3, 2021. doi:10.32408/compositionality-3-2.

8 Nicolas Behr, Jean Krivine, Jakob L. Andersen, and Daniel Merkle. Rewriting theory for the life sciences: A unifying theory of ctmc semantics. *Theoretical Computer Science*, 884:68–115, 2021.

9 Nicolas Behr and Pawel Sobocinski. Rule Algebras for Adhesive Categories (extended journal version). *LMCS*, Volume 16, Issue 3, 2020. doi:10.23638/LMCS-16(3:2)2020.

10 Andrea Corradini et al. Sesqui-Pushout Rewriting. In *Graph Transformations*, volume 4178 of *LNCS*, pages 30–45. Springer Berlin Heidelberg, 2006.

11 Brian Day. On closed categories of functors. In *Reports of the Midwest Category Seminar IV*, number 137 in Lecture Notes in Mathematics, pages 1–38, Berlin, Heidelberg, New York, 1970. Springer-Verlag.

12 Y. Diers. *Familles universelles de morphismes*, volume 145 of *Publications de l'U.E.R. mathématiques pures et appliquées*. Université des sciences et techniques de Lille I, 1978.

13 H. Ehrig et al. Fundamentals of Algebraic Graph Transformation. *Monographs in Theoretical Computer Science*, 2006. doi:10.1007/3-540-31188-2.

14 Niles Johnson and Donald Yau. *2-Dimensional Categories*. Oxford University Press, January 2021. doi:10.1093/oso/9780198871378.001.0001.

15 Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer New York, 2 edition, 1978. doi:10.1007/978-1-4757-4721-8.

16 Tom Leinster. *Higher Operads, Higher Categories*, volume 298 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 2004.

17 Paolo Perrone and Walter Tholen. Kan extensions are partial colimits. *Applied Categorical Structures*, 30(4):685–753, 2022. doi:10.1007/s10485-021-09671-9.

18 Michael Shulman. Framed bicategories and monoidal fibrations. *Theory and Applications of Categories*, 20(18):650–738, 2008.

For the Metatheory of Type Theory, Internal Sconing Is Enough

Rafaël Bocquet
Eötvös Loránd University, Budapest, Hungary

Ambrus Kaposi
Eötvös Loránd University, Budapest, Hungary

Christian Sattler
Chalmers University of Technology, Gothenburg, Sweden

Abstract

Metatheorems about type theories are often proven by interpreting the syntax into models constructed using categorical gluing. We propose to use only sconing (gluing along a global section functor) instead of general gluing. The sconing is performed internally to a presheaf category, and we recover the original glued model by externalization.

Our method relies on constructions involving two notions of models: first-order models (with explicit contexts) and higher-order models (without explicit contexts). Sconing turns a displayed higher-order model into a displayed first-order model.

Using these, we derive specialized induction principles for the syntax of type theory. The input of such an induction principle is a boilerplate-free description of its motives and methods, not mentioning contexts. The output is a section with computation rules specified in the same internal language. We illustrate our framework by proofs of canonicity and normalization for type theory.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases type theory, presheaves, canonicity, normalization, sconing, gluing

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.18

Related Version *Full Version including Appendices for Parametricity and Cubical Type Theory*: `https://arxiv.org/abs/2302.05190`

Funding *Ambrus Kaposi*: Supported by the Bolyai Fellowship of the Hungarian Academy of Sciences and by the "Application Domain Specific Highly Reliable IT Solutions" project of the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 funding scheme.

1 Introduction

The syntax of a type theory can be presented as the initial object in the category of models of a generalized algebraic theory (GAT), i.e. as a quotient inductive-inductive type (QIIT). Initiality provides an induction principle for the syntax, namely the dependent eliminator of the QIIT. Metatheoretic properties of the syntax, such as canonicity or normalization, can then be proven by carefully constructing models of the theory displayed over the syntax, or equivalently the motives and methods of the induction principle. However, the presentation of the syntax as a QIIT includes an explicit encoding of the substitution calculus of the theory; in particular every type or term former comes with a substitution rule. If all of the components of a complicated model are written explicitly, one has to prove that they all respect substitution. More importantly, when working exclusively at this level of generality, it is not easy to abstract the proof methods into reusable theorems.

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 18; pp. 18:1–18:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

An alternative to the (first-order) genereralized algebraic presentation is to present a type theory as a second-order or higher-order theory, for example by using a Logical Framework [20] or Uemura's representable map categories [36]. A higher-order presentation enables the use of higher-order abstract syntax (HOAS), in which binders are encoded by metatheoretic functions. In practice, this means that stability under substitution is implicit. Semantically, HOAS admits an interpretation in the internal language of presheaf categories [22]. Within the internal language of a presheaf category, all constructions are automatically stable under the morphisms of the base category.

In this work we propose an approach that combines the strengths of the first-order and higher-order presentations. We consider two notions of models of a type theory: first-order models correspond to the first-order presentation, while higher-order models correspond to the higher-order presentation. Typically, first-order models are categorical models of type theory (categories with families equipped with additional structure), while higher-order models are approximately universes closed under the type formers of the theory. Both of these notions make sense both externally and in the internal language of a presheaf category. We also have notions of displayed higher-order and first-order models, corresponding to the motives and methods of induction principles. We present a small number of constructions switching between external, internal, first-order and higher-order models. Any of these constructions is individually simple, but they can be composed to derive the induction principles we need to prove metatheoretic results. The constructions are listed below (FOM = first-order model, HOM = higher-order model). The restriction and externalization operations can also be applied to displayed first-order models and morphisms of first-order models.

Construction	Input	Output
Internalization	FOM $\mathcal{M}$	HOM $\mathbb{C}$ in $\mathbf{Psh}(\mathcal{M})$
Set-contextualization	HOM $\mathbb{M}$	FOM $\mathbf{Set}_{\mathbb{M}}$
Telescopic contextualization	HOM $\mathbb{M}$	FOM $\mathbf{Tele}_{\mathbb{M}}$
Restriction	Functor $F : \mathcal{C} \to \mathcal{D}$, FOM $\mathcal{M}$ in $\mathbf{Psh}(\mathcal{D})$	FOM $F^*(\mathcal{M})$ in $\mathbf{Psh}(\mathcal{C})$
Externalization	FOM $\mathcal{M}$ in $\mathbf{Psh}(\mathcal{D})$	External FOM $1^*_{\mathcal{D}}(\mathcal{M})$
Scone-contextualization	FOM $\mathcal{M}$, Displayed HOM $\mathbb{M}^\bullet$ over $\mathcal{M}$	Displayed FOM $\mathbf{Scone}_{\mathbb{M}^\bullet}$ over $\mathcal{M}$

The most notable operations are the contextualizations[1], which turn higher-order models into first-order models. The **Set**-contextualization generalizes the construction of a first-order model from a universe; its underlying category is always the category of sets. The telescopic contextualization is the contextual core of the **Set**-contextualization; it restricts the underlying category to a category of telescopes. The **Scone**-contextualization is a generalization of the **Set**-contextualization to displayed higher-order and first-order models, which correspond to the motives and methods of induction principles; its underlying category is always the Sierpinski cone, also called scone, of some first-order model $\mathcal{M}$.

Our main observation is that sconing (that is **Scone**-contextualization) internally to a presheaf category corresponds when viewed externally to a more complicated gluing construction. For example, the normalization model from [12] can be recovered as the externalization of the **Scone**-contextualization of a displayed higher-order model constructed in presheaves over the category of renamings.

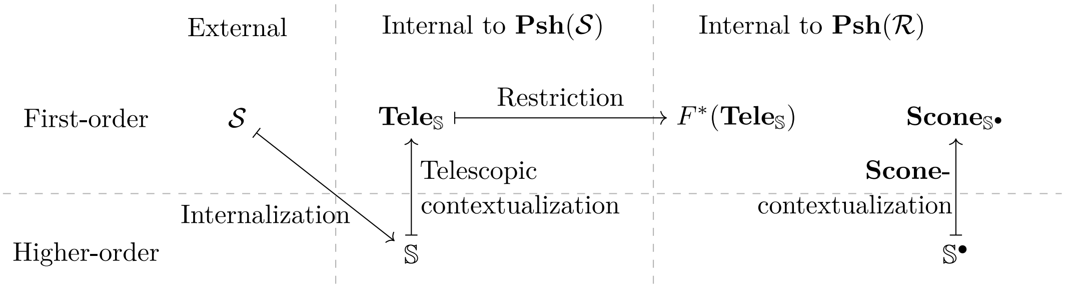
[1] The word contextualization reflects the fact that these operations make contexts explicit. It is not related to the notion of contextual model.

The main intended application of these constructions is the statement and proofs of relative induction principles. It is typically the case that the result of induction over the syntax of type theory only holds over some of the syntactic contexts, or is only stable under some of the syntactic substitutions. For example, canonicity only holds over the empty context. Normalization holds over every context, but is only stable under renamings. This situation is described by a functor into the syntax $F : \mathcal{R} \to \mathcal{S}$: the result of the induction should be stable under the morphisms of $\mathcal{R}$. These functors are called "figure shapes" by Sterling [32], they are also related to worlds in Twelf [29]. We prove the relative induction principles for the following functors. For cubical type theories, see Appendix C in the Full Version of the paper.

$1_{\mathbf{Cat}}$	$\to \mathcal{S}_{\mathcal{T}}$	Canonicity [12] (Theorem 14)
$\mathbf{Ren}_{\mathcal{S}}$	$\to \mathcal{S}_{\mathcal{T}}$	Normalization [4, 12] (Theorem 18)
$\square$	$\to \mathcal{S}_{\mathsf{CTT}}$	(Homotopy/strict) canonicity for cubical type theory [13]
$\mathcal{A}_{\square}$	$\to \mathcal{S}_{\mathsf{CTT}}$	Normalization for cubical type theory [34]

We also show how to prove canonicity and normalization by instantiating the relative induction principles for $1_{\mathbf{Cat}} \to \mathcal{S}_{\mathcal{T}}$ and $\mathbf{Ren}_{\mathcal{S}} \to \mathcal{S}_{\mathcal{T}}$. We don't prove canonicity nor normalization for cubical type theory, but we expect that the currently known proofs could be reformulated as instances of the relative induction principles for $\square \to \mathcal{S}_{\mathsf{CTT}}$ and $\mathcal{A}_{\square} \to \mathcal{S}_{\mathsf{CTT}}$.

Typically, the category $\mathcal{R}$ can be described as the initial object of some category of structured categories. A relative induction principle with respect to the functor F is an induction principle that combines the universal properties of $\mathcal{R}$ and $\mathcal{S}$. In our previous work [10], the relative induction principles were stated in terms of the rather ad-hoc notions of "displayed models without context extensions" and "relative sections". In the present work, the input of a relative induction principle is a displayed higher-order model $\mathbb{S}^\bullet$, and the result is just a (first-order) section $[\![-]\!]$ of $\mathbf{Scone}_{\mathbb{S}^\bullet}$. The previous notion of "displayed model without context extension" is recovered in the special case of displayed higher-order models over $F^*(\mathbf{Tele}_{\mathbb{S}})$. The following diagram illustrates the constructions involved in the statement and proof of a relative induction principle ($F : \mathcal{R} \to \mathcal{S}$).


External
Internal to $\mathbf{Psh}(\mathcal{S})$
Internal to $\mathbf{Psh}(\mathcal{R})$
First-order
Higher-order
$\mathcal{S}$
$\mathbf{Tele}_{\mathbb{S}}$
Restriction
$F^*(\mathbf{Tele}_{\mathbb{S}})$
$\mathbf{Scone}_{\mathbb{S}^\bullet}$
Telescopic contextualization
Scone-contextualization
Internalization
$\mathbb{S}$
$\mathbb{S}^\bullet$

Both $\mathbb{S}^\bullet$ and $\mathbf{Scone}_{\mathbb{S}^\bullet}$ are displayed over the internal first-order model $F^*(\mathbf{Tele}_{\mathbb{S}})$.

$$\begin{tikzcd}
\mathbb{S}^\bullet \arrow[r, mapsto] \arrow[dr] & \mathbf{Scone}_{\mathbb{S}^\bullet} \arrow[d, shift right] \\
& F^*(\mathbf{Tele}_{\mathbb{S}}) \arrow[u, dashed, shift right, "{[\![-]\!]}"']
\end{tikzcd}$$

An important feature of our work is that the section $[\![-]\!]$ admits good computational behaviour, although we do not formally analyze this behaviour. In fact, part of our understanding comes from looking at the computational behaviour of the congruence operation ap in higher observational type theory [31, 6], which is also a morphism of (first-order) models internally to presheaves over the syntax of H.O.T.T.

Related work

Logical relations and categorical gluing. The initial motivation for this work was the understanding of algebraic and reduction-free normalization proofs for dependent type theories. Variants of categorical gluing were used to prove canonicity or normalization for simple types [2, 15, 35], System F [3], and dependent types [4, 12, 14]. These can be contrasted with reduction based normalization proofs such as [1, 30].

Logical relations were used to prove syntactic parametricity for type theory [9, 5] and definability for simply typed lambda calculus [25]. It was shown that categorical gluing generalizes both syntactic parametricity and canonicity proofs [27].

Logical frameworks and higher-order abstract syntax. Higher-order abstract syntax (HOAS) is the use of metatheoretic functions to specify syntactic binders. Hofmann [22] has explained how HOAS can be interpreted in the internal language of presheaf categories.

Uemura [36] has given a general definition of type theory based on these ideas, which we will call second-order generalized algebraic theories (SOGATs). It generalizes notions of second-order algebraic theories that have been studied by Fiore and Mahmoud [16]. Harper presents an equational variant of logical framework [19] for defining theories with bindings corresponding to SOGATs. We believe that the constructions in our paper generalize to any SOGAT. Gratzer and Sterling [18] propose using LCCCs to define higher order theories (without representability conditions) which correspond to our higher order models, but we also consider first order models.

Synthetic Tait computability. Synthetic Tait computability (STC, [32, 33]) is an approach that relies on the internal language of the Artin gluing of toposes, typically constructed by gluing syntactic and semantic toposes. A pair of open/closed modalities can be used to distinguish the syntactic and semantic parts in the internal language of the Artin gluing. STC has been applied to proofs of normalization for cubical type theory [34], multimodal type theory [17] and simplicial type theory [37].

Both STC and our approach provide a synthetic setting for proofs of metatheorems. Our approach is perhaps simpler in some aspects, e.g. we don't use any modalities and don't need to use realignment; we acknowledge that this is partly a matter of preference. The main advantage of our approach over STC is that we have an internal specification of the result of an induction principle.

In his thesis [32], Sterling briefly discusses the notion of Henkin model of a higher-order theory. A Henkin model is a higher-order model with a non-standard interpretation of the dependent products. We note that Henkin models are closely related to first-order models: the Henkin models of a second-order theory are equivalent to the democratic first-order models (this is [36, Theorem 7.30]).

Contributions

The main takeaway of this paper is that when using HOAS, we should not discard the first order presentation, even in an internal setting. In particular, the right notion of displayed higher-order model (input of an induction principle) lies over a first-order model. The output of an induction principle is also a section of first-order models, still internal.

The technical contributions are:

- the relative induction principles which combine the initiality of the underlying category and initiality of the syntax; we derive four such induction principles;
- the internalization, contextualization, externalization constructions which let us formulate the relative induction principles;
- the derivation of internal sections by analyzing the category of sections.

The main takeaway is supported by applications of the induction principles: boilerplate-free proofs of canonicity and normalization. The fact that the notion of section is specified internally has the feature that we can reuse it directly in subsequent inductions. We exploit this in the proof of uniqueness of normal forms which is proven in a separate step after normalization, relying on how the section computes normal forms.

Structure of the paper

In Section 2 we define first-order- and higher-order models of an example type theory, and constructions relating them. In Section 3, we define displayed higher order models which collect the motives and methods of an induction principle. We also define sconing which turns a displayed higher order model into its first order variant, also providing a notion of section. Then we move on to applications: we prove canonicity in Section 4 using an induction principle relative to the empty context (Theorem 14) which is a trivial consequence of our previous definitions. In Section 5, we prove normalization using an induction principle relative to renamings (Theorem 18). This induction principle is proved using the methods described in Section 6. Another application (syntactic parametricity) of Theorem 14 is described in the Full Version of the paper, Appendix B. In Appendix C of the Full Version, cubical variants of the above induction principles are also proven.

Background

We assume some familiarity with the categorical semantics of type theory [11, 5] and with the use of extensional type theory as the internal language of presheaf categories [21].

We use the notion of locally representable dependent presheaf to encode context extensions (see Definition 21). This definition is a more indexed formulation of the notion of representable natural transformation, which was used by Awodey to give an alternative definition of CwFs, known as natural models [8].

2 First-order and higher-order models

Our running example is a minimal dependent type theory $\mathcal{T}$ with only Π-types, but our constructions directly generalize to larger type theories. Some other type theories are considered in the Full Version of this paper, including dependent type theories with universes and cubical type theories. We leave generalization to arbitrary second-order generalized algebraic theories to future work.

We define notions of higher-order and first-order models for $\mathcal{T}$. A higher-order model is essentially a universe closed under dependent products, while a first-order model is a category with families (CwF) equipped with Π-types. The higher-order models are the models of some higher-order theory $\mathcal{T}^{\mathsf{ho}}$ (a theory whose operations can have a higher-order sort; classified by some locally cartesian closed category), whereas the first-order models are the models of some (first-order) essentially[2] algebraic theory $\mathcal{T}^{\mathsf{fo}}$ (a theory whose operations have a first-order sort; classified by some finitely complete category).

[2] The distinction between generalized algebraic theories and essentially algebraic theories is not relevant in this paper.

2.1 Definitions

▶ **Definition 1.** *A* ***higher-order model*** *of $\mathcal{T}$ consists of the following families and operations:*

$$\begin{aligned}
&\mathsf{Ty} && : \mathrm{Set},\\
&\mathsf{Tm} && : \mathsf{Ty} \to \mathrm{Set},\\
&\Pi && : \forall (A : \mathsf{Ty})(B : \mathsf{Tm}(A) \to \mathsf{Ty}) \to \mathsf{Ty},\\
&\mathsf{app} && : \forall A\ B\ (f : \mathsf{Tm}(\Pi(A, B)))\ (a : \mathsf{Tm}(A)) \to \mathsf{Tm}(B(a)),\\
&\mathsf{lam} && : \forall A\ B\ (b : (a : \mathsf{Tm}(A)) \to \mathsf{Tm}(B(a))) \to \mathsf{Tm}(\Pi(A, B)),
\end{aligned}$$

subject to equations corresponding to the β- and η-rules:

$$\mathsf{lam}(\lambda a \mapsto \mathsf{app}(f, a)) = f, \qquad \mathsf{app}(\mathsf{lam}(b), a) = b(a). \qquad \lrcorner$$

▶ **Definition 2.** *A* ***first-order model*** *of $\mathcal{T}$ is a CwF $\mathcal{C}$ equipped with Π-types.* ⌟

The notion of first-order model can be presented by a first-order generalized algebraic theory $\mathcal{T}^{\mathsf{fo}}$. Equivalently, a first-order model is a category $\mathcal{C}$ with a terminal object $1_{\mathcal{C}}$, along with a (global) higher-order model $\mathbb{C}$ of $\mathcal{T}$ in $\mathbf{Psh}(\mathcal{C})$ such that the dependent presheaf $\mathbb{C}.\mathsf{Tm}$ is locally representable. The higher-order model $\mathbb{C}$ is called the internalization of $\mathcal{C}$.

We use blackboard bold letters $\mathbb{M}$, $\mathbb{N}$, $\mathbb{C}$, $\mathbb{S}$, $\mathbb{R}$, etc. to refer to higher-order models, and calligraphic letters $\mathcal{M}$, $\mathcal{N}$, $\mathcal{C}$, $\mathcal{S}$, $\mathcal{R}$, etc. to refer to first-order models. We try to use the corresponding letter for the underlying internal higher-order model of a first-order model, e.g. if $\mathcal{C}$ is an external first-order model, we use $\mathbb{C}$ for its underlying internal higher-order model in $\mathbf{Psh}(\mathcal{C})$. We denote the components of a model by $\mathbb{C}.\mathsf{Ty}$, $\mathcal{C}.\mathsf{Tm}$, $\mathcal{C}.\Pi$, etc.

We write $\mathbf{Mod}_{\mathcal{T}}$ for the 1-category of first-order models of $\mathcal{T}$, and $\mathcal{S}_{\mathcal{T}}$ or just $\mathcal{S}$ for its initial object (the letter “S” standing for both syntax and substitutions).

2.2 Contextualization

In general, we almost never want to construct all of the components of a first-order model explicitly, because checking functoriality and naturality conditions without relying on the internal language of a presheaf model is tedious. For some models however, the functoriality and naturality conditions hold trivially. This is the case for the “standard model” of type theory over the category of sets: when defining this standard model in intensional type theory, all naturality and functoriality conditions hold definitionally.

The construction of the standard model generalizes to the construction of a first-order model from a higher-order model, which we now describe.

▶ **Construction 3** (**Set**-Contextualization)**.** *Let $\mathbb{M}$ be a higher-order model of $\mathcal{T}$. We construct a first-order model $\mathbf{Set}_{\mathbb{M}}$, called the* **Set*****-contextualization*** *of $\mathbb{M}$. Its underlying category is the category* **Set** *of sets.*

The types and terms of $\mathbf{Set}_{\mathbb{M}}$ are indexed families of types and terms of $\mathbb{M}$:

- *A type over $\Gamma \in \mathbf{Set}$ is a function $\Gamma \to \mathbb{M}.\mathsf{Ty}$.*
- *Type substitution along a function $f : \Delta \to \Gamma$ is precomposition with f.*
- *A term of type $A : \Gamma \to \mathbb{M}.\mathsf{Ty}$ is a dependent function $(\gamma : \Gamma) \to \mathbb{M}.\mathsf{Tm}(A(\gamma))$.*
- *Term substitution along a function $f : \Delta \to \Gamma$ is precomposition with f.*
- *The functoriality of substitution is associativity of function composition.*

The context extensions are given by dependent sums in **Set***:*

$$(\Gamma.A) \triangleq (\gamma : \Gamma) \times \mathbb{M}.\mathsf{Tm}(A(\gamma)).$$

The type-theoretic operations are all defined pointwise:

$$\begin{aligned}
\mathbf{Set}_{\mathbb{M}}.\Pi(\Gamma, A, B) &\triangleq \lambda\gamma \mapsto \mathbb{M}.\Pi(A(\gamma), \lambda a \mapsto B(\gamma, a)),\\
\mathbf{Set}_{\mathbb{M}}.\mathsf{app}(\Gamma, f, a) &\triangleq \lambda\gamma \mapsto \mathbb{M}.\mathsf{app}(f(\gamma), a(\gamma)),\\
\mathbf{Set}_{\mathbb{M}}.\mathsf{lam}(\Gamma, b) &\triangleq \lambda\gamma \mapsto \mathbb{M}.\mathsf{lam}(\lambda a \mapsto b(\gamma, a)).
\end{aligned}$$

The β- and η-rules for $\mathbf{Set}_{\mathbb{M}}$ *hold as a consequence of the corresponding rules for* $\mathbb{M}$*.*

The naturality conditions are all trivial. For example, in the case of the Π *type former, we have to check* $\Pi(\Gamma, A, B)[f] = \Pi(\Delta, A[f], B[f^+])$ *for any* $f : \Delta \to \Gamma$*, where* $f^+(\gamma, a) = (f(\gamma), a)$*. This amounts to checking the equality*

$$(\lambda\gamma \mapsto \mathbb{M}.\Pi(A(\gamma), \lambda a \mapsto B(\gamma, a))) \circ f = (\lambda\gamma \mapsto \mathbb{M}.\Pi((A \circ f)(\gamma), \lambda a \mapsto (B \circ f^+)(\gamma, a))). \lrcorner$$

▶ Remark 4. Note that the underlying category **Set** of $\mathbf{Set}_{\mathbb{M}}$ now has two CwF structures:

- An *inner* CwF structure, as defined in Construction 3.
- An *outer* CwF structure, corresponding to the usual CwF structure on the category of sets, modeling extensional type theory.

Together, they form a model of two-level type theory [7].

2.3 Telescopic contextualization

A first-order order model is said to be contextual when every object can be uniquely written as an iterated context extension starting from the empty context. The contextual first-order models form a coreflective subcategory $\mathbf{Mod}^{\mathsf{cxl}}_{\mathcal{T}}$ of $\mathbf{Mod}_{\mathcal{T}}$: the inclusion $\mathbf{Mod}^{\mathsf{cxl}}_{\mathcal{T}} \to \mathbf{Mod}_{\mathcal{T}}$ has a right adjoint cxl: the *contextual core* $\mathsf{cxl}(\mathcal{C})$ has as objects the iterated context extensions of $\mathcal{C}$, also known as telescopes, over the empty context.

▶ **Definition 5.** *Let* $\mathbb{M}$ *be a higher-order model of* $\mathcal{T}$*. The* ***telescopic contextualization*** $\mathbf{Tele}_{\mathbb{M}}$ *is the contextual core of the* **Set***-contextualization* $\mathbf{Set}_{\mathbb{M}}$*.* ⌟

By general properties of the contextual core, there is a model morphism $\lfloor - \rfloor : \mathbf{Tele}_{\mathbb{M}} \to \mathbf{Set}_{\mathbb{M}}$ that is bijective on types and terms. (There is a cofibrantly generated factorization system on first-order models with such morphisms as its right class. The contextual models are precisely those in the left class.)

When working internally to some presheaf category $\mathbf{Psh}(\mathcal{C})$, another related construction involves the internal subcategory spanned by $よ : \mathsf{Ob}_{\mathcal{C}} \to \mathrm{Set}$, where $\mathsf{Ob}_{\mathcal{C}}$ is the discrete presheaf on the set of objects of $\mathcal{C}$, and $よ$ internalizes the Yoneda embedding. This "Yoneda universe" has been used by Hu et al. [23] to give semantics to contextual types.

2.4 Internal first-order models

Since the notion of first-order model is described by an essentially algebraic theory, it can be interpreted in any finitely complete category. In particular, there is a notion of internal first-order model in any category $\widehat{\mathcal{C}}$ of small presheaves, obtained by letting Set stand for the Hofmann-Streicher universe of the presheaf topos $\mathbf{Psh}(\mathcal{C})$ in the definition of model.

▶ **Proposition 6.** *The following three notions are equivalent:*

1. *First-order models of $\mathcal{T}$ in $\mathbf{Psh}(\mathcal{C})$;*
2. *Finite limit preserving functors $\mathcal{T}^{\mathsf{fo}} \to \widehat{\mathcal{C}}$, where $\mathcal{T}^{\mathsf{fo}}$ is the finitely complete category classifying the first-order models of $\mathcal{T}$;*
3. *Functors $\mathcal{C} \to \mathbf{Mod}_{\mathcal{T}}^{\mathsf{op}}$.*

Proof. This is well-known [24, D1.2.14]. The equivalence between (1) and (2) is the fact that $\mathcal{T}^{\mathsf{fo}}$ classifies the first-order models of $\mathcal{T}$. The equivalence between (2) and (3) follows from the fact that finite limits in $\widehat{\mathcal{C}}$ are computed pointwise. ◀

2.5 Restriction and externalization

Another important operation on first-order models is the restriction of a first-order model $\mathcal{M}$ internal to $\mathbf{Psh}(\mathcal{D})$ along a functor $F : \mathcal{C} \to \mathcal{D}$. This restricted model $F^*(\mathcal{M})$ is a first-order model internal to $\mathbf{Psh}(\mathcal{C})$. If $\mathcal{M}$ is seen as a functor $\mathcal{D} \to \mathbf{Mod}_{\mathcal{T}}^{\mathsf{op}}$, then the restriction $F^*(\mathcal{M}) : \mathcal{C} \to \mathbf{Mod}_{\mathcal{T}}^{\mathsf{op}}$ is simply the precomposition $(\mathcal{M} \circ F)$. If $\mathcal{M}$ is seen instead as a finite-limits preserving functor $\mathcal{T}^{\mathsf{fo}} \to \mathbf{Psh}(\mathcal{D})$, then $F^*(\mathcal{M})$ is postcomposition with the inverse image functor $F^* : \mathbf{Psh}(\mathcal{D}) \to \mathbf{Psh}(\mathcal{C})$. These two definitions coincide up to the equivalence of Proposition 6; which is thus natural in the base category.

▶ Remark 7. A more explicit computation of the restriction can be given in the internal language of $\mathbf{Psh}(\mathcal{C})$ using the dependent right adjoint associated to the adjunction $(F_! \dashv F^*)$. When $\mathcal{M}$ is the **Set**- or telescopic contextualization of a higher-order model, then the dependent right adjoint allows for the use of HOAS when working with $F^*(\mathcal{M})$. ⌟

A special case of the restriction is the *externalization* of an internal first-order model.

▶ **Definition 8.** *Let $\mathcal{C}$ be any category with a terminal object $1_{\mathcal{C}}$, and consider the functor $1_{\mathcal{C}} : 1_{\mathbf{Cat}} \to \mathcal{C}$ that selects this terminal object. For any internal first-order model $\mathcal{M}$ in $\mathbf{Psh}(\mathcal{C})$, we have an external first-order model $1_{\mathcal{C}}^*(\mathcal{M})$, called the **externalization** of $\mathcal{M}$.* ⌟

Given any higher-order model $\mathbb{M}$ in $\mathbf{Psh}(\mathcal{C})$, we can construct the externalization $1_{\mathcal{C}}^*(\mathbf{Tele}_{\mathbb{M}})$ of its telescopic first-order model. Up to isomorphism, all external contextual first-order models arise as the externalization of a telescopic contextualization.

▶ **Lemma 9.** *Let $\mathcal{C}$ be an external first-order model, with $\mathbb{C}$ its underlying internal higher-order model. Then $1_{\mathcal{C}}^*(\mathsf{Tele}_{\mathbb{C}})$ is the contextual core of $\mathcal{C}$.*

Proof. See Corollary 24. ◀

In particular, since the initial model $\mathcal{S}$ is contextual, the externalization $1_{\mathcal{S}}^*(\mathsf{Tele}_{\mathbb{S}})$ of its telescopic contextualization is isomorphic to $\mathcal{S}$.

We can also construct the externalization $1_{\mathcal{C}}^*(\mathbf{Set}_{\mathbb{C}})$ of the internal **Set**-contextualization of an higher-order model $\mathbb{C}$. The underlying category of $1_{\mathcal{C}}^*(\mathbf{Set}_{\mathbb{C}})$ is the category of presheaves over $\mathcal{C}$ (restricted to some universe level); and $1_{\mathcal{C}}^*(\mathbf{Set}_{\mathbb{C}})$ is an external model of two-level type theory (its underlying category is the restriction of $\mathbf{Psh}(\mathcal{C})$ to some universe level). Recall that there is, internally to $\mathbf{Psh}(\mathcal{C})$, a morphism $\lfloor - \rfloor : \mathbf{Tele}_{\mathbb{C}} \to \mathbf{Set}_{\mathbb{C}}$ of first-order models. This morphism can also be externalized, giving an external morphism $1_{\mathcal{C}}^*(\lfloor - \rfloor) : 1_{\mathcal{C}}^*(\mathsf{Tele}_{\mathbb{C}}) \to 1_{\mathcal{C}}^*(\mathbf{Set}_{\mathbb{C}})$ of first-order models. When $1_{\mathcal{C}}^*(\mathsf{Tele}_{\mathbb{C}}) \cong \mathcal{C}$, this is a simple construction of the embedding of $\mathcal{C}$ into the presheaf model of two-level type theory.

3 Displayed higher-order models

3.1 Motives and methods

We now define the notion of displayed higher-order model, which collects the motives and methods of induction principles. One could expect that a displayed higher-order model would be displayed over a base higher-order model. We instead define the notion of displayed higher-order model over a base first-order model; it is always possible to turn higher-order models into first-order models using a contextualization, but not every first-order model arises in this way.

▶ **Definition 10.** *Let $\mathcal{M}$ be a first-order model of $\mathcal{T}$. A* ***displayed higher-order model*** $\mathbb{M}^\bullet$ *over $\mathcal{M}$ consists of the following data:*

$$\begin{aligned}
&\mathsf{Ty}^\bullet\ \ : \mathcal{M}.\mathsf{Ty}(1_{\mathcal{M}}) \to \mathrm{Set},\\
&\mathsf{Tm}^\bullet : \forall (A : \mathcal{M}.\mathsf{Ty}(1_{\mathcal{M}}))\ (A^\bullet : \mathsf{Ty}^\bullet(A)) \to \mathcal{M}.\mathsf{Tm}(1_{\mathcal{M}}, A) \to \mathrm{Set},\\
&\Pi^\bullet\ \ : \forall (A : \mathcal{M}.\mathsf{Ty}(1_{\mathcal{M}}))\ (A^\bullet : \mathsf{Ty}^\bullet(A))\\
&\qquad (B : \mathcal{M}.\mathsf{Ty}(1_{\mathcal{M}}.A))\ (B^\bullet : \forall (a : \mathcal{M}.\mathsf{Tm}(1_{\mathcal{M}}, A))(a^\bullet : \mathsf{Tm}^\bullet(A^\bullet, a)) \to \mathsf{Ty}^\bullet(B[a]))\\
&\qquad \to \mathsf{Ty}^\bullet(\Pi(A, B)),\\
&\mathsf{app}^\bullet : \forall A\ A^\bullet\ B\ B^\bullet\ (f : \mathcal{M}.\mathsf{Tm}(1_{\mathcal{M}}, \Pi(A, B)))\ (f^\bullet : \mathsf{Tm}^\bullet(\Pi^\bullet(A^\bullet, B^\bullet), f))\\
&\qquad (a : \mathcal{M}.\mathsf{Tm}(1_{\mathcal{M}}, A))\ (a^\bullet : \mathsf{Tm}^\bullet(A^\bullet, a))\\
&\qquad \to \mathsf{Tm}^\bullet(B^\bullet(a^\bullet), \mathsf{app}(f, a)),\\
&\mathsf{lam}^\bullet : \forall A\ A^\bullet\ B\ B^\bullet\ (b : \mathcal{M}.\mathsf{Tm}(1_{\mathcal{M}}.(a : A), B[a]))\\
&\qquad (b^\bullet : \forall (a : \mathcal{M}.\mathsf{Tm}(1_{\mathcal{M}}, A))(a^\bullet : \mathsf{Tm}^\bullet(A^\bullet, a)) \to \mathsf{Tm}^\bullet(B^\bullet(a^\bullet), b[a]))\\
&\qquad \to \mathsf{Tm}^\bullet(\Pi^\bullet(A^\bullet, B^\bullet), \mathsf{lam}(b)),
\end{aligned}$$

such that the following equalities hold:

$$\mathsf{app}^\bullet(\mathsf{lam}^\bullet(b^\bullet), a^\bullet) = b^\bullet(a^\bullet), \qquad \mathsf{lam}^\bullet(\lambda a^\bullet \mapsto \mathsf{app}^\bullet(f^\bullet, a^\bullet)) = f^\bullet. \qquad \lrcorner$$

Most of the components of a displayed higher-order model only depend on the closed types and terms of $\mathcal{M}$; only the binders need to refer to open types and terms.

Note that the data of a displayed higher-order model over the terminal first-order model is equivalent to the data of a non-displayed higher-order model.

3.2 Displayed contextualization

Given any displayed higher-order model $\mathbb{M}^\bullet$ over $\mathcal{M}$, we construct a displayed first-order model over $\mathcal{M}$. This construction is a displayed generalization of the **Set**-contextualization.

The underlying displayed category of this construction is the *Sierpinski cone*, or scone, of $\mathcal{M}$. The scone of a category $\mathcal{C}$ with a terminal object is the comma category $(\mathbf{Set} \downarrow \Gamma_{\mathcal{C}})$, where $\Gamma_{\mathcal{C}} : \mathcal{C} \to \mathbf{Set}$ is the global section functor $\Gamma_{\mathcal{C}} = \mathcal{C}(1_{\mathcal{C}}, -)$.

▶ **Construction 11** (Displayed contextualization). *Fix a displayed higher-order model $\mathbb{M}^\bullet$ over a first-order model $\mathcal{M}$. We construct a displayed first-order model $\mathbf{Scone}_{\mathbb{M}^\bullet}$ over $\mathcal{M}$, called the* ***displayed contextualization*** *of $\mathbb{M}^\bullet$.*

- *An object of $\mathbf{Scone}_{\mathbb{M}^\bullet}$ over $\Gamma \in \mathcal{M}$ is a family*

$$\Gamma^\dagger : \mathcal{M}(1_{\mathcal{M}}, \Gamma) \to \mathrm{Set}$$

over the global elements (i.e. closing substitutions) of Γ.

- *A morphism of* $\mathbf{Scone}_{\mathbb{M}^\bullet}$ *from* $\Gamma^\dagger$ *to* $\Delta^\dagger$ *over a base morphism* $f : \mathcal{M}(\Gamma, \Delta)$ *is a family*

  $$f^\dagger : \forall(\gamma : \mathcal{M}(1_{\mathcal{M}}, \Gamma)) \to \Gamma^\dagger(\gamma) \to \Delta^\dagger(f \circ \gamma).$$

  The identity displayed morphism is given by

  $$\mathsf{id}^\dagger \triangleq \lambda\gamma\ \gamma^\bullet \mapsto \gamma^\bullet,$$

  whereas the composition of two displayed morphisms $f^\dagger$ *and* $g^\dagger$ *is*

  $$f^\dagger \circ^\dagger g^\dagger \triangleq \lambda\gamma\ \gamma^\bullet \mapsto f^\dagger(g^\dagger(\gamma^\bullet)).$$

- *A type of* $\mathbf{Scone}_{\mathbb{M}^\bullet}$ *over an object* $\Gamma^\dagger$ *and a type* $A : \mathcal{M}.\mathsf{Ty}(\Gamma)$ *is a function*

  $$A^\dagger : \forall\gamma\ (\gamma^\dagger : \Gamma^\dagger(\gamma)) \to \mathsf{Ty}^\bullet(A[\gamma]).$$

  The restriction of $A^\dagger$ *along a displayed morphism* $f^\dagger$ *is*

  $$A^\dagger[f^\dagger] \triangleq \lambda\gamma^\bullet \mapsto A^\dagger(f^\dagger(\gamma^\bullet)).$$

- *A term of* $\mathbf{Scone}_{\mathbb{M}^\bullet}$ *of type* $A^\dagger$ *over an object* $\Gamma^\dagger$ *and a term* $a : \mathcal{M}.\mathsf{Tm}(\Gamma, A)$ *is a function*

  $$a^\dagger : \forall\gamma\ (\gamma^\bullet : \Gamma^\dagger(\gamma)) \to \mathsf{Tm}^\bullet(A^\dagger(\gamma^\bullet), a[\gamma]).$$

  The restriction of $a^\dagger$ *along a displayed morphism* $f^\dagger$ *is*

  $$a^\dagger[f^\dagger] \triangleq \lambda\gamma^\bullet \mapsto a^\dagger(f^\dagger(\gamma^\bullet)).$$

- *The empty displayed context is the family*

  $$1^\dagger \triangleq \lambda\_ \mapsto \mathbf{1}.$$

- *The extension of a displayed context* $\Gamma^\dagger$ *by a displayed type* $A^\dagger$ *is the family*

  $$(\Gamma^\dagger.A^\dagger) \triangleq \lambda\langle\gamma, a\rangle \mapsto (\gamma^\bullet : \Gamma^\dagger(\gamma)) \times \mathsf{Tm}^\bullet(A^\dagger(\gamma^\bullet), a).$$

- *All type- and term- formers are defined pointwise using the corresponding component of* $\mathbb{M}^\bullet$*:*

  $$\begin{aligned} \Pi^\dagger(A^\dagger, B^\dagger) &\triangleq \lambda\gamma^\bullet \mapsto \Pi^\bullet(A^\dagger(\gamma^\bullet), \lambda a^\bullet \mapsto B^\dagger(\gamma^\bullet, a^\bullet)), \\ \mathsf{app}^\dagger(f^\dagger, a^\dagger) &\triangleq \lambda\gamma^\bullet \mapsto \mathsf{app}^\bullet(f^\dagger(\gamma^\bullet), a^\dagger(\gamma^\bullet)), \\ \mathsf{lam}^\dagger(b^\dagger) &\triangleq \lambda\gamma^\bullet \mapsto \mathsf{lam}^\bullet(\lambda a^\bullet \mapsto b^\dagger(\gamma^\bullet, a^\bullet)). \end{aligned}$$

- *The* β*- and* η*-rules hold as a consequence of the* β*- and* η*-rules of* $\mathbb{M}^\bullet$*.*
- *All naturality conditions are trivial.* ⌟

Note that when $\mathbb{M}$ is a higher-order model seen as a displayed higher-order model over the terminal first-order model, then $\mathbf{Scone}_{\mathbb{M}}$ is equivalent to $\mathbf{Set}_{\mathbb{M}}$.

3.3 Sections of a displayed higher-order model

The notion of displayed higher-order model corresponds to the motives and methods of an induction principle. We now define the notion of section of a displayed higher-order model, corresponding to the result of applying an induction principle: it is simply defined as a section of the displayed contextualization.

▶ **Definition 12.** *A **section** of a displayed higher-order model* $\mathbb{M}^\bullet$ *is a section* $[\![-]\!]$ *of its displayed contextualization* $\mathbf{Scone}_{\mathbb{M}^\bullet}$ *(in* $\mathbf{Mod}_{\mathcal{T}}$*).* ⌟

The definition of section of a displayed higher-order model $\mathbb{M}^\bullet$ over $\mathcal{M}$ can be unfolded to the following components:

- For every object $\Gamma : \mathcal{M}$, a family

$$[\![\Gamma]\!] : \mathcal{M}(1_{\mathcal{M}}, \Gamma) \to \mathbf{Set}$$

of *environments*.
- For every morphism $f : \mathcal{M}(\Gamma, \Delta)$, a family

$$[\![f]\!] : \forall \gamma \to [\![\Gamma]\!](\gamma) \to [\![\Delta]\!](f \circ \gamma)$$

of maps between environments.
- For every type $A : \mathcal{M}.\mathsf{Ty}(\Gamma)$, a family

$$[\![A]\!] : \forall \gamma \ (\gamma^\bullet : [\![\Gamma]\!](\gamma)) \to \mathsf{Ty}^\bullet(A[\gamma])$$

of displayed types over closures of A.
- For every term $a : \mathcal{M}.\mathsf{Tm}(\Gamma, A)$, a family

$$[\![a]\!] : \forall \gamma \ (\gamma^\bullet : [\![\Gamma]\!](\gamma)) \to \mathsf{Tm}^\bullet([\![A]\!](\gamma^\bullet), a[\gamma])$$

of displayed terms over closures of a.
- Subject to functoriality and naturality equations:

$$\begin{aligned}
&[\![\mathsf{id}]\!](\gamma^\bullet) = \gamma^\bullet,\\
&[\![f \circ g]\!](\gamma^\bullet) = [\![f]\!]([\![g]\!](\gamma^\bullet)),\\
&[\![A[f]]\!](\gamma^\bullet) = [\![A]\!]([\![f]\!](\gamma^\bullet)),\\
&[\![a[f]]\!](\gamma^\bullet) = [\![A]\!]([\![f]\!](\gamma^\bullet)).
\end{aligned}$$

- Such that context extensions are preserved:

$$\begin{aligned}
&[\![1_{\mathcal{M}}]\!](\star) = \{\star\},\\
&[\![\Gamma.A]\!](\gamma, a) = (\gamma^\bullet : [\![\Gamma]\!](\gamma)) \times (a^\bullet : \mathsf{Tm}^\bullet([\![A]\!](\gamma^\bullet), a)),\\
&[\![\lambda\gamma \mapsto (\delta(\gamma), a(\gamma))]\!](\gamma) = ([\![\delta]\!](\gamma), [\![a]\!](\gamma)).
\end{aligned}$$

- With computation rules for every type and term former:

$$\begin{aligned}
&[\![\lambda\gamma \mapsto \Pi(A(\gamma), \lambda a \mapsto B(\gamma, a))]\!](\gamma^\bullet) = \Pi^\bullet([\![A]\!](\gamma^\bullet), \lambda a^\bullet \mapsto [\![B]\!](\gamma^\bullet)),\\
&[\![\lambda\gamma \mapsto \mathsf{app}(f(\gamma), a(\gamma))]\!](\gamma^\bullet) = \mathsf{app}^\bullet([\![f]\!](\gamma^\bullet), [\![a]\!](\gamma^\bullet)),\\
&[\![\lambda\gamma \mapsto \mathsf{lam}(b(\gamma))]\!](\gamma^\bullet) = \mathsf{lam}^\bullet(\lambda a^\bullet \mapsto [\![b]\!](\gamma^\bullet, a^\bullet)).
\end{aligned}$$

When x is a closed type or term of $\mathcal{M}$, we write $[\![x]\!]$ for the interpretation $[\![x]\!](\star)$ of x in the empty environment. We may use underlined names to distinguish the variable of open terms. For instance, we may write $[\![\mathsf{app}(\underline{f}, \underline{a})]\!][\underline{f} \mapsto f', \underline{a} \mapsto a']$ instead of $[\![\lambda(f, a) \mapsto \mathsf{app}(f, a)]\!](f', a')$.

▶ Remark 13. Let $\mathbb{S}^\bullet$ be a displayed higher-order model over the first-order model $F^*(\mathsf{Tele}_{\mathbb{S}})$ in $\mathbf{Psh}(\mathcal{C})$, for some functor $F : \mathcal{C} \to \mathcal{S}$. The displayed contextualization $\mathbf{Scone}_{\mathbb{S}^\bullet}$ is a displayed first-order model over $F^*(\mathsf{Tele}_{\mathbb{S}})$. Then its externalization $1_{\mathcal{C}}^*(\mathbf{Scone}_{\mathbb{S}^\bullet})$ is an external displayed first-order model lying over $1_{\mathcal{C}}^*(F^*(\mathsf{Tele}_{\mathbb{S}})) = 1_{\mathcal{S}}^*(\mathsf{Tele}_{\mathbb{S}})$. Up to the isomorphism $1_{\mathcal{S}}^*(\mathsf{Tele}_{\mathbb{S}}) \cong \mathcal{S}$, the externalized $\mathbf{Scone}$-contextualization $1_{\mathcal{C}}^*(\mathbf{Scone}_{\mathbb{S}^\bullet})$ coincides with gluing. Its underlying category is the comma category $(\mathcal{S} \downarrow N_F)$, where $N_F : \mathcal{S} \to \mathbf{Psh}(\mathcal{C})$ is the nerve functor $\mathcal{S} \xrightarrow{\text{よ}} \mathbf{Psh}(\mathcal{S}) \xrightarrow{F^*} \mathbf{Psh}(\mathcal{C})$. ⌟

4 Example: canonicity proof

As a first example of a relative induction principle and its application, we prove canonicity for $\mathcal{T}$ extended with booleans (given by a type former Bool with constructors true and false and a dependent eliminator $\mathsf{elim}_{\mathsf{Bool}}$ with two computation rules).

We use the induction principle relative to the functor $1_{\mathcal{S}} : 1_{\mathbf{Cat}} \to \mathcal{S}$ that selects the terminal object in the syntax $\mathcal{S}$. It turns out that proving this specific relative induction principle is trivial.

▶ **Theorem 14** (Induction principle for $\mathbb{S}$ relative to $1_{\mathcal{S}} : 1_{\mathbf{Cat}} \to \mathcal{S}$). *Let $\mathbb{S}^\bullet$ be a displayed higher-order model over the initial model $\mathcal{S}$, or equivalently over $1_{\mathcal{S}}^*(\mathsf{Tele}_{\mathbb{S}})$. Then $\mathbf{Scone}_{\mathbb{S}^\bullet}$ admits a section $[\![-]\!]$ over $\mathcal{S}$.*

Proof. By initiality of $\mathcal{S}$. ◀

We now construct the displayed higher-order model $\mathbb{S}^\bullet$ over $1_{\mathcal{S}}^*(\mathsf{Tele}_{\mathbb{S}})$ that will be used to prove canonicity. A displayed type $A^\bullet$ over a closed type $A : \mathcal{S}.\mathsf{Ty}(1_{\mathcal{S}})$ is a Set-valued logical predicate over the closed terms of type A:

$$\mathsf{Ty}^\bullet(A) \triangleq \mathcal{S}.\mathsf{Tm}(1_{\mathcal{S}}, A) \to \mathrm{Set}.$$

A displayed term $a^\bullet$ of type $A^\bullet$ over a closed term $a : \mathcal{S}.\mathsf{Tm}(1_{\mathcal{S}}, A)$ is an element of the logical predicate $A^\bullet$ evaluated at a:

$$\mathsf{Tm}^\bullet(A^\bullet, a) \triangleq A^\bullet(a).$$

Given logical predicates $A^\bullet$ and $B^\bullet$, the logical predicate $\Pi^\bullet(A^\bullet, B^\bullet)$ expresses the fact that functions $f : \mathcal{S}.\mathsf{Tm}(1_{\mathcal{S}}, \Pi(A, B))$ should preserve the logical predicates.

$$\begin{aligned}
\Pi^\bullet(A^\bullet, B^\bullet) &\triangleq \lambda(f : \mathcal{S}.\mathsf{Tm}(1_{\mathcal{S}}, \Pi(A, B))) \mapsto (\forall a\ a^\bullet \to B^\bullet(a^\bullet, \mathsf{app}(f, a))),\\
\mathsf{app}^\bullet(f^\bullet, a^\bullet) &\triangleq f^\bullet(a^\bullet),\\
\mathsf{lam}^\bullet(b^\bullet) &\triangleq \lambda a^\bullet \mapsto b^\bullet(a^\bullet).
\end{aligned}$$

It is easy to check that the displayed β- and η-rules hold. The logical predicate $\mathsf{Bool}^\bullet : \mathcal{S}.\mathsf{Tm}(1_{\mathcal{S}}, \mathsf{Bool}) \to \mathrm{Set}$ is defined as an inductive family with two constructors $\mathsf{true}^\bullet : \mathsf{Bool}^\bullet(\mathsf{true})$ and $\mathsf{false}^\bullet : \mathsf{Bool}^\bullet(\mathsf{false})$. The displayed eliminator $\mathsf{elim}^\bullet_{\mathsf{Bool}}$ is defined using the elimination principle of $\mathsf{Bool}^\bullet$ and the displayed β-laws hold. This concludes the definition of all components of $\mathbb{S}^\bullet$.

▶ **Theorem 15.** *The initial model $\mathbb{S}$ satisfies canonicity: any closed boolean term $b : \mathcal{S}.\mathsf{Tm}(1_{\mathcal{S}}, \mathsf{Bool})$ is canonical, i.e. equal to exactly one of* true *or* false*.*

Proof. By the relative induction principle Theorem 14, the displayed higher-order model $\mathbb{S}^\bullet$ admits a section $[\![-]\!]$. Now given a closed boolean term $b : \mathcal{S}.\mathsf{Tm}(1_{\mathcal{S}}, \mathsf{Bool})$, we have $[\![b]\!] : \mathsf{Tm}^\bullet([\![\mathsf{Bool}]\!], b)$. By the computation rule of the section for Bool, $\mathsf{Tm}^\bullet([\![\mathsf{Bool}]\!], b) = \mathsf{Bool}^\bullet(b)$. Thus, $[\![b]\!] : \mathsf{Bool}^\bullet(b)$ witnesses the fact that b is canonical.

Since $[\![\mathsf{true}]\!] = \mathsf{true}^\bullet$, $[\![\mathsf{false}]\!] = \mathsf{false}^\bullet$ and $\mathsf{true}^\bullet \neq \mathsf{false}^\bullet$, we know that $\mathsf{true} \neq \mathsf{false}$. ◀

Note that in the canonicity proof, we have not needed to evaluate the section $[\![-]\!]$ on non-closed types or terms. Evaluating the section on open types and terms is usually only needed when encountering binders: the evaluation of the section on a closed binder depends on the evaluation of the section on an open type or term. The following is an example of the computation of the evaluation of the section $[\![-]\!]$ on the application of the boolean negation function $\mathsf{lam}(\lambda b \mapsto \mathsf{elim}_{\mathsf{Bool}}(\mathsf{Bool}, \mathsf{false}, \mathsf{true}, b))$ to true.

$$\begin{aligned}
&[\![\mathsf{app}(\mathsf{lam}(\lambda b \mapsto \mathsf{elim}_{\mathsf{Bool}}(\mathsf{Bool}, \mathsf{false}, \mathsf{true}, b)), \mathsf{true})]\!] \\
&\quad = [\![\mathsf{lam}(\lambda b \mapsto \mathsf{elim}_{\mathsf{Bool}}(\mathsf{Bool}, \mathsf{false}, \mathsf{true}, b))]\!]([\![\mathsf{true}]\!]) \\
&\quad = (\lambda b^\bullet \mapsto [\![\mathsf{elim}_{\mathsf{Bool}}(\mathsf{Bool}, \mathsf{false}, \mathsf{true}, \underline{b})]\!][\underline{b} \mapsto b^\bullet])(\mathsf{true}^\bullet) \\
&\quad = [\![\mathsf{elim}_{\mathsf{Bool}}(\mathsf{Bool}, \mathsf{false}, \mathsf{true}, \underline{b})]\!][\underline{b} \mapsto \mathsf{true}^\bullet] \\
&\quad = \mathsf{elim}^\bullet_{\mathsf{Bool}}(\mathsf{Bool}^\bullet, \mathsf{false}^\bullet, \mathsf{true}^\bullet, \mathsf{true}^\bullet) \\
&\quad = \mathsf{false}^\bullet.
\end{aligned}$$

5 Example: normalization proof

In this section we prove normalization for the initial model $\mathcal{S}$ of $\mathcal{T}$ using an induction principle relative to $F : \mathbf{Ren}_{\mathcal{S}} \to \mathcal{S}$, where $\mathbf{Ren}_{\mathcal{S}}$ is the category of renamings of $\mathcal{S}$, i.e. the category whose morphisms are the substitutions of $\mathcal{S}$ that are built out of variables. We use the alternative definition from [10] of $\mathbf{Ren}_{\mathcal{S}}$ as the initial object in a category of first-order renaming algebras.

5.1 The category of renamings

▶ **Definition 16.** *Let $\mathcal{C}$ be a first-order model of $\mathcal{T}$. A **higher-order renaming algebra** $\mathbb{C}$ over $\mathcal{C}$ consists of:*

$$\begin{aligned}
&\mathbb{C}.\mathsf{Var} : \mathcal{C}.\mathsf{Ty}(1_{\mathcal{C}}) \to \mathcal{U}, \\
&\mathbb{C}.\mathsf{var}\ : (A : \mathcal{C}.\mathsf{Ty}(1_{\mathcal{C}})) \to \mathbb{C}.\mathsf{Var}(A) \to \mathcal{C}.\mathsf{Tm}(1_{\mathcal{C}}, A).
\end{aligned}$$
⌟

▶ **Definition 17.** *Let $\mathcal{D}$ be a first-order model of $\mathcal{T}$. A **first-order renaming algebra** over $\mathcal{D}$ is a category $\mathcal{C}$ with a terminal object along with a functor $F : \mathcal{C} \to \mathcal{D}$ that preserves the terminal object and with the structure of a global higher-order renaming algebra $\mathbb{C}$ over $F^*(\mathsf{Tele}_{\mathbb{D}})$ such that $\mathbb{C}.\mathsf{Var}$ is locally representable and $\mathbb{C}.\mathsf{var}$ strictly preserves context extensions.* ⌟

Equivalently, a first-order renaming algebra over $\mathcal{D}$ is a CwF $\mathcal{C}$ together with a CwF morphism $F : \mathcal{C} \to \mathcal{D}$ whose action on types is bijective. (There is a cofibrantly generated factorization system with such morphisms as its right class. The renaming algebras are the objects in the left class.) The category of first-order renaming algebras over $\mathcal{S}$ is locally finitely presentable, and there is an initial first-order renaming algebra $\mathbf{Ren}_{\mathcal{S}}$. The category $\mathbf{Ren}_{\mathcal{S}}$ is the category of renamings of $\mathcal{S}$; in this section we write F for the functor $F : \mathbf{Ren}_{\mathcal{S}} \to \mathcal{S}$.

5.2 Relative induction principle

We pose $\mathcal{S}_F \triangleq F^*(\mathbf{Tele}_{\mathbb{S}})$; $\mathcal{S}_F$ is an internal first-order model in $\mathbf{Psh}(\mathbf{Ren}_{\mathcal{S}})$.

▶ **Theorem 18** (Induction principle for $\mathbb{S}$ relative to $F : \mathbf{Ren}_{\mathcal{S}} \to \mathcal{S}$)**.** *Let $\mathbb{S}^\bullet$ be a displayed higher-order model over $\mathcal{S}_F$. Given the additional data of*

$$\mathsf{var}^\bullet : \forall (A : \mathcal{S}_F.\mathsf{Ty}(1_{\mathcal{S}_F}))\ (A^\bullet : \mathsf{Ty}^\bullet(A))\ (x : \mathsf{Var}(A)) \to \mathsf{Tm}^\bullet(A^\bullet, \mathsf{var}(x)),$$

the displayed contextualization $\mathbf{Scone}_{\mathbb{S}^\bullet}$ admits a section $[\![-]\!]$ that satisfies the additional computation rule $[\![\mathsf{var}_A(x)]\!] = \mathsf{var}^\bullet([\![A]\!], x)$.

Proof. See Appendix A.4. ◀

Note that $\mathsf{var}_A(x)$ is always a closed term of $\mathcal{S}_F$, thus $[\![\mathsf{var}_A(x)]\!]$ does not depend on any environment.

5.3 Normal forms

Neutrals and normal forms are defined internally to $\mathbf{Psh}(\mathbf{Ren}_{\mathcal{S}})$, as inductive families

$$\mathsf{Ne}, \mathsf{Nf} : \forall (A : \mathcal{S}_F.\mathsf{Ty}(1_{\mathcal{S}_F}))\ (a : \mathcal{S}_F.\mathsf{Tm}(1_{\mathcal{S}_F}, A)) \to \mathrm{Set},$$

generated by the following constructors:

$$\begin{aligned}
\mathsf{var}^{\mathsf{ne}} &: (x : \mathsf{Var}(A)) \to \mathsf{Ne}_A(\mathsf{var}(x)),\\
\mathsf{app}^{\mathsf{ne}} &: \mathsf{Ne}_{\Pi(A,B)}(f) \to \mathsf{Nf}_A(a) \to \mathsf{Ne}_{B[a]}(\mathsf{app}(f, a)),\\
\mathsf{lam}^{\mathsf{nf}} &: ((a : \mathsf{Var}(A)) \to \mathsf{Nf}_{B[a]}(b[a])) \to \mathsf{Nf}_{\Pi(A,B)}(\mathsf{lam}(b)).
\end{aligned}$$

The goal of normalization is to prove that every term has a unique normal form:

$$\forall (A : \mathcal{S}_F.\mathsf{Ty}(1_{\mathcal{S}_F}))\ (a : \mathcal{S}_F.\mathsf{Tm}(1_{\mathcal{S}_F}, A)) \to \mathrm{isContr}(\mathsf{Nf}_A(a)).$$

This is accomplished in two steps. First a normalization function is obtained from the relative induction principle, witnessing the existence of normal forms. Then the uniqueness of normal forms is derived from the stability of the normalization; a fact that is proven by mutual induction on neutrals and normal forms.

5.4 Normalization displayed model

We now construct the normalization displayed higher-order model $\mathbb{S}^\bullet$ over $\mathcal{S}_F$.

A displayed type $A^\bullet : \mathsf{Ty}^\bullet(A)$ over a type $A : \mathcal{S}_F.\mathsf{Ty}(1_{\mathcal{S}_F})$ is a triple $(A^\bullet_p, A^\bullet_u, A^\bullet_q)$ consisting of a logical *predicate* $A^\bullet_p : \mathcal{S}_F.\mathsf{Tm}(1_{\mathcal{S}_F}, A) \to \mathrm{Set}$, over the terms of type A, valued in the universe of sets; an *unquoting* (or reflection) function $A^\bullet_u : (a : \mathcal{S}_F.\mathsf{Tm}(1_{\mathcal{S}_F}, A)) \to \mathsf{Ne}_A(a) \to A^\bullet_p(a)$, witnessing the fact that any neutral term satisfies the logical predicate $A^\bullet_p$; a *quoting* (or reification) function $A^\bullet_q : (a : \mathcal{S}_F.\mathsf{Tm}(1_{\mathcal{S}_F}, A)) \to A^\bullet_p(a) \to \mathsf{Nf}_A(a)$, witnessing the fact a term satisfying the logical predicate $A^\bullet_p$ admits a normal form. A displayed term $a^\bullet$ of type $A^\bullet$ over a term $a : \mathcal{S}_F.\mathsf{Tm}(1_{\mathcal{S}_F}, A)$ is an element of $A^\bullet_p(a)$: $\mathsf{Tm}^\bullet(A^\bullet, a) \triangleq A^\bullet_p(a)$. The logical predicate for Π-types is defined in the same way as in the canonicity model.

$$\Pi^\bullet_p(A^\bullet, B^\bullet)(f) \triangleq (\forall a\ a^\bullet \to B^\bullet_p(a^\bullet, \mathsf{app}(f, a))).$$

The unquoting function relies on the unquoting function of the codomain and the quoting function of the domain.

$$\Pi^\bullet_u(A^\bullet, B^\bullet)(f^{\mathsf{ne}}) \triangleq \lambda a^\bullet \mapsto B^\bullet_u(a^\bullet, \mathsf{app}^{\mathsf{ne}}(f^{\mathsf{ne}}, A^\bullet_q(a^\bullet))).$$

The quoting function says that any element of a Π-type is a lambda, as implied by the η-rule. It relies on the quoting function of the codomain and the unquoting function of the domain, and on the fact that every variable is neutral.

$$\Pi^\bullet_q(A^\bullet, B^\bullet)(f^\bullet) \triangleq \mathsf{lam}^{\mathsf{nf}}(\lambda a \mapsto \mathsf{let}\ a^\bullet = A^\bullet_u(\mathsf{var}^{\mathsf{ne}}(a))\ \mathsf{in}\ B^\bullet_q(a^\bullet, f^\bullet(a^\bullet))).$$

This completes the definition of the displayed higher-order model $\mathbb{S}^\bullet$. It remains to check the last hypothesis of the relative induction principle:

$$\begin{aligned}&\mathsf{var}^\bullet \quad : \ \forall A\ A^\bullet\ (x : \mathsf{Var}(A)) \to \mathsf{Tm}^\bullet(A^\bullet, \mathsf{var}(x)),\\ &\mathsf{var}^\bullet(A^\bullet, x) \triangleq A^\bullet_u(\mathsf{var}^{\mathsf{ne}}(x)).\end{aligned}$$

By the relative induction principle (Theorem 18), we obtain a section $[\![-]\!]$ of $\mathbf{Scone}_{\mathbb{S}^\bullet}$. We can then define the normalization function as follows:

$$\begin{aligned}&\mathsf{norm} \quad : \ \forall A\ (a : \mathcal{S}_F.\mathsf{Tm}(1_{\mathcal{S}_F}, A)) \to \mathsf{Nf}_A(a),\\ &\mathsf{norm}_A(a) \triangleq [\![A]\!]_q([\![a]\!]).\end{aligned}$$

5.5 Stability of normalization and uniqueness of normal forms

Finally, we show the uniqueness of normal forms following [26]: we prove that normalization is stable, that is every normal form for a term a is equal to the normal form of a obtained from the normalization function. As the proof relies on most of the computation rules of the section $[\![-]\!]$, it is a good example of computations with a section.

▶ **Lemma 19** (Stability). *Given any normal form* $a^{\mathsf{nf}} : \mathsf{Nf}_A(a)$*, we have* $a^{\mathsf{nf}} = \mathsf{norm}_A(a)$*.*

Proof. We prove the following two facts, by mutual induction on neutrals and normal forms:

$$(a^{\mathsf{ne}} : \mathsf{Ne}_A(a)) \to [\![a]\!] = [\![A]\!]_u(a^{\mathsf{ne}}), \qquad (a^{\mathsf{nf}} : \mathsf{Nf}_A(a)) \to a^{\mathsf{nf}} = [\![A]\!]_q([\![a]\!]).$$

Each case involves some of the computation rules of $[\![-]\!]$.

Case $a^{\mathsf{ne}} = \mathsf{var}^{\mathsf{ne}}(A, x)$

$$\begin{aligned}&[\![\mathsf{var}_A(x)]\!]\\ &\quad = \mathsf{var}^\bullet([\![A]\!], x) && \text{(by the computation rule for } [\![\mathsf{var}(-)]\!])\\ &\quad = [\![A]\!]_u(a^{\mathsf{ne}}). && \text{(by definition of } \mathsf{var}^\bullet)\end{aligned}$$

Case $a^{\mathsf{ne}} = \mathsf{app}^{\mathsf{ne}}(f^{\mathsf{ne}}, a^{\mathsf{nf}})$

$$\begin{aligned}&[\![\mathsf{app}(f, a)]\!]\\ &\quad = \mathsf{app}^\bullet([\![f]\!], [\![a]\!]) && \text{(by the computation rule for } [\![\mathsf{app}(-)]\!])\\ &\quad = [\![f]\!]([\![a]\!]) && \text{(by definition of } \mathsf{app}^\bullet)\\ &\quad = [\![\Pi(A, B)]\!]_u(f^{\mathsf{ne}}, [\![a]\!]) && \text{(by the induction hypothesis for } f^{\mathsf{ne}})\\ &\quad = [\![B[a]]\!]_u(\mathsf{app}^{\mathsf{ne}}(f^{\mathsf{ne}}, a^{\mathsf{nf}})).\\ &&& \text{(by definition of } \Pi^\bullet_u \text{ and the induction hypothesis for } a^{\mathsf{nf}})\end{aligned}$$

Case $a^{\mathsf{nf}} = \mathsf{lam}^{\mathsf{nf}}(b^{\mathsf{nf}})$

$$\begin{aligned}
&[\![\Pi(A,B)]\!]_q([\![\mathsf{lam}(b)]\!])\\
&\quad = \Pi^\bullet_q([\![A]\!], \lambda a^\bullet \mapsto [\![B(\underline{a})]\!][\underline{a} \mapsto a^\bullet])(\lambda a^\bullet \mapsto [\![b(\underline{a})]\!][\underline{a} \mapsto a^\bullet])\\
&\qquad\qquad \text{(by the computation rules for } [\![\Pi(-)]\!] \text{ and } [\![\mathsf{lam}]\!])\\
&\quad = \mathsf{lam}^{\mathsf{nf}}(\lambda a \mapsto \mathsf{let}\ a^\bullet = [\![A]\!]_u(\mathsf{var}^{\mathsf{ne}}(a))\ \mathsf{in}\ ([\![B(\underline{a})]\!][\underline{a} \mapsto a^\bullet])_q([\![b(\underline{a})]\!][\underline{a} \mapsto a^\bullet])\\
&\qquad\qquad \text{(by definition of } \Pi^\bullet_q)\\
&\quad = \mathsf{lam}^{\mathsf{nf}}(\lambda a \mapsto ([\![B(\underline{a})]\!][\underline{a} \mapsto [\![\mathsf{var}(a)]\!]])_q([\![b(\underline{a})]\!][\underline{a} \mapsto [\![\mathsf{var}(a)]\!]]))\\
&\qquad\qquad \text{(by the computation rule for } [\![\mathsf{var}(a)]\!])\\
&\quad = \mathsf{lam}^{\mathsf{nf}}(\lambda a \mapsto [\![B[\mathsf{var}(a)]]\!]_q([\![b[\mathsf{var}(a)]]\!])) \qquad \text{(by the naturality of } [\![-]\!])\\
&\quad = \mathsf{lam}^{\mathsf{nf}}(b^{\mathsf{nf}}). \qquad \text{(by the induction hypothesis for } b^{\mathsf{nf}}) \quad \blacktriangleleft
\end{aligned}$$

6 The category of sections of a displayed first-order model

The last tool that is needed for the proofs of relative induction principles is the *category of sections* of an internal displayed higher-order model. This replaces the use of a displayed inserter in our previous work [10].

Let $\mathcal{M}^\bullet$ be a global internal displayed first-order model over a first-order model $\mathcal{M}$, internally to some presheaf category $\mathbf{Psh}(\mathcal{C})$. We see $\mathcal{M}$ as a functor $\mathcal{M} : \mathcal{C} \to \mathbf{Mod}_{\mathcal{T}}^{\mathsf{op}}$, as justified by Proposition 6. Write $\mathbf{DispMod}_{\mathcal{T}}$ for the external category of a first-order model of $\mathcal{T}$ with a displayed first-order model over it. There is a forgetful functor $U : \mathbf{DispMod}_{\mathcal{T}} \to \mathbf{Mod}_{\mathcal{T}}$. Since the notion of displayed first-order model is also essentially algebraic, we can also view $\mathcal{M}^\bullet$ as a functor $\mathcal{C} \to \mathbf{DispMod}_{\mathcal{T}}^{\mathsf{op}}$ such that $U \circ \mathcal{M}^\bullet = \mathcal{M}$. Similarly, writing $\mathbf{Sect}_{\mathcal{T}}$ for the category of a displayed first-order model of $\mathcal{T}$ with a section, a section of $\mathcal{M}^\bullet$ can be identified with a functor $[\![-]\!] : \mathcal{C} \to \mathbf{Sect}_{\mathcal{T}}^{\mathsf{op}}$ such that $V \circ [\![-]\!] = \mathcal{M}^\bullet$, where V is the forgetful functor $\mathbf{Sect}_{\mathcal{T}} \to \mathbf{DispMod}_{\mathcal{T}}$.

▶ **Definition 20.** *The category* $\mathbf{Sect}_{\mathcal{T}}^{\mathsf{op}}[\mathcal{M}^\bullet]$ *of **sections** of* $\mathcal{M}^\bullet$ *is the pullback (in* $\mathbf{Cat}$*)*

$$\begin{tikzcd}
\mathbf{Sect}_{\mathcal{T}}^{\mathsf{op}}[\mathcal{M}^\bullet] \arrow[r, "{[\![-]\!]_0}"] \arrow[d, "\pi_0"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathbf{Sect}_{\mathcal{T}}^{\mathsf{op}} \arrow[d, "V"] \\
\mathcal{C} \arrow[r, "\mathcal{M}^\bullet"] & \mathbf{DispMod}_{\mathcal{T}}^{\mathsf{op}}
\end{tikzcd}$$

By the universal property of the pullback, the data of a section of $\mathcal{M}^\bullet$ is equivalent to the data of a section of π_0 in $\mathbf{Cat}$. The category $\mathbf{Sect}_{\mathcal{T}}^{\mathsf{op}}[\mathcal{M}^\bullet]$ is itself equipped with a section $[\![-]\!]_0$ of $\pi_0^*(\mathcal{M}^\bullet)$, which is called the *generic section* of $\mathcal{M}^\bullet$.

In order to prove an induction principle such as Theorem 18, we want to use the initiality of $\mathcal{C}$ in some category to obtain section of π_0. For example, when $\mathcal{C}$ is the category of renamings, it suffices to equip $\mathbf{Sect}_{\mathcal{T}}^{\mathsf{op}}[\mathcal{M}^\bullet]$ with the structure of a renaming algebra that is preserved by π_0.

This typically involves lifting the terminal object and context extensions of $\mathcal{C}$ to $\mathbf{Sect}_{\mathcal{T}}^{\mathsf{op}}[\mathcal{M}^\bullet]$. Conditions for the lifting of these finite limits are given in Appendix A.3; the initiality of the syntax is needed to lift the terminal object.

References

1 Andreas Abel, Joakim Öhman, and Andrea Vezzosi. Decidability of conversion for type theory in type theory. *Proc. ACM Program. Lang.*, 2(POPL):23:1–23:29, 2018. doi:10.1145/3158111.

2 Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Categorical reconstruction of a reduction free normalization proof. In David H. Pitt, David E. Rydeheard, and Peter T. Johnstone, editors, *Category Theory and Computer Science, 6th International Conference, CTCS '95, Cambridge, UK, August 7-11, 1995, Proceedings*, volume 953 of *Lecture Notes in Computer Science*, pages 182–199. Springer, 1995. doi:10.1007/3-540-60164-3_27.

3 Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Reduction-free normalisation for a polymorphic system. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 98–106. IEEE Computer Society, 1996. doi:10.1109/LICS.1996.561309.

4 Thorsten Altenkirch and Ambrus Kaposi. Normalisation by Evaluation for Dependent Types. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*, volume 52 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:16, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSCD.2016.6.

5 Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, pages 18–29, New York, NY, USA, 2016. ACM. doi:10.1145/2837614.2837638.

6 Thorsten Altenkirch, Ambrus Kaposi, and Michael Shulman. Towards higher observational type theory. In Delia Kesner and Pierre-Marie Pédrot, editors, *28th International Conference on Types for Proofs and Programs (TYPES 2022)*. University of Nantes, 2022. URL: https://types22.inria.fr/files/2022/06/TYPES_2022_paper_37.pdf.

7 Danil Annenkov, Paolo Capriotti, Nicolai Kraus, and Christian Sattler. Two-level type theory and applications. *CoRR*, abs/1705.03307, 2019. arXiv:1705.03307.

8 Steve Awodey. Natural models of homotopy type theory. *Mathematical Structures in Computer Science*, 28(2):241–286, 2018. doi:10.1017/S0960129516000268.

9 Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Proofs for free — parametricity for dependent types. *Journal of Functional Programming*, 22(02):107–152, 2012. doi:10.1017/S0956796812000056.

10 Rafaël Bocquet, Ambrus Kaposi, and Christian Sattler. Relative induction principles for type theories. *CoRR*, abs/2102.11649, 2021. arXiv:2102.11649.

11 Simon Castellan, Pierre Clairambault, and Peter Dybjer. Categories with families: Unityped, simply typed, and dependently typed. *CoRR*, abs/1904.00827, 2019. arXiv:1904.00827.

12 Thierry Coquand. Canonicity and normalization for dependent type theory. *Theor. Comput. Sci.*, 777:184–191, 2019. doi:10.1016/j.tcs.2019.01.015.

13 Thierry Coquand, Simon Huber, and Christian Sattler. Canonicity and homotopy canonicity for cubical type theory, 2021. arXiv:1902.06572.

14 Thierry Coquand, Simon Huber, and Christian Sattler. Canonicity and homotopy canonicity for cubical type theory. *Log. Methods Comput. Sci.*, 18(1), 2022. doi:10.46298/lmcs-18(1:28)2022.

15 Marcelo P. Fiore. Semantic analysis of normalisation by evaluation for typed lambda calculus. In *Proceedings of the 4th international ACM SIGPLAN conference on Principles and practice of declarative programming, October 6-8, 2002, Pittsburgh, PA, USA (Affiliated with PLI 2002)*, pages 26–37. ACM, 2002. doi:10.1145/571157.571161.

16 Marcelo P. Fiore and Ola Mahmoud. Second-order algebraic theories. *CoRR*, abs/1308.5409, 2013. arXiv:1308.5409.

17 Daniel Gratzer. Normalization for multimodal type theory. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3531130.3532398.

18 Daniel Gratzer and Jonathan Sterling. Syntactic categories for dependent type theory: sketching and adequacy. *CoRR*, abs/2012.10783, 2020. arXiv:2012.10783.

19 Robert Harper. An equational logical framework for type theories. *CoRR*, abs/2106.01484, 2021. arXiv:2106.01484.

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

21 Martin Hofmann. Syntax and semantics of dependent types. In *Semantics and Logics of Computation*, pages 79–130. Cambridge University Press, 1997.

22 Martin Hofmann. Semantical analysis of higher-order abstract syntax. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 204–213. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782616.

23 Jason Z. S. Hu, Brigitte Pientka, and Ulrich Schöpp. A category theoretic view of contextual types: from simple types to dependent types. *CoRR*, abs/2206.02831, 2022. doi:10.48550/arXiv.2206.02831.

24 Peter T Johnstone. *Sketches of an elephant: a Topos theory compendium.* Oxford logic guides. Oxford Univ. Press, New York, NY, 2002. URL: https://cds.cern.ch/record/592033.

25 Achim Jung and Jerzy Tiuryn. A new characterization of lambda definability. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*, volume 664 of *Lecture Notes in Computer Science*, pages 245–257. Springer, 1993. doi:10.1007/BFb0037110.

26 Ambrus Kaposi. *Type theory in a type theory with quotient inductive types.* PhD thesis, University of Nottingham, UK, 2017. URL: https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.713896.

27 Ambrus Kaposi, Simon Huber, and Christian Sattler. Gluing for 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 25:1–25:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSCD.2019.25.

28 Krzysztof Kapulkin and Peter LeFanu Lumsdaine. The homotopy theory of type theories. *Advances in Mathematics*, 337:1–38, 2018. doi:10.1016/j.aim.2018.08.003.

29 Frank Pfenning and Carsten Schürmann. System description: Twelf - A meta-logical framework for deductive systems. In Harald Ganzinger, editor, *Automated Deduction - CADE-16, 16th International Conference on Automated Deduction, Trento, Italy, July 7-10, 1999, Proceedings*, volume 1632 of *Lecture Notes in Computer Science*, pages 202–206. Springer, 1999. doi:10.1007/3-540-48660-7_14.

30 Loïc Pujet and Nicolas Tabareau. Impredicative Observational Equality. In *POPL 2023 - 50th ACM SIGPLAN Symposium on Principles of Programming Languages*, volume 7 of *Proceedings of the ACM on programming languages*, page 74, Boston, United States, January 2023. doi:10.1145/3571739.

31 Michael Shulman. Towards a third-generation HOTT. Talk series at the Homotopy Type Theory at CMU seminar, 2022.

32 Jonathan Sterling. *First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory.* PhD thesis, Carnegie Mellon University, 2021. Version 1.1, revised May 2022. doi:10.5281/zenodo.6990769.

33 Jonathan Sterling. Naïve logical relations in synthetic Tait computability. Unpublished manuscript, June 2022.

34 Jonathan Sterling and Carlo Angiuli. Normalization for cubical type theory. *CoRR*, abs/2101.11479, 2021. arXiv:2101.11479.

35 Jonathan Sterling and Bas Spitters. Normalization by gluing for free λ-theories. *CoRR*, abs/1809.08646, 2018. arXiv:1809.08646.

36 Taichi Uemura. A general framework for the semantics of type theory. *CoRR*, abs/1904.04097, 2019. `arXiv:1904.04097`.

37 Jonathan Weinberger, Benedikt Ahrens, Ulrik Buchholtz, and Paige North. Synthetic Tait computability for simplicial type theory. In *28th International Conference on Types for Proofs and Programs (TYPES 2022)*, 2022. URL: `https://types22.inria.fr/files/2022/06/TYPES_2022_paper_17.pdf`.

A Technical results

A.1 Local representability

We recall the definition of the notion of locally representable dependent presheaf, which encodes the notion of context extension.

▶ **Definition 21.** *Let $\mathcal{C}$ be a category, X be a presheaf over $\mathcal{C}$ and Y be a dependent presheaf over X. Then Y is said to be* ***locally representable*** *if for every element $x : X(\Gamma)$, the presheaf*

$$\begin{aligned} &Y_{|x} : (\mathcal{C}/\Gamma)^{\mathsf{op}} \to \mathbf{Set}, \\ &Y_{|x}(\Delta, \rho) \triangleq Y(\Delta, x[\rho]) \end{aligned}$$

is representable. The representing object, consisting of an extended context and a projection map, is written $(\Gamma.Y[x], \boldsymbol{p}_x)$ and the generic element is written $\boldsymbol{q}_x : Y(\Gamma.Y[x], x[\boldsymbol{p}_x])$.

Given any object $\Delta \in \mathcal{C}$, map $\rho : \Delta \to \Gamma$ and element $y : Y(\Delta, x[\rho])$, we write $\langle \rho, a \rangle$ for the unique morphism such that $\boldsymbol{p}_x \circ \langle \rho, y \rangle = \rho$ and $\boldsymbol{q}_x[\langle \rho, y \rangle] = y$. ⌟

A.2 Characterization of the telescopic contextualization

We define the contextual slices of a first-order model (which are called fibrant slices in [28]).

▶ **Definition 22** (Contextual slice). *Let $\mathcal{C}$ be a first-order model of $\mathcal{T}$. Given $\Gamma \in \mathcal{C}$, the contextual slice $(\mathcal{C} \mathbin{/\!\!/} \Gamma)$ is the contextual first-order model given by:*

- *Objects of $(\mathcal{C} \mathbin{/\!\!/} \Gamma)$ are telescopes (iterated context extensions) over Γ.*
- *Morphisms from Δ_1 to Δ_2 are morphisms from $\Gamma.\Delta_1$ to $\Gamma.\Delta_2$ in $(\mathcal{C}/\Gamma)$.*
- *The rest of the structure is inherited from $\mathcal{C}$ along the projection*

$$(\mathcal{C} \mathbin{/\!\!/} \Gamma) \ni \Delta \mapsto \Gamma.\Delta \in \mathcal{C}.$$ ⌟

The contextual slice is functorial in both $\mathcal{C}$ and (contravariantly) Γ:

- For any $f : \Delta \to \Gamma$, there is a pullback morphism $f^* : (\mathcal{C} \mathbin{/\!\!/} \Gamma) \to (\mathcal{C} \mathbin{/\!\!/} \Delta)$. Its actions on objects, substitutions, types and terms are all given by substitution along f.
- For any $F : \mathcal{C} \to \mathcal{D}$, there is a morphism $(F \mathbin{/\!\!/} \Gamma) : (\mathcal{C} \mathbin{/\!\!/} \Gamma) \to (\mathcal{D} \mathbin{/\!\!/} F(\Gamma))$. Its actions on objects, substitutions, types and terms are given by the actions of F on telescopes, substitutions, types and terms.
- The following diagrams commute (for any $F : \mathcal{C} \to \mathcal{D}$ and $f : \Delta \to \Gamma$):

$$\begin{tikzcd}
(\mathcal{C} \mathbin{/\!\!/} \Gamma) \arrow[r, "(F /\!\!/ \Gamma)"] \arrow[d, "f^*"] & (\mathcal{D} \mathbin{/\!\!/} F(\Gamma)) \arrow[d, "f^*"] \\
(\mathcal{C} \mathbin{/\!\!/} \Delta) \arrow[r, "(F /\!\!/ \Delta)"] & (\mathcal{D} \mathbin{/\!\!/} F(\Delta))\ .
\end{tikzcd}$$

- For any $f : \Theta \to \Gamma$ and object Δ of $(\mathcal{C} \mathbin{/\!\!/} \Gamma)$, we have $(f^* \mathbin{/\!\!/} \Delta) = \langle \boldsymbol{p}^*_\Delta(f), \boldsymbol{q}_\Delta \rangle^*$ as a morphism from $(\mathcal{C} \mathbin{/\!\!/} \Gamma.\Delta)$ to $(\mathcal{C} \mathbin{/\!\!/} \Theta.f^*(\Delta))$. Here $\langle f \circ \boldsymbol{p}_{f^*(\Delta)}, \boldsymbol{q}_{f^*(\Delta)} \rangle$ is a morphism from $\Theta.f^*(\Delta)$ to $\Gamma.\Delta$.

▶ **Lemma 23.** *Let $\mathcal{C}$ be a first-order model of $\mathcal{T}$. Then the telescopic contextualization $\mathbf{Tele}_{\mathbb{C}}$ corresponds the contextual slice functor*

$$(\mathcal{C} \mathbin{/\!\!/} -) : \mathcal{C} \to \mathbf{Mod}^{\mathsf{op}}_{\mathcal{T}}.$$

⌟

Proof. Immediate from unfolding the definitions. ◀

▶ **Corollary 24.** *The externalization $1^*_{\mathcal{C}}(\mathbf{Tele}_{\mathbb{C}})$ is the contextual core of $\mathcal{C}$.*

Proof. By Lemma 23, $1^*_{\mathcal{C}}(\mathbf{Tele}_{\mathbb{C}})$ is the contextual slice $(\mathcal{C} \mathbin{/\!\!/} 1_{\mathcal{C}})$, the contextual core of $\mathcal{C}$. ◀

▶ **Lemma 25.** *Let $\mathcal{C}$ be a contextual first-order model and $A : \mathcal{C}.\mathsf{Ty}(1_{\mathcal{C}})$ be a closed type. Then the contextual slice $(\mathcal{C} \mathbin{/\!\!/} A)$ satisfies the following universal property: for every model $\mathcal{E}$, morphism $F : \mathcal{C} \to \mathcal{E}$ and element $a : \mathcal{E}.\mathsf{Tm}(F(A))$, there is a unique morphism $\widetilde{F} : (\mathcal{C} \mathbin{/\!\!/} 1_{\mathcal{C}}.A) \to \mathcal{E}$ such that $\widetilde{F} \circ \boldsymbol{p}^*_A = F$ and $\widetilde{F}(\boldsymbol{q}_A) = a$.*

In other words, $(\mathcal{C} \mathbin{/\!\!/} 1_{\mathcal{C}}.A)$ is the free extension of $\mathcal{C}$ by a generic element $\boldsymbol{q}_A$ of type A.

See the Full Version of the paper for the proof.

▶ **Lemma 26.** *Given any $\Gamma \in \mathcal{C}$ and $A : \mathcal{C}.\mathsf{Ty}(\Gamma)$, then $(\mathcal{C} \mathbin{/\!\!/} \Gamma.A)$ satisfies the following universal property: for every model $\mathcal{E}$, morphism $F : (\mathcal{C} \mathbin{/\!\!/} \Gamma) \to \mathcal{E}$ and element $a : \mathcal{E}.\mathsf{Tm}(F(A))$, there is a unique morphism $\widetilde{F} : (\mathcal{C} \mathbin{/\!\!/} \Gamma.A) \to \mathcal{E}$ such that $\widetilde{F} \circ \boldsymbol{p}^*_A = F$ and $\widetilde{F}(\boldsymbol{q}_A) = a$.*

In other words, $(\mathcal{C} \mathbin{/\!\!/} \Gamma.A)$ is the free extension of $(\mathcal{C} \mathbin{/\!\!/} \Gamma)$ by a generic element $\boldsymbol{q}_A$ of type A.

Proof. We have $(\mathcal{C} \mathbin{/\!\!/} \Gamma.A) \cong ((\mathcal{C} \mathbin{/\!\!/} \Gamma) \mathbin{/\!\!/} 1_{(\mathcal{C} /\!\!/ \Gamma)}.A)$. Then the result follows from Lemma 25. ◀

A.3 Properties of the category of sections

We now prove the properties of the category of sections of a displayed first-order model. Fix a global internal first-order model $\mathcal{M} : \mathcal{C} \to \mathbf{Mod}^{\mathsf{op}}_{\mathcal{T}}$ and a displayed first-order model $\mathcal{M}^\bullet : \mathcal{C} \to \mathbf{DispMod}^{\mathsf{op}}_{\mathcal{T}}$ over $\mathcal{M}$.

We prove conditions that relate the existence of some finite limits in $\mathbf{Sect}^{\mathsf{op}}_{\mathcal{T}}[\mathcal{M}^\bullet]$ to the universal properties of the first-order models $\mathcal{M}(\Gamma)$ for $\Gamma \in \mathcal{C}$.

We have defined the category $\mathbf{Sect}^{\mathsf{op}}_{\mathcal{T}}[\mathcal{M}^\bullet]$ of sections of $\mathcal{M}^\bullet$ as the following pullback

$$\begin{tikzcd}
\mathbf{Sect}^{\mathsf{op}}_{\mathcal{T}}[\mathcal{M}^\bullet] \arrow[r, "{[\![-]\!]_0}"] \arrow[d, "\pi_0"] \arrow[dr, phantom, "\lrcorner", very near start] & \mathbf{Sect}^{\mathsf{op}}_{\mathcal{T}} \arrow[d] \\
\mathcal{C} \arrow[r, "\mathcal{M}^\bullet"] & \mathbf{DispModel}^{\mathsf{op}}_{\mathcal{T}} \ .
\end{tikzcd}$$

Unfolding the definition, an object of $\mathbf{Sect}^{\mathsf{op}}_{\mathcal{T}}[\mathcal{M}^\bullet]$ is a pair $(\Gamma, [\![-]\!]_\Gamma)$ where $\Gamma \in \mathcal{C}$ and $[\![-]\!]_\Gamma$ is a section of $\mathcal{M}^\bullet(\Gamma)$ over $\mathcal{M}(\Gamma)$. A morphism of from $(\Gamma, [\![-]\!]_\Gamma)$ to $(\Delta, [\![-]\!]_\Delta)$ is a morphism $f : \mathcal{C}(\Gamma, \Delta)$ such that the outer square commutes in the following diagram:

$$\begin{tikzcd}
\mathcal{M}^\bullet(\Delta) \arrow[r, "\mathcal{M}^\bullet(f)"] \arrow[d] & \mathcal{M}^\bullet(\Gamma) \arrow[d] \\
\mathcal{M}(\Delta) \arrow[r, "\mathcal{M}(f)"] \arrow[u, bend left, "{[\![-]\!]_\Gamma}"] & \mathcal{M}(\Gamma) \ . \arrow[u, bend right, "{[\![-]\!]_\Delta}"']
\end{tikzcd}$$

▶ **Lemma 27.** *If $\mathcal{C}$ has a terminal object $1_{\mathcal{C}}$ and $\mathcal{M}(1_{\mathcal{C}})$ is the initial model of $\mathcal{T}$, then $\mathbf{Sect}^{\mathsf{op}}_{\mathcal{M}^\bullet}$ has a terminal object that is strictly preserved by π_0.*

Proof. Since $\mathcal{M}(1_{\mathcal{C}})$ is initial, we obtain a section $[\![-]\!]_{1_{\mathcal{C}}}$ of $\mathcal{M}^\bullet(1_{\mathcal{C}})$. We now prove that $(1_{\mathcal{C}}, [\![-]\!]_{1_{\mathcal{C}}})$ is terminal in $\mathbf{Sect}^{\mathsf{op}}_{\mathcal{M}^\bullet}$. Let $(\Gamma, [\![-]\!]_\Gamma)$ be any object of $\mathbf{Sect}^{\mathsf{op}}_{\mathcal{M}^\bullet}$. A morphism from $(\Gamma, [\![-]\!]_\Gamma)$ to $(1_{\mathcal{C}}, [\![-]\!]_{1_{\mathcal{C}}})$ is a morphism $f : \mathcal{C}(\Gamma, 1_{\mathcal{C}})$ such that the following square commutes:

$$\begin{tikzcd}
\mathcal{M}^\bullet(1_{\mathcal{C}}) \arrow[r, "\mathcal{M}^\bullet(f)"] & \mathcal{M}^\bullet(\Gamma) \\
\mathcal{M}(1_{\mathcal{C}}) \arrow[u, "{[\![-]\!]_{1_{\mathcal{C}}}}"] \arrow[r, "\mathcal{M}(f)"] & \mathcal{M}(\Gamma) \arrow[u, "{[\![-]\!]_\Gamma}"'] .
\end{tikzcd}$$

Since $1_{\mathcal{C}}$ is terminal, there is only one morphism $f : \mathcal{C}(\Gamma, 1_{\mathcal{C}})$, and the corresponding square commutes by initiality of $\mathcal{M}(1_{\mathcal{C}})$. ◀

▶ **Definition 28.** *Let X be a presheaf over $\mathcal{C}$ and Y be a locally representable dependent presheaf over X. Assume given the data of global elements*

$$\begin{aligned}
&f_X : X \to \mathcal{M}.\mathsf{Ty}(1_{\mathcal{M}}),\\
&f_Y : (x : X) \to Y(x) \to \mathcal{M}.\mathsf{Tm}(1_{\mathcal{M}}, f_X(x))
\end{aligned}$$

of $\mathbf{Psh}(\mathcal{C})$.

We say that f_Y is compatible with $\mathcal{M}$ if for every element $x : X(\Gamma)$, the external first-order model $\mathcal{M}(\Gamma.Y(x))$ satisfies the following universal property: for every first-order model $\mathcal{E}$, morphism $E : \mathcal{M}(\Gamma) \to \mathcal{E}$ and element $z : \mathcal{E}.\mathsf{Tm}(1_{\mathcal{E}}, E(f_X(x)))$, there is a unique morphism $\widetilde{E} : \mathcal{M}(\Gamma.Y(x)) \to \mathcal{E}$ such that $E = \widetilde{E} \circ \mathcal{M}(\boldsymbol{p}_x)$ and $z = \widetilde{E}(f_Y(x, \boldsymbol{q}_x))$.

In other words, $\mathcal{M}(\Gamma.Y(x))$ should be the free extension of $\mathcal{M}(\Gamma)$ by an element $f_Y(x, \boldsymbol{q}_x)$ of type $f_X(x)$, the extension being witnessed by the morphism $\mathcal{M}(\boldsymbol{p}_x) : \mathcal{M}(\Gamma) \to \mathcal{M}(\Gamma.Y(x))$. ⌟

▶ **Lemma 29.** *Let $F : \mathcal{C} \to \mathcal{D}$ be a CwF morphism.*

Then the condition of Definition 28 is satisfied, with $\mathcal{M} = F^*(\mathsf{Tele}_{\mathbb{D}})$, X and Y being respectively the types and terms of $\mathcal{C}$, and f_X and f_Y being the actions of F on types and terms.

Proof. By Definition 22, $\mathcal{M}$ correspond to the functor

$$\mathcal{C} \ni \Gamma \mapsto (\mathcal{D} /\!\!/ F(\Gamma)) \in \mathbf{Mod}^{\mathsf{op}}_{\mathcal{T}}.$$

Thus, we need to check that given any $\Gamma \in \mathcal{C}$ and $A : \mathcal{C}.\mathsf{Ty}(\Gamma)$, the model $(\mathcal{D} /\!\!/ F(\Gamma.A))$ is the free extension of $(\mathcal{D} /\!\!/ F(\Gamma))$ by a generic element of type $F(A)$. Since F preserves extensions, $F(\Gamma.A) \cong F(\Gamma).F(A)$, and thus the result follows by Lemma 26. ◀

▶ **Lemma 30.** *Let X be a presheaf over $\mathcal{C}$ and Y be a locally representable presheaf family over X equipped with operations f_X and f_Y satisfying the condition of Definition 28. Finally, assume that for every $(\Gamma, [\![-]\!]_\Gamma) \in \mathbf{Sect}^{\mathsf{op}}_{\mathcal{M}^\bullet}$, $\Delta \in \mathcal{C}$, $\gamma : \mathcal{C}(\Delta, \Gamma)$, $x : X(\Gamma)$ and $y : Y(\Delta, x[\gamma])$ we have*

$$f^\bullet_Y(x, y) : \mathcal{M}^\bullet(\Delta).\mathsf{Tm}^\bullet(1^\bullet, \mathcal{M}^\bullet(\gamma)([\![f_X(x)]\!]_\Gamma), f_Y(x[\gamma], y)),$$

naturally in $(\Gamma, [\![-]\!]_\Gamma)$ and Δ.

Consider the presheaf $X_0 \triangleq \pi_0^*(X)$ *over* $\mathbf{Sect}^{\mathsf{op}}_{\mathcal{M}^\bullet}$ *and the dependent presheaf* Y_0 *over* X_0 *specified on objects by:*

$$Y_0((\Gamma, [\![-]\!]_\Gamma), x) \triangleq \{y : Y(\Gamma, x) \mid [\![f_Y(x,y)]\!]_\Gamma = f^\bullet_Y(x,y)\}.$$

Then the presheaf family Y_0 *is locally representable and the action induced by the first projections* $Y_0((\Gamma, [\![-]\!]_\Gamma), x) \to Y(\Gamma, x)$ *strictly preserve context extensions.*

Proof. Let $(\Gamma, [\![-]\!]_\Gamma)$ be an object of $\mathbf{Sect}^{\mathsf{op}}_{\mathcal{M}^\bullet}$ and $x : X(\Gamma)$ be an element of X_0 at this object. We have to prove that the presheaf $Y_{0|x}$ over $(\mathbf{Sect}^{\mathsf{op}}_{\mathcal{M}^\bullet}/(\Gamma, [\![-]\!]_\Gamma))$ is representable.

Consider the diagram:

$$\begin{tikzcd}
\mathcal{M}^\bullet(\Gamma) \arrow[r, "\mathcal{M}^\bullet(\boldsymbol{p}_x)"] \arrow[d] & \mathcal{M}^\bullet(\Gamma.Y(x)) \arrow[d] \\
\mathcal{M}(\Gamma) \arrow[r, "\mathcal{M}(\boldsymbol{p}_x)"] \arrow[u, bend left, "{[\![-]\!]_\Gamma}"] & \mathcal{M}(\Gamma.Y(x)) \arrow[u, bend right, dashed, "{[\![-]\!]_{\Gamma.Y(x)}}"']
\end{tikzcd}$$

We construct a section $[\![-]\!]_{\Gamma.Y(x)}$ of $\mathcal{M}^\bullet(\Gamma.Y(x))$ over $\mathcal{M}(\Gamma.Y(x))$. Using the universal property of $\mathcal{M}(\Gamma.Y(x))$, we define $[\![-]\!]_{\Gamma.Y(x)}$ as the unique extension of $\mathcal{M}^\bullet(\boldsymbol{p}_x) \circ [\![-]\!]_\Gamma$ that sends $f_Y(x, \boldsymbol{q}_x)$ to $f^\bullet_Y(x, \boldsymbol{q}_x)$.

We can check that $\boldsymbol{p}_x$ lifts to a morphism $\boldsymbol{p}_x : (\Gamma.Y(x), [\![-]\!]_{\Gamma.Y(x)}) \to (\Gamma, [\![-]\!]_\Gamma)$ in $\mathbf{Sect}^{\mathsf{op}}_{\mathcal{M}^\bullet}$.

We now show that $((\Gamma.Y(x), [\![-]\!]_{\Gamma.Y(x)}), \boldsymbol{p}_x)$ represents the functor $Y_{0|x}$. Let $(\Delta, [\![-]\!]_\Delta)$ be another object of $\mathbf{Sect}^{\mathsf{op}}_{\mathcal{M}^\bullet}$, with a morphism $\rho : (\Delta, [\![-]\!]_\Delta) \to (\Gamma, [\![-]\!]_\Gamma)$ and an element $y : Y_{0|x}((\Delta, [\![-]\!]_\Delta), \rho)$. Unfolding the definitions, we have $y : Y(\Delta, x[\rho])$ with $[\![f_Y(x[\rho], y)]\!]_\Delta = f^\bullet_Y(x[\rho], y)$.

The local representability of Y implies that there is a unique morphism $\widetilde{\rho} : \Delta \to \Gamma.Y(x)$ in $\mathcal{C}$ such that $\boldsymbol{p}_x \circ \widetilde{\rho} = \rho$ and $\boldsymbol{q}_x[\widetilde{\rho}] = y$. We have to show that this morphism lifts to $\mathbf{Sect}^{\mathsf{op}}_{\mathcal{M}^\bullet}$, i.e. that the following square commutes:

$$\begin{tikzcd}
\mathcal{M}^\bullet(\Gamma.Y(x)) \arrow[r, "\mathcal{M}^\bullet(\widetilde{\rho})"] \arrow[d] & \mathcal{M}^\bullet(\Delta) \arrow[d] \\
\mathcal{M}(\Gamma.Y(x)) \arrow[r, "\mathcal{M}(\widetilde{\rho})"] \arrow[u, bend left, "{[\![-]\!]_{\Gamma.Y(x)}}"] & \mathcal{M}(\Delta) \arrow[u, bend right, "{[\![-]\!]_\Delta}"']
\end{tikzcd}$$

By the universal property of $\mathcal{M}(\Gamma.Y(x))$, it suffices to show that $f_Y(x, \boldsymbol{q}_x)$ is mapped to the same element by the compositions $\mathcal{M}^\bullet(\widetilde{\rho}) \circ [\![-]\!]_{\Gamma.Y(x)}$ and $[\![-]\!]_\Delta \circ \mathcal{M}(\widetilde{\rho})$. We compute $\mathcal{M}^\bullet(\widetilde{\rho})([\![f_Y(x, \boldsymbol{q}_x)]\!]_{\Gamma.Y(x)}) = \mathcal{M}^\bullet(\widetilde{\rho})(f^\bullet_Y(x, \boldsymbol{q}_x)) = f^\bullet_Y(x[\rho], y)$ and $[\![\mathcal{M}(\widetilde{\rho})(f_Y(x, \boldsymbol{q}_x))]\!]_\Delta = [\![f_Y(x[\rho], y)]\!]_\Delta = f^\bullet_Y(x[\rho], y)$.

This completes the proof that $((\Gamma.Y(x), [\![-]\!]_{\Gamma.Y(x)}), \boldsymbol{p}_x)$ represents the functor $Y_{0|x}$.

We have proven that $Y_{0|x}$ is representable for every x, i.e. that Y_0 is locally representable. The first projections $Y_0((\Gamma, [\![-]\!]_\Gamma), x) \to Y(\Gamma, x)$ strictly preserve the chosen representing objects. ◀

A.4 Proofs of relative induction principles

Proof of Theorem 18. We consider the category $\mathbf{Sect}^{\mathsf{op}}_{\mathcal{T}}[\mathbf{Scone}_{\mathbb{S}^\bullet}]$ of sections of $\mathbf{Scone}_{\mathbb{S}^\bullet}$.

By Lemma 27, the category $\mathbf{Sect}^{\mathsf{op}}_{\mathcal{T}}[\mathbf{Scone}_{\mathbb{S}^\bullet}]$ has a terminal object. We equip it with the structure of a higher-order renaming algebra $(\mathsf{Var}_0, \mathsf{var}_0)$ over $(F \circ \pi_0) : \mathbf{Sect}^{\mathsf{op}}_{\mathcal{T}}[\mathbf{Scone}_{\mathbb{S}^\bullet}] \to \mathcal{S}$ as follows:

$$\mathsf{Var}_0((\Gamma, [\![-]\!]_\Gamma), A) \triangleq \{a : \mathsf{Var}(\Gamma, A) \mid [\![\mathsf{var}(\Gamma, a)]\!]_\Gamma = \mathsf{var}^\bullet(\Gamma, [\![A]\!]_\Gamma, a)\},$$

$$\mathsf{var}_0((\Gamma, [\![-]\!]_\Gamma), A, a) \triangleq \mathsf{var}(\Gamma, a).$$

By Lemma 29, the action of $F : \mathcal{R} \to \mathcal{S}$ on variables is compatible with $\mathcal{S}_F$. By Lemma 30, the presheaf family Var_0 is locally representable and the first projections $\mathsf{Var}_0((\Gamma, [\![-]\!]_\Gamma), A) \to \mathsf{Var}(\Gamma, A)$ strictly preserve context extensions. By initiality of $\mathcal{R}$ among first-order renaming algebras, we obtain a section H of π_0 in the category of first-order renaming algebras.

We thus have a section $[\![-]\!] \triangleq H^*([\![-]\!]_0)$ of $\mathbf{Scone}_{\mathbb{S}^\bullet}$ in $\mathbf{Psh}(\mathcal{R})$. The action of H on variables proves that it satisfies the equality $[\![\mathsf{var}_A(x)]\!] = \mathsf{var}^\bullet([\![A]\!], x)$. ◀

The Logical Essence of Compiling with Continuations

José Espírito Santo

Centre of Mathematics, University of Minho, Portugal

Filipa Mendes

Centre of Mathematics, University of Minho, Portugal

Abstract

The essence of compiling with continuations is that conversion to continuation-passing style (CPS) is equivalent to a source language transformation converting to administrative normal form (ANF). Taking as source language Moggi's computational lambda-calculus (λC), we define an alternative to the CPS-translation with target in the sequent calculus LJQ, named *value-filling style* (VFS) translation, and making use of the ability of the sequent calculus to represent contexts formally. The VFS-translation requires no type translation: indeed, double negations are introduced only when encoding the VFS target language in the CPS target language. This optional encoding, when composed with the VFS-translation reconstructs the original CPS-translation. Going back to direct style, the "essence" of the VFS-translation is that it reveals a new sublanguage of ANF, the *value-enclosed style* (VES), next to another one, the *continuation-enclosing style* (CES): such an alternative is due to a dilemma in the syntax of λC, concerning how to expand the application constructor. In the typed scenario, VES and CES correspond to an alternative between two proof systems for call-by-value, LJQ and natural deduction with generalized applications, confirming proof theory as a foundation for intermediate representations.

2012 ACM Subject Classification Theory of computation → Proof theory; Theory of computation → Operational semantics; Theory of computation → Type structures

Keywords and phrases Continuation-passing style, Sequent calculus, Generalized applications, Administrative normal form

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.19

Related Version *Full Version*: `https://arxiv.org/abs/2304.14752`

Funding The first author was partially financed by Portuguese Funds through FCT (Fundação para a Ciência e a Tecnologia) within the Projects UIDB/00013/2020 and UIDP/00013/2020.

1 Introduction

The conversion of a program in a source call-by-value language to continuation-passing style (CPS) by an optimizing translation that reduces on the fly the so-called administrative redexes produces programs which can be translated back to direct style, so that the final result, obtained by composing the two stages of translation, is a new program in the source language which can be obtained from the original one by reduction to administrative normal form (ANF) – a program transformation in the source language [10, 24]. This fact has been dubbed the "essence" of compiling with continuations and has had a big impact and generated an on-going debate in the theory and practice of compiler design [11, 16, 18].

Our starting point is the refinement of that "essence", obtained in [25], in the form of a reflection of the CPS target in the computational λ-calculus [20], the latter playing the role of source language and here denoted λC – see Fig. 1. Then we ask: What is the proof-theoretical meaning of this reflection? What is the logical reading of this reflection in the typed setting? Of course, the CPS-translation has a well-known logical reading as a

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 19; pp. 19:1–19:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

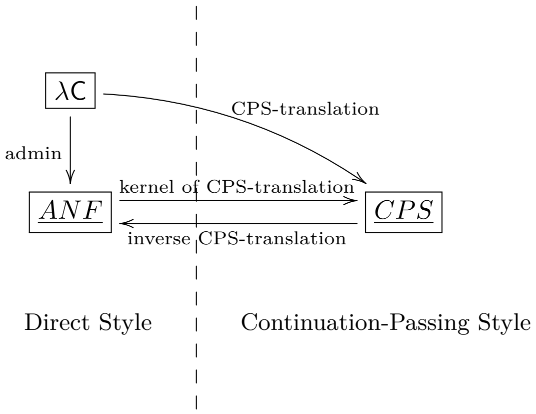


Figure 1 The essence of compiling with continuations.

negative translation, based on the introduction of double negations, capable of translating a classical source calculus with control operators [19, 12, 26]. But it is not clear how this reading is articulated with the reflection in Fig.1, which provides a *decomposition* of the CPS-translation as the reduction to ANF followed by a "kernel" translation that relates the "kernel" ANF with CPS.

It is also well-known that the CPS-translation can be decomposed in several ways: indeed in the reference [25] alone we may find two of them, one through the monadic meta-language [21], the other through the linear λ-calculus [17]. Here we will propose another intermediate language, the sequent calculus LJQ [3, 4]. The calculus LJQ has a long history and several applications in proof theory [3] and can be turned into a typed call-by-value λ-calculus in equational correspondence with $\lambda\mathsf{C}$ [4]. Here we want to show it has a privileged role as a tool to analyze the CPS-translation.

Languages of proof terms for the sequent calculus handle contexts (*i*.e. λ-terms with a hole) formally [13, 1, 8, 5]. This seems most convenient, since a continuation may be seen as a certain kind of context, and suggests that we can write an alternative translation into the sequent calculus, as if we were CPS-translating, but without the need to pass around a reification of the current continuation as a λ-abstraction, nor the concomitant need to translate types by the insertion of double negations, to make room for a type $A^\sim$ of values, a type $\neg A^\sim$ of continuations and a type $\neg\neg A^\sim$ of programs, out of a source type A.

We develop this in detail, which requires: to rework entirely the term calculus for LJQ and obtain a system, named λQ, more manageable for our purposes; and to identify the kernel and the sub-kernel of λQ, the latter being the target system, named VFS after *value-filling style*, of the new translation. In the end, we are rewarded with an isomorphism between VFS and the target of the CPS-translation, which, when composed with the alternative translation, reconstructs the CPS-translation. The isomorphism is a negative translation, reduced to the role of optional and late stage of translation.

Going back to direct style, the "essence" of the VFS-translation is that it reveals a new sublanguage of ANF, the *value-enclosed style* (VES), next to another sublanguage of ANF, the *continuation-enclosing style* (CES): such alternative between VES and CES is due to a dilemma in the syntax of $\lambda\mathsf{C}$, concerning how to *expand* the application constructor. Hence, these two sub-kernels of $\lambda\mathsf{C}$ are under a layer of expansion – and the same was already true for the passage from the kernel to the sub-kernel of λQ.

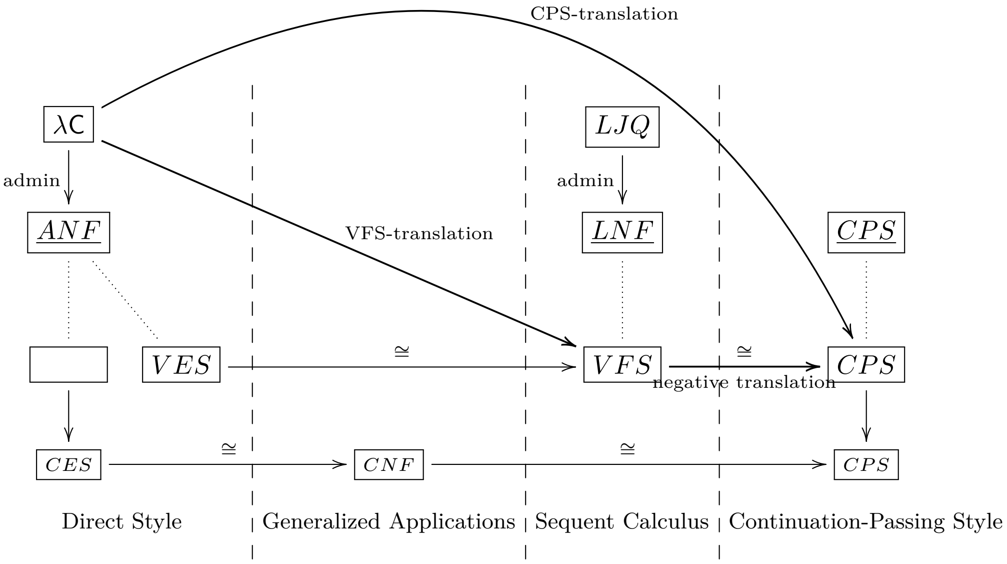


Figure 2 The logical essence of compiling with continuations.

While VES corresponds to the sub-kernel VFS of λQ, CES corresponds to a fragment of $\lambda\mathsf{J}_v$ [6], a call-by-value λ-calculus with generalized applications; the fragment is that of *commutative normal forms* (CNF), that is, normal forms w. r. t. the commutative conversions, naturally arising when application is generalized, which reduce both the head term and the argument in an application to the form of values. So the alternative between VES and CES is also a reflection, in the source language, of the alternative between two proof systems for call-by-value: the sequent calculus LJQ and the natural deduction system behind $\lambda\mathsf{J}_v$.

A summary is contained in Fig. 2: it shows a proof-theoretical background hidden in Fig. 1, which this paper wants to reveal. In the process, we want to confirm proof theory as a foundation for intermediate representations useful in the compilation of functional languages.

Plan of the paper. Section 2 recalls $\lambda\mathsf{C}$ and the CPS-translation. Section 3 contains our reworking of LJQ. Section 4 introduces the alternative translation into LJQ and the decomposition of the CPS-translation. Section 5 goes back to direct style and studies the sub-kernels of $\lambda\mathsf{C}$. Section 6 summarizes our contribution and discusses related and future work. All proofs can be found in the long version of this paper [7].

2 Background

Preliminaries. Simple types (=formulas) are given by $A, B, C ::= a | A \supset B$. In typing systems, a context Γ will always be a *consistent* set of declarations $x : A$; consistency here means that no variable can be declared with two different types in Γ.

We recall the concepts of equational correspondence, pre-Galois connection and reflection [4, 24, 25] characterizing different forms of relationship between two calculi.

▶ **Definition 1.** *Let $(\Lambda_1, \to_1)$ and $(\Lambda_2, \to_2)$ be two calculi and, for each $i = 1, 2$, let $\twoheadrightarrow_i$ (resp. $\leftrightarrow_i$) be the reflexive-transitive (resp. reflexive-transitive-symmetric) closure of $\to_i$. Consider the mappings $f : \Lambda_1 \to \Lambda_2$ and $g : \Lambda_2 \to \Lambda_1$.*

- *f and g form an **equational correspondence** between Λ_1 and Λ_2 if the following conditions hold: (1) If $M \to_1 N$ then $f(M) \leftrightarrow_2 f(N)$; (2) If $M \to_2 N$ then $g(M) \leftrightarrow_1 g(N)$; (3) $M \leftrightarrow_1 g(f(M))$; (4) $f(g(M)) \leftrightarrow_2 M$.*
- *f and g form a **pre-Galois connection** from Λ_1 to Λ_2 if the following conditions hold: (1) If $M \to_1 N$ then $f(M) \twoheadrightarrow_2 f(N)$; (2) If $M \to_2 N$ then $g(M) \twoheadrightarrow_1 g(N)$; (3) $M \twoheadrightarrow_1 g(f(M))$.*
- *f and g form a **reflection** in Λ_1 of Λ_2 if the following conditions hold: (1) If $M \to_1 N$ then $f(M) \twoheadrightarrow_2 f(N)$; (2) If $M \to_2 N$ then $g(M) \twoheadrightarrow_1 g(N)$; (3) $M \twoheadrightarrow_1 g(f(M))$; (4) $f(g(M)) = M$.*

Note that if f and g form a pre-Galois connection from Λ_1 to Λ_2 and $\to_2$ is confluent, then $\to_1$ is also confluent. Besides, it is also important to observe that if f and g form a reflection from Λ_1 to Λ_2, then g and f form a pre-Galois connection from Λ_2 to Λ_1.

Computational lambda-calculus. The computational λ-calculus [20] is defined in Fig. 3. In addition to ordinary λ-terms, one also has *let-expressions* $\mathsf{let}\, x := M \,\mathsf{in}\, N$: these are explicit substitutions which trigger only after the actual parameter M is reduced to a value (that is, a variable or λ-abstraction). So, in addition to the rule let_v that triggers substitution, there are reduction rules – let_1, let_2 and $assoc$ – dedicated to that preliminary reduction of actual parameters in let-expressions.

For the reduction of β-redexes, we adopt the rule B from [4], which triggers even if the argument N is not a value, and just generates a let-expression. Most presentations of $\lambda\mathsf{C}$ [20, 25] have rule β_v instead, which reads $(\lambda x.M)V \to [V/x]M$. The two versions of the system are equivalent. In our presentation, the effect of β_v is achieved with B followed by let_v. Conversely, when N is not a value, we can perform the reduction

$$(\lambda x.M)N \to \mathsf{let}\, y := N \,\mathsf{in}\, (\lambda x.M)y \to \mathsf{let}\, y := N \,\mathsf{in}\, [y/x]M =_\alpha \mathsf{let}\, x := N \,\mathsf{in}\, M \;\; .$$

The first step is by let_2, the second by β_v. The last term is the contractum of B.

In this paper, we leave the η-rule for λ-abstraction out of the definition of $\lambda\mathsf{C}$, and similarly for other systems – since it plays no rule in what we want to say. But we include the η-rule for let-expressions, and other incarnations of it in other systems.

In [4, 25] the $\lambda\mathsf{C}$-calculus is studied in its untyped version. Here we will also consider its simply-typed version, which handles sequents $\Gamma \vdash_{\mathsf{C}} M : A$, where Γ is a set of declarations $x : A$. The typing rules are obvious, Fig. 3 only contains the rule for typing let-expressions.

The *kernel* of the computational λ-calculus [25] is defined in Fig. 4. It is named here $\underline{ANF}$, after "administrative normal form", because its terms are the normal forms w. r. t. the administrative rules of $\lambda\mathsf{C}$: let_1, let_2 and $assoc$ [25].

In the kernel, only a specific form of applications and two forms of let-expressions are primitive. The general form of a let-expression, written $\mathsf{LET}\, y := M \,\mathsf{in}\, P$, is a derived form defined by recursion on M as follows:

$$\begin{array}{rcl} \mathsf{LET}\, y := V \,\mathsf{in}\, P & = & \mathsf{let}\, y := V \,\mathsf{in}\, P \\ \mathsf{LET}\, y := VW \,\mathsf{in}\, P & = & \mathsf{let}\, y := VW \,\mathsf{in}\, P \\ \mathsf{LET}\, y := (\mathsf{let}\, x := V \,\mathsf{in}\, M) \,\mathsf{in}\, P & = & \mathsf{let}\, x := V \,\mathsf{in}\, \mathsf{LET}\, y := M \,\mathsf{in}\, P \\ \mathsf{LET}\, y := (\mathsf{let}\, x := VW \,\mathsf{in}\, M) \,\mathsf{in}\, P & = & \mathsf{let}\, x := VW \,\mathsf{in}\, \mathsf{LET}\, y := M \,\mathsf{in}\, P \end{array}$$

Obviously, given M and P in the kernel, $\mathsf{let}\, y := M \,\mathsf{in}\, P \twoheadrightarrow_{assoc} \mathsf{LET}\, y := M \,\mathsf{in}\, P$ in $\lambda\mathsf{C}$. Hence, a B_v-step in the kernel can be simulated in $\lambda\mathsf{C}$ as a B-step followed by a series of $assoc$-steps. On the other hand B'_v is a restriction of rule B to the sub-syntax, and the same is true of the remaining rules of the kernel.

$$
\begin{array}{rrcll}
\text{(terms)} & M,N,P,Q & ::= & V \mid MN \mid \mathsf{let}\, x := M \,\mathsf{in}\, N & \\
\text{(values)} & V,W & ::= & x \mid \lambda x.M &
\end{array}
$$

$$
\begin{array}{rrcll}
(B) & (\lambda x.M)N & \to & \mathsf{let}\, x := N \,\mathsf{in}\, M & \\
(let_v) & \mathsf{let}\, x := V \,\mathsf{in}\, M & \to & [V/x]M & \\
(\eta_{let}) & \mathsf{let}\, x := M \,\mathsf{in}\, x & \to & M & \\
(assoc) & \mathsf{let}\, y := (\mathsf{let}\, x := M \,\mathsf{in}\, N) \,\mathsf{in}\, P & \to & \mathsf{let}\, x := M \,\mathsf{in}\, \mathsf{let}\, y := N \,\mathsf{in}\, P & \\
(let_1) & MN & \to & \mathsf{let}\, x := M \,\mathsf{in}\, xN & (a) \\
(let_2) & VN & \to & \mathsf{let}\, x := N \,\mathsf{in}\, Vx & (b)
\end{array}
$$

$$
\frac{\Gamma \vdash_{\mathsf{C}} M : A \quad \Gamma, x : A \vdash_{\mathsf{C}} N : B}{\Gamma \vdash_{\mathsf{C}} \mathsf{let}\, x := M \,\mathsf{in}\, N : B}
$$

Figure 3 The computational λ-calculus, here also named λC-calculus. Provisos: (a) M is not a value. (b) N is not a value. Typing rules for x, $\lambda x.M$ and MN as usual.

$$
\begin{array}{rrcl}
\text{(terms)} & M,N,P,Q & ::= & V \mid VW \mid \mathsf{let}\, x := V \,\mathsf{in}\, M \mid \mathsf{let}\, x := VW \,\mathsf{in}\, M \\
\text{(values)} & V,W & ::= & x \mid \lambda x.M
\end{array}
$$

$$
\begin{array}{rrcl}
(B_v) & \mathsf{let}\, y := (\lambda x.M)V \,\mathsf{in}\, P & \to & \mathsf{let}\, x := V \,\mathsf{in}\, \mathsf{LET}\, y := M \,\mathsf{in}\, P \\
(B'_v) & (\lambda x.M)V & \to & \mathsf{let}\, x := V \,\mathsf{in}\, M \\
(let_v) & \mathsf{let}\, x := V \,\mathsf{in}\, M & \to & [V/x]M \\
(\eta_{let}) & \mathsf{let}\, x := VW \,\mathsf{in}\, x & \to & VW
\end{array}
$$

Figure 4 The kernel of the computational λ-calculus, here named $\underline{ANF}$.

Notice that in the form $\mathsf{let}\, x := VW \,\mathsf{in}\, M$ the immediate sub-expressions are V, W and M – but not VW. For this reason, there is no overlap between the redexes of rules B_v and B'_v, nor between the redexes of rules B'_v and η_{let}.

Our presentation of the kernel is very close to the original one in [25], as detailed in Appendix B.

CPS-translation. We present in this subsection the call-by-value CPS-translation of λC. It is a "refined " translation [4], in the sense that it reduces "administrative redexes" at translation time, as already done in [23].

The target of the translation is the system $\underline{CPS}$, presented in Fig. 5. This target is a subsystem of the λ-calculus (or of Plotkin's call-by-value λ_v-calculus – the "indifference property" [23]), whose expressions are the union of four different classes of λ-terms (commands, continuations, values and terms), and whose reduction rules are either particular cases of rules β and η (the cases of σ_v or η_k, respectively), or are derivable as two β-steps (the case of β_v). Each command or continuation has a unique free occurrence of k, which is a fixed (in the calculus) *continuation variable*. A term is obtained by abstracting this variable over a command. A command is always composed of a continuation K, to which a value may be passed (the form KV), or which is going to instantiate k in the command resulting from an application VW (the form VWK).

There is a simply-typed version of this target, *not* found in [23, 4, 25], defined as follows. Simple types are augmented with a new type $\bot$, and we adopt the usual abbreviation

$\neg A := A \supset \bot$. Then, as defined in Fig. 5, one has: two subclasses of such types, one ranged by $\mathcal{A}$, $\mathcal{A}'$ and the other ranged over by $\mathcal{B}$, $\mathcal{B}'$; four kinds of sequents, one per each syntactic class; and one typing rule for each syntactic constructor.

$$
\begin{array}{rrcl}
\text{(Commands)} & M, N & ::= & KV \,|\, VWK \\
\text{(Continuations)} & K & ::= & \lambda x.M \,|\, k \\
\text{(Values)} & V, W & ::= & \lambda x.P \,|\, x \\
\text{(Terms)} & P & ::= & \lambda k.M
\end{array}
$$

$$
\begin{array}{llcll}
(\sigma_v) & (\lambda x.M)V & \to & [V/x]M & \\
(\beta_v) & (\lambda xk.M)WK & \to & [K/k][W/x]M & \\
(\eta_k) & \lambda x.Kx & \to & K & \text{if } x \notin FV(K)
\end{array}
$$

Types: $\mathcal{A} ::= a \,|\, \mathcal{A} \supset \mathcal{B} \qquad \mathcal{B} ::= \neg\neg\mathcal{A}$

Contexts Γ: sets of declarations $(x : \mathcal{A})$

Sequents: $k : \neg\mathcal{A}, \Gamma \vdash_{\mathsf{CPS}} M : \bot \quad k : \neg\mathcal{A}, \Gamma \vdash_{\mathsf{CPS}} K : \neg\mathcal{A}' \quad \Gamma \vdash_{\mathsf{CPS}} V : \mathcal{A} \quad \Gamma \vdash_{\mathsf{CPS}} P : \mathcal{B}$

$$\frac{k : \neg\mathcal{A}, \Gamma \vdash_{\mathsf{CPS}} K : \neg\mathcal{A}' \quad \Gamma \vdash_{\mathsf{CPS}} V : \mathcal{A}'}{k : \neg\mathcal{A}, \Gamma \vdash_{\mathsf{CPS}} KV : \bot}$$

$$\frac{\Gamma \vdash_{\mathsf{CPS}} V : \mathcal{A} \supset \neg\neg\mathcal{A}' \quad \Gamma \vdash_{\mathsf{CPS}} W : \mathcal{A} \quad k : \neg\mathcal{A}'', \Gamma \vdash_{\mathsf{CPS}} K : \neg\mathcal{A}'}{k : \neg\mathcal{A}'', \Gamma \vdash_{\mathsf{CPS}} VWK : \bot}$$

$$\frac{k : \neg\mathcal{A}, \Gamma, x : \mathcal{A}' \vdash_{\mathsf{CPS}} M : \bot}{k : \neg\mathcal{A}, \Gamma \vdash_{\mathsf{CPS}} \lambda x.M : \neg\mathcal{A}'} \qquad \frac{}{k : \neg\mathcal{A}, \Gamma \vdash_{\mathsf{CPS}} k : \neg\mathcal{A}}$$

$$\frac{\Gamma, x : \mathcal{A} \vdash_{\mathsf{CPS}} P : \mathcal{B}}{\Gamma \vdash_{\mathsf{CPS}} \lambda x.P : \mathcal{A} \supset \mathcal{B}} \qquad \frac{}{\Gamma, x : \mathcal{A} \vdash_{\mathsf{CPS}} x : \mathcal{A}}$$

$$\frac{k : \neg\mathcal{A}, \Gamma \vdash_{\mathsf{CPS}} M : \bot}{\Gamma \vdash_{\mathsf{CPS}} \lambda k.M : \neg\neg\mathcal{A}}$$

Figure 5 The system $\underline{CPS}$.

The CPS-translation is defined in Fig. 6. It comprises: For each $V \in \lambda\mathsf{C}$, a value $V^\dagger$; for each term $M \in \lambda\mathsf{C}$ and continuation $K \in \underline{CPS}$, a command $(M : K)$; for each term $M \in \lambda\mathsf{C}$, a command $M^\star$ and a term $\overline{M}$.

In the typed setting, each simple type A of $\lambda\mathsf{C}$ determines an $\mathcal{A}$-type $A^\dagger$ and a $\mathcal{B}$-type $\overline{A}$, as in Fig. 6. The translation preserves typing, according to the admissible typing rules displayed in the last row of the same figure.

3 Sequent calculus LJQ and its simplification λQ

In this section we start by recapitulating the term calculus for LJQ designed by Dyckhoff-Lengrand [4]. Next we do some preliminary work, by proposing a simplified variant, named λQ, more appropriate for our purposes in this paper. Finally, we also single out the *kernel* of λQ, which is the sub-calculus of "administrative" normal forms. This further simplification will be necessary for the later analysis of CPS.

$$\begin{array}{rcl} x^{\dagger} & = & x \\ (\lambda x.M)^{\dagger} & = & \lambda x.\overline{M} \\ \overline{M} & = & \lambda k.M^{\star} \\ M^{\star} & = & (M : k) \end{array} \qquad \begin{array}{rcll} (V : K) & = & KV^{\dagger} & \\ (PQ : K) & = & (P : \lambda m.(mQ : K)) & (a) \\ (VQ : K) & = & (Q : \lambda n.(Vn : K)) & (b) \\ (VW : K) & = & V^{\dagger}W^{\dagger}K & \\ (\mathsf{let}\, y := M \,\mathsf{in}\, P : K) & = & (M : \lambda y.(P : K)) & \end{array}$$

$$\overline{A} = \neg\neg A^{\dagger} \qquad a^{\dagger} = a \qquad (A \supset B)^{\dagger} = A^{\dagger} \supset \overline{B}$$

$$\frac{\Gamma \vdash_{\mathsf{C}} V : A}{\Gamma^{\dagger} \vdash_{\mathsf{CPS}} V^{\dagger} : A} \qquad \frac{\Gamma \vdash_{\mathsf{C}} M : A \quad k : \neg B^{\dagger}, \Gamma \vdash_{\mathsf{CPS}} K : \neg A^{\dagger}}{k : \neg B^{\dagger}, \Gamma^{\dagger} \vdash_{\mathsf{CPS}} (M : K) : \perp}$$

$$\frac{\Gamma \vdash_{\mathsf{C}} M : A}{k : \neg A^{\dagger}, \Gamma^{\dagger} \vdash_{\mathsf{CPS}} M^{\star} : \perp} \qquad \frac{\Gamma \vdash_{\mathsf{C}} M : A}{\Gamma^{\dagger} \vdash_{\mathsf{CPS}} \overline{M} : \overline{A}}$$

Figure 6 The CPS-translation, from $\lambda\mathsf{C}$ to $\underline{CPS}$, with admissible typing rules. Provisos: (a) P is not a value. (b) Q is not a value.

The original term calculus. An abridged presentation of the original term calculus for LJQ by Dyckhoff-Lengrand is found in Fig. 7 [1]. The separation between terms and values corresponds to the separation between the two kinds of sequents handled by LJQ: the ordinary sequents $\Gamma \Rightarrow M : A$ and the focused sequents $\Gamma \rightarrow V : A$. There are three forms of cut and the reduction rules correspond to cut-elimination rules. We may think of the forms $\mathsf{C}_1(V, x.W)$ and $\mathsf{C}_2(V, x.N)$ as explicit substitutions: in this abridged presentation we omitted the rules for their stepwise execution.

We now introduce a slight modification of λLJQ, named λLjQ, determined by two changes in the reduction rules: in rule (6) we omit the proviso; and rule (5) is dropped. A former redex of (5) is reduced by (6) – now possible because there is no proviso – followed by (4), achieving the same effect as previous rule (5).

In fact, very soon we will define a *big* modification and simplification of the original λLJQ, which is more appropriate to our goals here. But we need to justify that big modification, by a comparison with the original system. For the purpose of this comparison, we will use, not λLJQ, but λLjQ instead. So, the first thing we do is to check that λLjQ has the same properties as the original.

The maps between $\lambda\mathsf{C}$ and λLJQ defined by Dyckhoff-Lengrand can be seen as maps to and from λLjQ instead. Next, it is easy to see that such maps still establish an equational correspondence, now between $\lambda\mathsf{C}$ and λLjQ. It turns out that the correspondence is also a pre-Galois connection from λLjQ to $\lambda\mathsf{C}$. Because of this, λLjQ inherits confluence of $\lambda\mathsf{C}$, as λLJQ did.

A simplified calculus. We now define the announced simplified calculus, named λQ. It is presented in Fig. 8. The idea is to drop the cut forms $\mathsf{C}_1(V, x.W)$ and $\mathsf{C}_2(V, x.N)$, which correspond to explicit substitutions. Since only one form of cut remain, $\mathsf{C}_3(M, x.N)$, we write it as $\mathsf{C}(M, x.N)$. The typing rules of the surviving constructors remain the same. The omitted reduction rules for the stepwise execution of substitution are now dropped, since they concerned the omitted forms of cut. Rules (1) and (3) are renamed as B_v and η_{cut}, respectively. Rules (4) and (6) are renamed π_1 and π_2, respectively, and we let $\pi := \pi_1 \cup \pi_2$. Rules (2) and (7) are combined into a single rule named σ_v.

[1] See Appendix A for the full system.

$$\begin{array}{llcl}
\text{(terms)} & M,N & ::= & \uparrow V \mid x(V,y.N) \mid \mathsf{C}_2(V,x.N) \mid \mathsf{C}_3(M,x.N) \\
\text{(values)} & V,W & ::= & x \mid \lambda x.M \mid \mathsf{C}_1(V,x.W)
\end{array}$$

$$\begin{array}{lrcll}
(1) & \mathsf{C}_3(\uparrow(\lambda x.M), y.y(V,z.N)) & \to & \mathsf{C}_3(\mathsf{C}_3(\uparrow V, x.M), z.N) & (a) \\
(2) & \mathsf{C}_3(\uparrow x, y.N) & \to & [x/y]N & \\
(3) & \mathsf{C}_3(M, x.\uparrow x) & \to & M & \\
(4) & \mathsf{C}_3(z(V,y.P), x.N) & \to & z(V, y.\mathsf{C}_3(P,x.N)) & \\
(5) & \mathsf{C}_3(\mathsf{C}_3(\uparrow W, y.y(V,z.P)), x.N) & \to & \mathsf{C}_3(\uparrow W, y.y(V, z.\mathsf{C}_3(P,x.N))) & (b) \\
(6) & \mathsf{C}_3(\mathsf{C}_3(M,y.P), x.N) & \to & \mathsf{C}_3(M, y.\mathsf{C}_3(P,x.N)) & (c) \\
(7) & \mathsf{C}_3(\uparrow(\lambda x.M), y.N) & \to & \mathsf{C}_2(\lambda x.M, y.N) & (d)
\end{array}$$

$$\frac{}{\Gamma, x : A \to x : A}\,Ax \qquad \frac{\Gamma \to V : A}{\Gamma \Rightarrow \uparrow V : A}\,Der$$

$$\frac{\Gamma, x : A \Rightarrow M : B}{\Gamma \to \lambda x.M : A \supset B}\,R\supset \qquad \frac{\Gamma \Rightarrow M : A \quad \Gamma, x : A \Rightarrow N : B}{\Gamma \Rightarrow \mathsf{C}_3(M,x.N) : B}\,Cut_3$$

$$\frac{\Gamma, x : A \supset B \to V : A \quad \Gamma, x : A \supset B, y : B \Rightarrow N : C}{\Gamma, x : A \supset B \Rightarrow x(V,y.N) : C}\,L\supset$$

Figure 7 The original calculus by Dyckhoff-Lengrand, here named λLJQ-calculus (abridged). Provisos: (a) $y \notin FV(V) \cup FV(N)$. (b) $y \notin FV(V) \cup FV(P)$). (c) If rule (5) does not apply. (d) If rule (1) does not apply.

The design of rule σ_v is interesting. Rule (2) fired a variable substitution operation $[x/y]-$, already present in the original calculus. The contractum of rule (7), being an explicit substitution, has to be replaced by the call to an appropriate, *implicit*, substitution operator $[\lambda x.M/y]-$, whose stepwise execution should be coherent with the omitted reduction rules for $\mathsf{C}_1(V,x.W)$ and $\mathsf{C}_2(V,x.N)$. Hopefully, the sought operation and the already present variable substitution operation are subsumed by a value substitution operation $[V/y]-$.

The critical clause is the definition of $[V/y](y(W,z.P))$. We adopt $[V/y](y(W,z.P)) = \mathsf{C}(\uparrow V, y.y([V/y]W, z.[V/y]P))$ in the case $V = \lambda x.M$, but not in the case of $V = x$, because σ_v would immediately generate a cycle in the case $y \notin FV(V) \cup FV(N)$. We adopt instead $[x/y](y(W,z.P)) = x([x/y]W, z.[x/y]P)$ which moreover is what the original calculus dictates. Notice that another cycle would arise, if a B_v-redex was contracted by σ_v. But this is blocked by the proviso of the latter rule.

There is a map $(\_)^{\surd} : \lambda LjQ \to \lambda Q$, based on the idea of translating the omitted cuts by calls to substitution: $\mathsf{C}_1(V,x.W)$ is mapped to $[V/x]W$ and $\mathsf{C}_2(V,y.N)$ is mapped to $[V/x]N$. This map, together with the inclusion $\lambda Q \subset \lambda LjQ$ (seeing $\mathsf{C}(M,x.N)$ as $\mathsf{C}_3(M,x.N)$) gives a reflection of λQ in λLjQ. This reflection allows to conclude easily that reduction in λLjQ is conservative over reduction in λQ. Moreover, this reflection can be composed with the equational correspondence between $\lambda\mathsf{C}$ and λLjQ to produce an equational correspondence between $\lambda\mathsf{C}$ and λQ. Finally, this reflection is also a pre-Galois connection from λQ to λLjQ. Thus, confluence of λQ can be pulled back from the confluence of λLjQ.

To sum up, we obtained a more manageable calculus, conservatively extended by the original one, which, as the latter, is confluent and is in equational correspondence with $\lambda\mathsf{C}$.

$$\begin{array}{rrcl}
\text{(terms)} & M,N & ::= & \uparrow V \mid x(V,y.N) \mid \mathsf{C}(M,x.N) \\
\text{(values)} & V,W & ::= & x \mid \lambda x.M
\end{array}$$

$$\begin{array}{rrcll}
(B_v) & \mathsf{C}(\uparrow(\lambda x.M),y.y(V,z.N)) & \to & \mathsf{C}(\mathsf{C}(\uparrow V,x.M),z.N) & \text{if } y\notin FV(V)\cup FV(N) \\
(\sigma_v) & \mathsf{C}(\uparrow V,y.N) & \to & [V/y]N & \text{if } B_v \text{ does not apply} \\
(\eta_{cut}) & \mathsf{C}(M,x.\uparrow x) & \to & M & \\
(\pi_1) & \mathsf{C}(z(V,y.P),x.N) & \to & z(V,y.\mathsf{C}(P,x.N)) & \\
(\pi_2) & \mathsf{C}(\mathsf{C}(M,y.P),x.N) & \to & \mathsf{C}(M,y.\mathsf{C}(P,x.N)) &
\end{array}$$

Figure 8 The simpified λLJQ-calculus, named λQ-calculus.

The kernel of the simplified calculus. For a moment, we do an analogy between $\lambda\mathsf{C}$ and λQ. As was recalled in Section 2, the former system admits a *kernel*, a subsystem of "administrative" normal forms, which are the normal forms with respect to a subset of the set of reduction rules [25]. For λQ, the "administrative" normal forms are very easy to characterize: in a cut $\mathsf{C}(M,x.N)$, M has to be of the form $\uparrow V$. Logically, this means that the left premiss of the cut comes from a sequent $\Gamma \to V : A$; given that such sequents are obtained either with Ax or $R\supset$, the cut formula A in that premiss is not a passive formula of the previous inference; hence the cut is fully permuted to the left – so we call such forms *left normal forms*. The reduction rules of λQ which perform left permutation are rules π_1 and π_2 (even though textually the outer cut in the redex of those rules seems to move to the right after the reduction), so these rules are declared "administrative".

The kernel of λQ is named $\underline{LNF}$. The specific form of cut allowed, namely $\mathsf{C}(\uparrow V,x.N)$, is written $\mathsf{C}_v(V,x.N)$. No other change is made to the grammar of terms. Given $M,N\in\underline{LNF}$, the general form of cut becomes in $\underline{LNF}$ a derived constructor written $\mathsf{C}_v(M:z.N)$ and defined by recursion on M as follows:

$$\begin{array}{rcl}
\mathsf{C}_v(\uparrow V : z.N) & = & \mathsf{C}_v(V,z.N) \\
\mathsf{C}_v(x(V,y.M) : z.N) & = & x(V,y.\mathsf{C}_v(M:z.N)) \\
\mathsf{C}_v(\mathsf{C}_v(V,y.M) : z.N) & = & \mathsf{C}_v(V,y.\mathsf{C}_v(M:z.N))
\end{array}$$

As to reduction rules, rule B_v in $\underline{LNF}$ reads

$$\mathsf{C}_v(\lambda x.M,y.y(V,z.N)) \to \mathsf{C}_v(V,x.\mathsf{C}_v(M:z.N)) \ .$$

Notice that the contractum is the same as $\mathsf{C}_v(\mathsf{C}_v(\uparrow V : x.M) : z.N)$. The proviso remains the same: $y\notin FV(V)\cup FV(N)$. As to the other reduction rules: there is no change to rule σ_v; the specific form of rule η_{cut} that survives becomes a particular case of σ_v, hence is omitted; and the system has no π-rules.

There is a map $(\_)^{\triangledown} : \lambda Q \to \underline{LNF}$ based on the idea of replacing $\mathsf{C}(M,x.N)$ by $\mathsf{C}_v(M:x.N)$. This map, together with the inclusion $\underline{LNF}\subset\lambda Q$ (seeing $\mathsf{C}_v(V,x.N)$ as $\mathsf{C}(\uparrow V,x.N)$), gives a reflection in λQ of $\underline{LNF}$. Quite obviously, $M \twoheadrightarrow_\pi M^{\triangledown}$; in fact $M^{\triangledown}$ is a π-normal form, as are all the expressions of $\underline{LNF}$.

$\underline{LNF}$ is a stepping stone in the way to the definition, in the next section, of the value-filling style fragment, which will be a central player in this paper.

4 The value-filling style

In this section we define the target language VFS (a fragment of $\underline{LNF}$) of a new compilation of $\lambda\mathsf{C}$, the value-filling style translation. Next we slightly modify the target $\underline{CPS}$, and introduce the negative translation, mapping VFS to the modified $\underline{CPS}$. Then we show that the CPS-translation is decomposed in terms of the alternative compilation and the negative translation; and that the negative translation is in fact an isomorphism.

The sub-kernel of LJQ. We now define the *sub-kernel* of λQ, a language named VFS that will serve as a target language for compilation alternative to CPS. Despite the simplicity of λQ, there is still room for a simplification: to forbid the left-introduction constructor $y(W, x.M)$ to stand as a term on its own. However, we regret that, by that omission, that term cannot be used in a very particular situation: as the term N in $\mathsf{C}_v(V, y.N)$, when $y \notin FV(W) \cup FV(M)$. So, we keep that particular combination of cut and left-introduction as a separate form of cut. The result is presented in Fig. 9.

$$\begin{array}{rrcl}
\text{(terms)} & M, N & ::= & \uparrow V \mid \mathsf{C}_v(V, c) \\
\text{(values)} & V, W & ::= & x \mid \lambda x.M \\
\text{(formal contexts)} & c & ::= & x.M \mid (W, x.M)
\end{array}$$

$$\begin{array}{lrcl}
(B_v) & \mathsf{C}_v(\lambda x.M, (V, y.N)) & \to & \mathsf{C}_v(V, x.\mathsf{C}_v(M : y.N)) \\
(\sigma_v) & \mathsf{C}_v(V, y.N) & \to & [V/y]N
\end{array}$$

$$\frac{\Gamma \to V : A \quad \Gamma|A \Rightarrow c : B}{\Gamma \Rightarrow \mathsf{C}_v(V, c) : B} \qquad \frac{\Gamma, x : A \Rightarrow M : B}{\Gamma|A \Rightarrow x.M : B} \qquad \frac{\Gamma \to W : A \quad \Gamma, x : B \Rightarrow M : C}{\Gamma|A \supset B \Rightarrow (W, x.M) : C}$$

Figure 9 The sub-kernel of the λQ, named VFS. Typing rules for $\uparrow V$, x and $\lambda x.M$ as before.

In fact, we introduce a third syntactic class, that of *formal contexts* – this terminology will be justified later. Think of $(W, x.M)$ as $y.y(W, x.M)$ with $y \notin FV(W) \cup FV(M)$. The new class allows us to account uniformly for the two possible forms of cut: $\mathsf{C}_v(V, c)$. The reduction rules of VFS are those of the kernel $\underline{LNF}$, restricted to the sub-kernel: pleasantly, the side conditions have vanished! Moreover, the operation $[V/y]N$ is now plain substitution.

There is, again, an auxiliary operation used in the contractum of B_v. Cut $\mathsf{C}_v(M : c')$ and formal context $(c : c')$ are defined by simultaneous recursion on M and c as follows:

$$\begin{array}{rcl@{\qquad}rcl}
\mathsf{C}_v(\uparrow V : c') & = & \mathsf{C}_v(V, c') & ((x.M) : c') & = & x.\mathsf{C}_v(M : c') \\
\mathsf{C}_v(\mathsf{C}_v(V, c) : c') & = & \mathsf{C}_v(V, (c : c')) & ((W, x.M) : c') & = & (W, x.\mathsf{C}_v(M : c'))
\end{array}$$

In the type system, a third form of sequents is added for the typing of formal contexts. We know the formula A in $\Gamma \to V : A$ is a focus [4], but the formula A in $\Gamma|A \Rightarrow c : B$ is not, since it can simply be selected from the context Γ in the typing rule for $x.M$.

We already know how to map VFS back to $\underline{LNF}$. How about the inverse direction? How do we compensate the omission of $y(W, x.M)$? The answer is: by the following expansion

$$y(W, x.M) \leftarrow_{\sigma_v} \mathsf{C}_v(y, z.z(W, x.M)) = \mathsf{C}_v(y, (W, x.M)) \tag{1}$$

The VFS-translation. The system VFS is the target of a translation of $\lambda\mathsf{C}$ alternative to the CPS-translation, to be introduced now. The idea is to represent a term of $\lambda\mathsf{C}$, not as a command of CPS (in terms of a continuation that is called of passed), but rather as a cut of the sequent calculus VFS, making use of "formal contexts". Later, we will give a detailed comparison with the CPS-translation, which will make sense of the terminology "formal context" and "value-filling"; more importantly, the comparison will show that VFS and the translation into it is a style equivalent to CPS, but much simpler, in particular due to this very objective fact: there is no translation of types involved.

The VFS-translation is given in Fig. 10. It comprises: For each $V \in \lambda\mathsf{C}$, a value V° in VFS; for each $M \in \lambda\mathsf{C}$ and formal context $c \in VFS$, a cut $(M;c)$ in VFS; for each $M \in \lambda\mathsf{C}$, a cut $M^\bullet$ in VFS. Again: there is no translation of types.

$$
\begin{array}{rcl}
x^\circ & = & x \\
(\lambda x.M)^\circ & = & \lambda x.M^\bullet \\
\\
M^\bullet & = & (M; x.\uparrow x)
\end{array}
\qquad
\begin{array}{rcll}
(V;x.N) & = & \mathsf{C}_v(V^\circ, x.N) & \\
(PQ;x.N) & = & (P; m.(mQ;x.N)) & (*) \\
(VQ;x.N) & = & (Q; n.(Vn;x.N)) & (**) \\
(VW;x.N) & = & \mathsf{C}_v(V^\circ, (W^\circ, x.N)) & \\
(\mathsf{let}\, y := M \,\mathsf{in}\, P; x.N) & = & (M; y.(P;x.N)) &
\end{array}
$$

$$
\frac{\Gamma \vdash_{\mathsf{C}} V : A}{\Gamma \to V^\circ : A}
\qquad
\frac{\Gamma \vdash_{\mathsf{C}} M : A \quad \Gamma | A \Rightarrow c : B}{\Gamma \Rightarrow (M;c) : B}
\qquad
\frac{\Gamma \vdash_{\mathsf{C}} M : A}{\Gamma \Rightarrow M^\bullet : A}
$$

Figure 10 The VFS-translation, from $\lambda\mathsf{C}$ to VFS. Provisos: $(*)$ P is not a value. $(**)$ Q is not a value.

▶ **Theorem 1** (Simulation).

1. *Let $R \in \{B, let_v, \eta_{let}\}$. If $M \to_R N$ in $\lambda\mathsf{C}$ then $M^\bullet \twoheadrightarrow N^\bullet$ in VFS.*
2. *Let $R \in \{let_1, let_2, assoc\}$. If $M \to_R N$ in $\lambda\mathsf{C}$ then $M^\bullet = N^\bullet$ in VFS.*

The language CPS. Recall the CPS-translation of $\lambda\mathsf{C}$, given in Fig. 6, with target system $\underline{CPS}$, given in Fig. 5, our own reworking of Reynold's translation and respective target [4]. We now introduce a tiny modification in the CPS-translation, an η-expansion of k in the definition of $M^\star$: $M^\star = (M : \lambda x.kx)$. This requires a slight modification of the target system. First, the grammar of commands and continuations becomes:

$$
\text{(Commands)} \quad M, N \quad ::= \quad kV \,|\, KV \,|\, VWK \qquad \text{(Continuations)} \quad K \quad ::= \quad \lambda x.M
$$

The continuation variable k is no longer by itself a continuation – but nothing is lost with respect to $\underline{CPS}$, since k may be expanded thus:

$$
k \leftarrow_{\eta_k} \lambda x.kx \tag{2}
$$

Since K is now necessarily a λ-abstraction, the η_k-reduction $\lambda x.Kx \to K$ of $\underline{CPS}$ becomes a σ_v-reduction in the modified target, and so the latter system has no rule η_k.

We do a further modification to the reduction rules: instead of following [25] and having rule β_v, we prefer that the modified target system has the rule $(\lambda xk.M)WK \to (\lambda x.[K/k]M)W$, named B_v. That is, we substitute K, but not W.[2] The new contractum is a σ_v-redex, that can be immediately reduced to produce the effect of $\underline{CPS}$'s rule β_v.

[2] We could have made this modification in Fig. 5, without any change to our results. The only thing to observe is that, if we want $\underline{CPS}$ (or its modification) to consists of syntax that is derivable from the ordinary λ-calculus or Plotkin's call-by-value λ-calculus, then we have to consider these systems equipped with the well-known permutation $(\lambda x.M)VV' \to (\lambda x.MV')V$.

In the typed case, the typing rule for k is replaced by this one:

$$\frac{\Gamma \vdash_{\mathsf{CPS}} V : \mathcal{A}}{k : \neg\mathcal{A}, \Gamma \vdash_{\mathsf{CPS}} kV :\bot}$$

No other modification is introduced w. r. t. Fig. 5. The obtained system is named CPS.

For the modified CPS-translation, we reuse the notation $\overline{M}$, $M^{\star}$, $V^{\dagger}$ and $(M : K)$. From now on, "CPS-translation" refers to the modified one, while the original one will be called $\underline{\text{CPS}}$-translation.

In $\underline{CPS}$, k is a fixed continuation variable. In CPS, k is a fixed *covariable*, again occurring exactly once in each command and continuation. The word "covariable" intends to be reminiscent of the covariables, or "names", of the $\lambda\mu$-calculus [22]. Accordingly, kV is intended to be reminiscent of the naming constructor of that calculus, and some "structural substitution" should be definable in CPS.

Indeed, consider the following notion of *context* for CPS: $\mathbb{C} ::= K[\_] \,|\, [\_]WK$. Filling the hole $[\_]$ of $\mathbb{C}$ with V results in the command $\mathbb{C}[V]$. Then, we can define the structural substitution operation $[\mathbb{C}/k]-$ whose critical clause is $[\mathbb{C}/k](kV) = \mathbb{C}[V]$. There is no need to recursively apply the operation to V, since $k \notin FV(V)$.

Now in the case $\mathbb{C} = K[\_]$, the structural substitution $[\mathbb{C}/k]-$ is the same operation as the ordinary substitution $[K/k]-$, and it turns out that we will only need this case of substitution. That is why we will not see the structural substitution anymore in this paper.

However, contexts $\mathbb{C}$ will be crucial for understanding the relationship between VFS and CPS. In preparation for that, we derive typing rules for contexts of CPS. The corresponding sequents are of the form $\Gamma|A \vdash_{\mathsf{CPS}} \mathbb{C} :\bot$, where A is the type of the hole of $\mathbb{C}$. Hence, the command $\mathbb{C}[V]$ is typed as follows:

$$\frac{\Gamma \vdash_{\mathsf{CPS}} V : A \quad \Gamma|A \vdash_{\mathsf{CPS}} \mathbb{C} :\bot}{\Gamma \vdash_{\mathsf{CPS}} \mathbb{C}[V] :\bot}\ \mathbb{C}1$$

The rules for typing $\mathbb{C}$ are obtained from the rules for typing KV and VWK in Fig. 5, erasing the premise relative to V and declaring V's type as the type of the hole of $\mathbb{C}$:

$$\frac{k : \neg\mathcal{A}, \Gamma \vdash_{\mathsf{CPS}} K : \neg\mathcal{A}'}{k : \neg\mathcal{A}, \Gamma|\mathcal{A}' \vdash_{\mathsf{CPS}} K[\_] :\bot}\ \mathbb{C}2 \qquad \frac{\Gamma \vdash_{\mathsf{CPS}} W : \mathcal{A} \quad k : \neg\mathcal{A}'', \Gamma \vdash_{\mathsf{CPS}} K : \neg\mathcal{A}'}{k : \neg\mathcal{A}'', \Gamma|\mathcal{A} \supset \neg\neg\mathcal{A}' \vdash_{\mathsf{CPS}} [\_]WK :\bot}\ \mathbb{C}3$$

We also observe that $K_{\mathbb{C}} := \lambda z.\mathbb{C}[z]$ is a continuation, and that $K_{\mathbb{C}}V \to_{\sigma_v} \mathbb{C}[V]$ in CPS.

VFS vs CPS: the negative translation. We now see that the CPS-translation can be decomposed as the VFS-translation followed by a *negative* translation of system VFS. This latter translation is a CPS-translation, hence involving, at the level of types, the introduction of double negations (hence the name "negative"). It turns out that this negative translation is an isomorphism between VFS and CPS, at the levels of proofs and proof reduction. This renders the last stage of translation (the negative stage) and its style of representation (the CPS style) an optional addition to what is already achieved with VFS.

The negative translation is found in Fig. 11. It comprises: For each $V \in VFS$, a value $V^{\sim}$ in CPS; for each $M \in VFS$, a command $M^{\wr}$ and a term M^{-} in CPS.

The translation has a typed version, mapping between the typed version of source and target calculi. This requires a translation of types: for each simple type A of VFS, there is an $\mathcal{A}$-type $A^{\sim}$ and a $\mathcal{B}$-type A^{-}, as defined in Fig. 11. The translation preserves typing, according to the admissible rules displayed in the last row of the same table.

$$
\begin{array}{rcl@{\qquad}rcl}
x^{\sim} & = & x & (\uparrow V)^{\wr} & = & kV^{\sim} \\
(\lambda x.M)^{\sim} & = & \lambda x.M^{-} & \mathsf{C}_v(V, x.M)^{\wr} & = & (\lambda x.M^{\wr})V^{\sim} \\
M^{-} & = & \lambda k.M^{\wr} & \mathsf{C}_v(V, (W, x.M))^{\wr} & = & V^{\sim}W^{\sim}(\lambda x.M^{\wr})
\end{array}
$$

$$
A^{-} = \neg\neg A^{\sim} \qquad a^{\sim} = a \qquad (A \supset B)^{\sim} = A^{\sim} \supset B^{-}
$$

$$
\frac{\Gamma \rightarrow V : A}{\Gamma^{\sim} \vdash_{\mathsf{CPS}} V^{\sim} : A^{\sim}} \quad \frac{\Gamma \Rightarrow M : A}{k : \neg A^{\sim}, \Gamma^{\sim} \vdash_{\mathsf{CPS}} M^{\wr} :\perp} \quad \frac{\Gamma \Rightarrow M : A}{\Gamma^{\sim} \vdash_{\mathsf{CPS}} M^{-} : A^{-}}
$$

Figure 11 The negative translation, from VFS to CPS, with admissible typing rules.

The negative translation is defined at the level of terms and values. How about formal contexts? A formal context c is translated as a context $c^{\wr}$ of CPS, defined as follows:

$$
(x.M)^{\wr} \quad = \quad (\lambda x.M^{\wr})[\_] \qquad\qquad (W, x.M)^{\wr} \quad = \quad [\_]W^{\sim}(\lambda x.M^{\wr})
$$

Then the definition of $\mathsf{C}_v(V, c)^{\wr}$ can be made uniform in c as $c^{\wr}[V^{\sim}]$. The translation of non-values $\mathsf{C}_v(V, c)^{\wr}$ is thus defined as filling the (translation) of V in the hole of the actual context $c^{\wr}$ that translates the formal context c. Hence the name "value-filling" of the translation.

We have two admissible typing rules:

$$
\frac{\Gamma|A \Rightarrow c : B}{k : \neg B^{\sim}, \Gamma^{\sim}|A^{\sim} \vdash_{\mathsf{CPS}} c^{\wr} :\perp}\ (a) \qquad \frac{\Gamma|A \Rightarrow c : B}{k : \neg B^{\sim}, \Gamma^{\sim} \vdash_{\mathsf{CPS}} K_{c^{\wr}} : \neg A^{\sim}}\ (b)
$$

Rule (a) follows from typing rules $\mathbb{C}2$ and $\mathbb{C}3$; rule (b) is obtained from (a) and rule $\mathbb{C}1$.

It is no exaggeration to say that typing rule (b) is the heart of the negative translation. In the sequent calculus VFS we can single out a formula A in the l. h. s. of the sequent to act as the type of the hole of a (formal) context c. In CPS, we have the related concept of a continuation K, a function of type $A \supset\perp$. The type B of c has to be stored as the negated type $\neg B$ of a special variable k. Cutting with c in the sequent calculus corresponds to applying K, to obtain a command, of type $\perp$. But the cut produces a term of type B, while the best we can do in CPS is to abstract k, to obtain $\neg\neg B$. In the sequent calculus, a type A may have uses in both sides of the sequent. To approximate this flexibility in CPS, a type A requires types $\mathcal{A}$, $\neg\mathcal{A}$, and $\neg\neg\mathcal{A} = \mathcal{B}$, presupposing $\perp$.

▶ **Theorem 2** (Decomposition of the CPS-translation)**.**

1. *For all* $V \in \lambda\mathsf{C}$, $V^{\circ\sim} = V^{\dagger}$.
2. *For all* $M \in \lambda\mathsf{C}$, $N \in VFS$, $(M; x.N)^{\wr} = (M : \lambda x.N^{\wr})$.
3. *For all* $M \in \lambda\mathsf{C}$, $M^{\bullet\wr} = M^{\star}$.
4. *For all* $M \in \lambda\mathsf{C}$, $M^{\bullet -} = \overline{M}$.

Nothing is lost, if we wish to replace CPS with VFS, because the negative translation is an isomorphism. Its inverse translation comprises: For each term $P \in CPS$, a term $P^{+} \in VFS$; for each command $M \in CPS$, a term $M^{\times} \in VFS$; for each value $V \in CPS$, a value $V^{\times\!\!\times} \in VFS$. The definition is as follows:

$$
\begin{array}{rcl}
(\lambda k.M)^{+} & = & M^{\times} \\
(kV)^{\times} & = & \uparrow(V^{\times\!\!\times}) \\
((\lambda x.M)V)^{\times} & = & \mathsf{C}_v(V^{\times\!\!\times}, x.M^{\times}) \\
(VW(\lambda x.M))^{\times} & = & \mathsf{C}_v(V^{\times\!\!\times}, (W^{\times\!\!\times}, x.M^{\times})) \\
x^{\times\!\!\times} & = & x \\
(\lambda x.P)^{\times\!\!\times} & = & \lambda x.P^{+}
\end{array}
$$

▶ **Theorem 3** ($VFS \cong CPS$)**.**

1. *For all $M, V \in VFS$, $M^{-^{+}} = M$ and $M^{\wr^{\times}} = M$ and $V^{\sim^{\divideontimes}} = V$.*
2. *For all $P, M, V \in CPS$, $P^{+^{-}} = P$ and $M^{\times^{\wr}} = M$ and $V^{\divideontimes^{\sim}} = V$.*
3. *If $M_1 \to M_2$ in VFS then $M_1^{\wr} \to M_2^{\wr}$ in CPS (hence $M_1^{-} \to M_2^{-}$ in CPS).*
4. *If $M_1 \to M_2$ in CPS then $M_1^{\times} \to M_2^{\times}$ in VFS. Hence If $P_1 \to P_2$ in CPS then $P_1^{+} \to P_2^{+}$ in VFS.*

5 Back to direct style

We now do to the VFS-translation what [10, 25] did to the CPS-translation, that is, try to find a program transformation in the source language λC that corresponds to the effect of the translation. We have seen in Section 4 that the VFS-translation identifies reduction steps generated by let_1, let_2 and $assoc$. So we start from the normal forms w. r. t. these rules, that is, from the kernel $\underline{ANF}$ (recall Fig. 4). We first identify two sub-syntaxes relevant in this analysis. Next, we point out the proof-theoretical meaning of such alternative.

Two sub-kernels of $\underline{ANF}$. It turns out that the syntax of $\underline{ANF}$, despite its simplicity, still contains several dilemmas: (1) Do we need a let-expression whose actual parameter is a value V? Or should we normalize with respect to let_v? (2) Do we need VW to stand alone as a term and also as the actual parameter of a let-expression? (3) Is η_{let} a reduction or an expansion? Some of these dilemmas give rise to the following diagram:

$$\begin{tikzcd}
VW & & \mathsf{let}\, x := V \,\mathsf{in}\, xW \arrow[ll, "let_v"'] \\
\mathsf{let}\, y := VW \,\mathsf{in}\, y \arrow[u, "\eta_{let}"] & & \mathsf{let}\, x := V \,\mathsf{in}\, \underbrace{\mathsf{let}\, y := xW \,\mathsf{in}\, y}_{c_x} \arrow[ll, "let_v"] \arrow[u, "\eta_{let}"']
\end{tikzcd} \qquad (3)$$

We take this diagram as giving, in its lower row, two different ways of expanding VW. These two alternatives signal two sub-syntaxes of $\underline{ANF}$ without VW. In the alternative corresponding to the expansion $\mathsf{let}\, y := VW \,\mathsf{in}\, y$, we are free to, additionally, normalize w. r. t. let_v and get rid of the form $\mathsf{let}\, x := V \,\mathsf{in}\, M$. In the alternative $\mathsf{let}\, x := V \,\mathsf{in}\, \mathsf{let}\, y := xW \,\mathsf{in}\, y$, we are not free to normalize w. r. t. let_v, as otherwise we might reverse the intended expansions. In both cases, values are $V, W ::= x \,|\, \lambda x.M$. Moreover, we do not want to consider η_{let} as a reduction rule; and rule B_v' disappears, since there are no applications VW.

In the first sub-kernel, named *CES*, terms M are given by the grammar

$$M ::= V \,|\, \mathsf{let}\, x := VW \,\mathsf{in}\, M \quad .$$

We call this representation *continuation enclosing* style, since the "serious" (=non-value) terms have the form of an application VW enclosed in a let-expression. The unique reduction rule of *CES* is

$$(\beta_v) \qquad \mathsf{let}\, y := (\lambda x.M)V \,\mathsf{in}\, P \to \mathsf{LET}\, y := [V/x]M \,\mathsf{in}\, P$$

In $\underline{ANF}$, it corresponds to a B_v-step followed by let_v-step. The operation $\mathsf{LET}\, y := M \,\mathsf{in}\, P$ of $\underline{ANF}$ is reused, except that the base case of its definition integrates a further let_v-step: $\mathsf{LET}\, y := V \,\mathsf{in}\, P = [V/y]P$.

In the second sub-kernel, named VES, terms are given by the grammar

$$\begin{array}{rcl} M, N & ::= & V \mid \mathsf{let}\, x := V \,\mathsf{in}\, c_x \\ c_x & ::= & M \mid \mathsf{let}\, y := xW \,\mathsf{in}\, N, \ \text{where } x \notin FV(W) \cup FV(N) \end{array}$$

We call this representation *value enclosed* style, since the serious terms have the form of a value enclosed in a let-expression. There are two reduction rules:

$$\begin{array}{lrcl} (B_v) & \mathsf{let}\, y := (\lambda x.M) \,\mathsf{in}\, \mathsf{let}\, z := yV \,\mathsf{in}\, P & \to & \mathsf{let}\, x := V \,\mathsf{in}\, \mathsf{LET}\, z := M \,\mathsf{in}\, P \\ (let_v) & \mathsf{let}\, y := V \,\mathsf{in}\, N & \to & [V/y]N \end{array}$$

In VES, we define $\mathsf{LET}\, y := M \,\mathsf{in}\, P$ and $\mathsf{LET}\, y := c_z \,\mathsf{in}\, P$, which are a term and an element of the class c_z, respectively, the latter satisfying $z \notin FV(P)$. The definition is by simultaneous recursion on M and c_z as follows:

$$\begin{array}{rcl} \mathsf{LET}\, y := V \,\mathsf{in}\, P & = & \mathsf{let}\, y := V \,\mathsf{in}\, P \\ \mathsf{LET}\, y := (\mathsf{let}\, z := V \,\mathsf{in}\, c_z) \,\mathsf{in}\, P & = & \mathsf{let}\, z := V \,\mathsf{in}\, \mathsf{LET}\, y := c_z \,\mathsf{in}\, P \\ \mathsf{LET}\, y := (\mathsf{let}\, x := zW \,\mathsf{in}\, N) \,\mathsf{in}\, P & = & \mathsf{let}\, x := zW \,\mathsf{in}\, \mathsf{LET}\, y := N \,\mathsf{in}\, P \end{array}$$

In the second equation, since in the l. h. s. P is not in the scope of the (inner) let-expression, we may assume $z \notin FV(P)$. So, the proviso for the call $\mathsf{LET}\, y := c_z \,\mathsf{in}\, P$ in the r. h. s. is satisfied. In the third equation, c_z in the l. h. s. is $\mathsf{let}\, x := zW \,\mathsf{in}\, N$. By definition of c_z, $z \notin FV(W) \cup FV(N)$; moreover, we may assume $z \notin FV(P)$: hence the r. h. s. is in c_z.

Despite the trouble with variable conditions, this definition corresponds to the operator $\mathsf{LET}\, y := M \,\mathsf{in}\, P$ of $\underline{ANF}$ restricted to the syntax of VES. Therefore, rule B_v of VES corresponds, in $\underline{ANF}$, to a let_v-step followed by a B_v-step.

Proof-theoretical alternative. We now see that VES is related to the sequent calculus VFS, while $\textsc{ces}$ is related to a fragment $\textsc{cnf}$ of the call-by-value λ-calculus with generalized applications $\lambda \mathsf{J}_v$ introduced in [6]. In both cases, the relation is an isomorphism, in the sense of a type-preserving bijection with a 1-1 simulation of reduction steps.

▶ **Theorem 4.** *$VES \cong VFS$ and $\textsc{ces} \cong \textsc{cnf}$.*

Therefore the alternative between the two sub-kernels corresponds to the alternative between two proof-systems for call-by-value, the sequent calculus LJQ and the natural deduction system with general elimination rules behind $\lambda \mathsf{J}_v$.

A $\lambda \mathsf{J}_v$-term is either a value or a generalized applications $M(N, x.P)$, with typing rule

$$\frac{\Gamma \vdash_{\mathsf{J}} M : A \supset B \quad \Gamma \vdash_{\mathsf{J}} N : A \quad \Gamma, x : B \vdash_{\mathsf{J}} P : C}{\Gamma \vdash_{\mathsf{J}} M(N, x.P) : C}$$

If the head term M is itself an application $M_1(M_2, y.M_3)$, then M_3 has type $A \supset B$ and the term can be rearranged as $M_1(M_2, y.M_3(N, x.P))$, to bring M_3 and N together. This is a known *commutative conversion* [15], here named π_1, which aims to convert the head term M to a value V. On the other hand, if the argument N is itself an application $N_1(N_2, y.N_3)$, then N_3 has type A and the term can be rearranged as $N_1(N_2, y.M(N_3, x.P))$, to bring M and N_3 together. This is a conversion π_2 which has *not* been studied, and which aims to convert the argument N to a value W.

The combined effect of $\pi := \pi_1 \cup \pi_2$ is to reduce generalized applications to the form $V(W, x.P)$, called *commutative normal form.* On these forms, the β_v-rule of $\lambda \mathsf{J}_v$ reads

$$(\beta_v) \qquad (\lambda y.M)(W, x.P) \to [[W/y]M \backslash x]P$$

The *left substitution* operation $[N\backslash x]P$ is defined by

$$[V\backslash x]P = [V/x]P \qquad\qquad [V(W, y.N_3)\backslash x]P = V(W, y.[N_3\backslash x]P)$$

The commutative normal forms, equipped with β_v, constitute the system $\textsc{cnf}$.

$$\begin{array}{rcl}
\Psi(V) & = & \uparrow\Psi_v(V) \\
\Psi(\mathsf{let}\, x := V \,\mathsf{in}\, c_x) & = & \mathsf{C}_v(\Psi_v V, \Psi_x(c_x)) \\
\Psi_v(x) & = & x \\
\Psi_v(\lambda x.M) & = & \lambda x.\Psi M \\
\Psi_x(M) & = & x.\Psi M \\
\Psi_x(\mathsf{let}\, y := xW \,\mathsf{in}\, N) & = & (\Psi W, y.\Psi N) \\
\\
\Theta(\uparrow V) & = & \Theta_v(V) \\
\Theta(\mathsf{C}_v(V, c)) & = & \mathsf{let}\, x := \Theta_v V \,\mathsf{in}\, \Theta_x(c) \\
\Theta_v(x) & = & x \\
\Theta_v(\lambda x.M) & = & \lambda x.\Theta M \\
\Theta_x(y.M) & = & [x/y](\Theta M) \\
\Theta_x(W, y.N) & = & \mathsf{let}\, y := x(\Theta_v W) \,\mathsf{in}\, \Theta N
\end{array}$$

Figure 12 Translation from VES to VFS and vice-versa.

$$\begin{array}{rcl}
\Upsilon(x) & = & x \\
\Upsilon(\lambda x.M) & = & \lambda x.\Upsilon M \\
\Upsilon(\mathsf{let}\, x := VW \,\mathsf{in}\, M) & = & \Upsilon V(\Upsilon W, x.\Upsilon M) \\
\\
\Phi(x) & = & x \\
\Phi(\lambda x.M) & = & \lambda x.\Phi M \\
\Phi(V(W, x.M)) & = & \mathsf{let}\, x := \Phi V \Phi W \,\mathsf{in}\, \Phi M
\end{array}$$

Figure 13 Translation from $\textsc{ces}$ to $\textsc{cnf}$ and vice-versa.

The announced isomorphisms are given in Figs. 12 and 13. The map $\Psi : VES \to VFS$ requires the key auxiliary map Ψ_x, whose design is guided by types: if $\Gamma, x : A \vdash_{\mathsf{C}} c_x : B$ then $\Gamma|A \Rightarrow \Psi_x(c_x) : B$. The isomorphism $\Upsilon : \textsc{ces} \to \textsc{cnf}$ should be obvious. It can be proved that the operation $\mathsf{LET}\, y := M \,\mathsf{in}\, P$ in $\textsc{ces}$ is translated as left substitution: $\Upsilon(\mathsf{LET}\, y := M \,\mathsf{in}\, P) = [\Upsilon M\backslash y]\Upsilon P$.

A final point. The sub-kernel VES is isomorphic to the CPS-target, after composition with the negative translation: $VES \cong VFS \cong CPS$. A variant of the negative translation delivers:

▶ **Theorem 5.** $\textsc{cnf} \cong \textsc{cps}$.

So we also have $\textsc{ces} \cong \textsc{cnf} \cong \textsc{cps}$. Here $\textsc{cps}$ is the sub-calculus of CPS where commands KV are omitted and σ_v normalization is enforced. Its unique reduction rule, named β_v, becomes

$$(\beta_v) \qquad\qquad (\lambda y.\lambda k.M)W(\lambda x.N) \to [\lambda x.N/k][W/y]M$$

The definition of substitution $[\lambda x.N/k]M$ has the following critical clause:

$$[\lambda x.N/k](kV) = [V/x]N$$

This clause does the reduction of the σ_v-redex $(\lambda x.N)V$ on the fly; and it echoes the critical clause of a structural substitution. Moreover, $\textit{CPS}$ is the target of a version of the CPS-translation, obtained by changing just one clause: $(V : \lambda x.M) = [V^\dagger/x]\overline{M}$.

The variant of the negative translation yielding $\textit{CNF} \cong \textit{CPS}$ is defined by

$$(V(W, x.M))^\wr = V^\sim W^\sim(\lambda x.M^\wr)$$

All the other needed clauses as before. For the isomorphism, we have to prove:

$$([N\backslash x]M)^\wr = [\lambda x.M^\wr/k]N^\wr$$

This is a last minute bonus: a $\textit{CPS}$ explanation of left substitution.

6 Conclusions

Contributions. We list our main contribution: the VFS-translation; the negative translation as an isomorphism between the VFS and CPS targets; the decomposition of the CPS-translation in terms of the VFS-translation and the negative translation; the two sub-kernels of λC and their perfect relationship with appropriate fragments of the sequent calculus LJQ and natural deduction with general eliminations; the reworking of the term calculus for LJQ.

In all, we took the polished account of the essence of CPS, obtained in [25] and illustrated in Fig. 1, and revealed a rich proof-theoretical background, as in Fig. 2, with a double layer of sub-kernels, under a layer of expansions (see the dotted lines in Fig. 2 and recall (1), (2), and (3)), intersecting an intermediate zone, between the source language and the CPS targets, of calculi corresponding to proof systems.

Related work. In [4], LJQ is studied as a source language, while the CPS translation of LJQ is a tool to establish indirectly a connection with λC, through their respective kernels, in order to confirm that cut-elimination in LJQ is connected with call-by-value computation. There is nothing wrong with using the sequent calculus as source language and translating it with CPS: this has been done abundantly, even by the first author [1, 27, 4, 9]. But the point made here is that the sequent calculus should also be used as a tool to analyze the CPS-translation, and is able to play a special role as an intermediate language.

The sequent calculus was put forward as an intermediate representation for compilation of functional programs in [2]. This study addresses compilation of programs for a real-world language; designs an intermediate language *Sequent Core* (SC) inspired in the sequent calculus for such source language; and compares SC with CPS heuristically w. r. t. several desirable properties in the context of optimized compilation. In the present paper, we address the foundations of compilation, employing theoretical languages; pick the sequent calculus LJQ, which is a standard systems with decades of history in proof-theory [3]; and compare LJQ and CPS, not through a benchmarking of competing languages, but through mathematical results showing their intimate connection.

Future work. We know an appropriate CPS target will be capable of interpreting a classical extension of our chosen source language. The problem in moving in this direction is that there is no standard extension of λC with control operators readily available. Source languages with let-expressions and control operators can be found in [14, 5], but adopting them means to redo all that we have done here – that is another project. On the other hand, maybe a system with generalized applications will make a good source language. The system $\lambda \mathsf{J}_v$ performed well in this paper, since its sub-kernel of administrative normal forms ($\textit{CNF}$) is reachable without consideration of expansions – a sign of a well calibrated syntax.

References

1 Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000*, SIGPLAN Notices 35(9), pages 233–243. ACM, 2000. doi:10.1145/351240.351262.

2 Paul Downen, Luke Maurer, Zena M. Ariola, and Simon Peyton Jones. Sequent calculus as a compiler intermediate language. In Jacques Garrigue, Gabriele Keller, and Eijiro Sumii, editors, *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*, pages 74–88. ACM, 2016.

3 Roy Dyckhoff and Stéphane Lengrand. LJQ, a strongly focused calculus for intuitionistic logic. In A. Beckmann, U. Berger, B. Löwe, and J. V Tucker, editors, *Proc. of the 2nd Conference on Computability in Europe (CiE'06)*, volume 3988 of *Lecture Notes in Computer Science*. Springer-Verlag, 2006.

4 Roy Dyckhoff and Stéphane Lengrand. Call-by-value lambda calculus and LJQ. *Journal of Logic and Computation*, 17:1109–1134, 2007.

5 José Espírito Santo. Towards a canonical classical natural deduction system. *Annals of Pure and Applied Logic*, 164(6):618–650, 2013.

6 José Espírito Santo. The call-by-value lambda-calculus with generalized applications. In Maribel Fernández and Anca Muscholl, editors, *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*, volume 152 of *LIPIcs*, pages 35:1–35:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

7 José Espírito Santo and Filipa Mendes. The logical essence of compiling with continuations. *CoRR*, 2023. URL: https://arxiv.org/2304.14752.

8 José Espírito Santo. The λ-calculus and the unity of structural proof theory. *Theory of Computing Systems*, 45:963–994, 2009.

9 José Espírito Santo, Ralph Matthes, and Luís Pinto. Continuation-passing-style and strong normalization for intuitionistic sequent calculi. *Logical Methods in Computer Science*, 5(2:11), 2009.

10 Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. The essence of compiling with continuations. In Robert Cartwright, editor, *Proceedings of the ACM SIGPLAN'93 Conference on Programming Language Design and Implementation (PLDI), Albuquerque, New Mexico, USA, June 23-25, 1993*, pages 237–247. ACM, 1993.

11 Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. The essence of compiling with continuations (with retrospective). In Kathryn S. McKinley, editor, *20 Years of the ACM SIGPLAN Conference on Programming Language Design and Implementation 1979-1999, A Selection*, pages 502–514. ACM, 2003.

12 Timothy G. Griffin. A formulae-as-types notion of control. In *ACM Conf. Principles of Programming Languages*. ACM Press, 1990.

13 Hugo Herbelin. A λ-calculus structure isomorphic to a Gentzen-style sequent calculus structure. In L. Pacholski and J. Tiuryn, editors, *Proceedings of CSL'94*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer-Verlag, 1995.

14 Hugo Herbelin and Stéphane Zimmermann. An operational account of call-by-value minimal and classical lambda-calculus in "natural deduction" form. In *Proceedings of Typed Lambda Calculi and Applications'09*, volume 5608 of *LNCS*, pages 142–156. Springer-Verlag, 2009.

15 Felix Joachimski and Ralph 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.

16 Andrew Kennedy. Compiling with continuations, continued. In Ralf Hinze and Norman Ramsey, editors, *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming, ICFP 2007, Freiburg, Germany, October 1-3, 2007*, pages 177–190. ACM, 2007.

17 John Maraist, Martin Odersky, David N. Turner, and Philip Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Theoretical Computer Science*, 228(1-2):175–210, 1999.

18 Luke Maurer, Paul Downen, Zena M. Ariola, and Simon Peyton Jones. Compiling without continuations. In Albert Cohen and Martin T. Vechev, editors, *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*, pages 482–494. ACM, 2017.

19 Albert R. Meyer and Mitchell Wand. Continuation semantics in typed lambda-calculi (summary). In Rohit Parikh, editor, *Logics of Programs, Conference, Brooklyn College, New York, NY, USA, June 17-19, 1985, Proceedings*, volume 193 of *Lecture Notes in Computer Science*, pages 219–224. Springer, 1985.

20 Eugenio Moggi. Computational lambda-calculus and monads. Technical Report ECS-LFCS-88-86, University of Edinburgh, 1988.

21 Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.

22 M. Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classic natural deduction. In *Int. Conf. Logic Prog. Automated Reasoning*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer Verlag, 1992.

23 Gordon Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

24 Amr Sabry and Matthias Felleisen. Reasoning about programms in continuation-passing-style. *LISP and Symbolic Computation*, 6(3/4):289–360, 1993.

25 Amr Sabry and Philip Wadler. A reflection on call-by-value. *ACM Trans. on Programming Languages and Systems*, 19(6):916–941, 1997.

26 Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry/Howard Isomorphism*, volume 149 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2006.

27 Philip Wadler. Call-by-value is dual to call-by-name. In Colin Runciman and Olin Shivers, editors, *Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming, ICFP 2003, Uppsala, Sweden, August 25-29, 2003*, pages 189–201. ACM, 2003.

A The original LJQ system

The original calculus by Dyckhoff-Lengrand is recalled in Fig. 14.

B Kernel of λC

Our presentation of the kernel of $\lambda\mathsf{C}$ given in Fig. 4 is very close to the original one in [25], as we now see. In [25], the terms M of the kernel are generated by the grammar:

$$\begin{array}{rcl} M, N, P & ::= & \mathbb{K}[V] | \mathbb{K}[VW] \\ V, W & ::= & x | \lambda x.M \\ \mathbb{K} & ::= & [\_] | \mathsf{let}\, x := [\_]\, \mathsf{in}\, P \end{array}$$

We take for granted the sets of terms and values of $\lambda\mathsf{C}$, together with the set of contexts of $\lambda\mathsf{C}$, which are $\lambda\mathsf{C}$-terms with a single hole, and the concept of hole filling in such contexts. This grammar defines simultaneously a subset of the terms of $\lambda\mathsf{C}$, a subset of the values of $\lambda\mathsf{C}$, and a subset of the contexts of $\lambda\mathsf{C}$.

The second production in the grammar of terms, $\mathbb{K}[VW]$, should be understood thus: given in the kernel values V, W and a context $\mathbb{K}$, the $\lambda\mathsf{C}$-term $\mathbb{K}[VW]$, obtained by filling the hole of $\mathbb{K}$ with the $\lambda\mathsf{C}$-term VW, is in the kernel. In $\lambda\mathsf{C}$, VW is a subterm of $\mathbb{K}[VW]$; but, as we observed in Section 2, in the kernel, the term VW is not an immediate subterm of $\mathbb{K}[VW]$ – the immediate subexpressions are just V, W, and $\mathbb{K}$. Notice the $\lambda\mathsf{C}$-term $M = VW$ is a term in the kernel, generated by the second production of the grammar with $\mathbb{K} = [\_]$. But that second production *should not* be interpreted as $\mathbb{K}[M]$ with $M = VW$.

There is no primitive $\mathbb{K}[M]$ in the kernel. Instead, there is the operation $(M : \mathbb{K})$, defined by recursion on M as follows:

$$\begin{array}{rcl} (V : \mathbb{K}) & = & \mathbb{K}[V] \\ (VW : \mathbb{K}) & = & \mathbb{K}[VW] \\ (\mathsf{let}\, x := V \,\mathsf{in}\, M : \mathbb{K}) & = & \mathsf{let}\, x := V \,\mathsf{in}\, (M : \mathbb{K}) \\ (\mathsf{let}\, x := VW \,\mathsf{in}\, M : \mathbb{K}) & = & \mathsf{let}\, x := VW \,\mathsf{in}\, (M : \mathbb{K}) \end{array}$$

It is easy to see that $(M : \mathsf{let}\, x := [\_] \,\mathsf{in}\, P) = \mathsf{LET}\, x := M \,\mathsf{in}\, P$ and that $(M : [\_]) = M$.

In [25], the kernel has the following reduction rule

$$(\beta.v) \qquad \mathbb{K}[(\lambda x.M)V] \to ([V/x]M : \mathbb{K}) \ .$$

There is no need for the requirement of maximal $\mathbb{K}$ in this rule, as done in [25], once the above clarification about $\mathbb{K}[VW]$ is obtained. We now see the relationship between $\beta.v$ and our B_v and B'_v.

Let $\mathbb{K} = \mathsf{let}\, y := [\_] \,\mathsf{in}\, P$. Then rule B_v can re written as

$$\mathbb{K}[(\lambda x.M)V] \to \mathsf{let}\, x := V \,\mathsf{in}\, (M : \mathbb{K}) \ .$$

The contractum is a let_v-redex, which could be immediately reduced, to achieve the effect of $\beta.v$. Here we prefer to delay this let_v-step, and the same applies to our rule B'_v, which corresponds to the case $\mathbb{K} = [\_]$. This issue of delaying let_v is also seen in Section 5.

Finally, rule η_{let} in [25] reads $\mathsf{let}\, x := [\_] \,\mathsf{in}\, \mathbb{K}[x] \to \mathbb{K}$. We argue that in our presentation we can derive

$$(M : \mathsf{let}\, x := [\_] \,\mathsf{in}\, \mathbb{K}[x]) \to (M : \mathbb{K}) \ .$$

If $\mathbb{K} = [\_]$, then we have to prove $\mathsf{LET}\, x := M \,\mathsf{in}\, x \to M$. This is proved by an easy induction on M: the case $M = V$ (resp. $M = VW$) gives rise to a σ_v-step (resp. η_{let}-step); the remaining two cases follow by induction hypothesis.

If $\mathbb{K} = \mathsf{let}\, y := [\_] \,\mathsf{in}\, P$, then we have to prove $\mathsf{LET}\, x := M \,\mathsf{in}\, \mathsf{let}\, y := x \,\mathsf{in}\, P \to \mathsf{LET}\, y := M \,\mathsf{in}\, P$. Now $\mathsf{let}\, y := x \,\mathsf{in}\, P \to_{let_v} [y/x]P$. Since $Q \to Q'$ implies $\mathsf{LET}\, x := M \,\mathsf{in}\, Q \to \mathsf{LET}\, x := M \,\mathsf{in}\, Q'$, we obtain $\mathsf{LET}\, x := M \,\mathsf{in}\, \mathsf{let}\, y := x \,\mathsf{in}\, P \to \mathsf{LET}\, x := M \,\mathsf{in}\, [y/x]P =_\alpha \mathsf{LET}\, y := M \,\mathsf{in}\, P$.

$$
\begin{array}{llcl}
\text{(terms)} & M, N & ::= & \uparrow V \mid x(V, y.N) \mid \mathsf{C}_2(V, x.N) \mid \mathsf{C}_3(M, x.N) \\
\text{(values)} & V, W & ::= & x \mid \lambda x.M \mid \mathsf{C}_1(V, x.W)
\end{array}
$$

$$
\begin{array}{lrcll}
(1) & \mathsf{C}_3(\uparrow(\lambda x.M), y.y(V, z.N)) & \to & \mathsf{C}_3(\mathsf{C}_3(\uparrow V, x.M), z.N) & (a) \\
(2) & \mathsf{C}_3(\uparrow x, y.N) & \to & [x/y]N & \\
(3) & \mathsf{C}_3(M, x.\uparrow x) & \to & M & \\
(4) & \mathsf{C}_3(z(V, y.P), x.N) & \to & z(V, y.\mathsf{C}_3(P, x.N)) & \\
(5) & \mathsf{C}_3(\mathsf{C}_3(\uparrow W, y.y(V, z.P)), x.N) & \to & \mathsf{C}_3(\uparrow W, y.y(V, z.\mathsf{C}_3(P, x.N))) & (b) \\
(6) & \mathsf{C}_3(\mathsf{C}_3(M, y.P), x.N) & \to & \mathsf{C}_3(M, y.\mathsf{C}_3(P, x.N)) & (c) \\
(7) & \mathsf{C}_3(\uparrow(\lambda x.M), y.N) & \to & \mathsf{C}_2(\lambda x.M, y.N) & (d) \\
(8) & \mathsf{C}_1(V, x.x) & \to & V & \\
(9) & \mathsf{C}_1(V, x.y) & \to & y & (e) \\
(10) & \mathsf{C}_1(V, x.(\lambda y.M)) & \to & \lambda y.\mathsf{C}_2(V, x.M) & \\
(11) & \mathsf{C}_2(V, x.\uparrow W) & \to & \uparrow(\mathsf{C}_1(V, x.W)) & \\
(12) & \mathsf{C}_2(V, x.x(W, z.N)) & \to & \mathsf{C}_2(\uparrow V, x.x(\mathsf{C}_1(V, x.W), z.\mathsf{C}_2(V, x.N))) & \\
(13) & \mathsf{C}_2(V, x.y(W, z.N)) & \to & y(\mathsf{C}_1(V, x.W), z.\mathsf{C}_2(V, x.N)) & (e) \\
(14) & \mathsf{C}_2(V, x.\mathsf{C}_3(M, y.N)) & \to & \mathsf{C}_3(\mathsf{C}_2(V, x.M), y.\mathsf{C}_2(V, x.N)) &
\end{array}
$$

Provisos: (a) $y \notin FV(V) \cup FV(N)$. (b) $y \notin FV(V) \cup FV(P)$). (c) If rule (5) does not apply. (d) If rule (1) does not apply. (e) $x \neq y$.

$$
\frac{}{\Gamma, x : A \to x : A}\, Ax
\qquad\qquad
\frac{\Gamma \to V : A}{\Gamma \Rightarrow \uparrow V : A}\, Der
$$

$$
\frac{\Gamma, x : A \Rightarrow M : B}{\Gamma \to \lambda x.M : A \supset B}\, R\supset
\qquad\qquad
\frac{\Gamma \Rightarrow M : A \quad \Gamma, x : A \Rightarrow N : B}{\Gamma \Rightarrow \mathsf{C}_3(M, x.N) : B}\, Cut_3
$$

$$
\frac{\Gamma \to V : A \quad \Gamma, x : A \to W : B}{\Gamma \to \mathsf{C}_1(V, x.W) : B}\, Cut_1
\qquad
\frac{\Gamma \to V : A \quad \Gamma, x : A \Rightarrow N : B}{\Gamma \Rightarrow \mathsf{C}_2(V, x.N) : B}\, Cut_2
$$

$$
\frac{\Gamma, x : A \supset B \to V : A \quad \Gamma, x : A \supset B, y : B \Rightarrow N : C}{\Gamma, x : A \supset B \Rightarrow x(V, y.N) : C}\, L\supset
$$

Figure 14 The original calculus by Dyckhoff-Lengrand.

On the Lattice of Program Metrics

Ugo Dal Lago
University of Bologna, Italy
INRIA Sophia Antipolis, France

Naohiko Hoshino
Sojo University, Japan

Paolo Pistone
University of Bologna, Italy

Abstract

In this paper we are concerned with understanding the nature of program metrics for calculi with higher-order types, seen as natural generalizations of program equivalences. Some of the metrics we are interested in are well-known, such as those based on the interpretation of terms in metric spaces and those obtained by generalizing observational equivalence. We also introduce a new one, called the interactive metric, built by applying the well-known Int-Construction to the category of metric complete partial orders. Our aim is then to understand how these metrics relate to each other, i.e., whether and in which cases one such metric refines another, in analogy with corresponding well-studied problems about program equivalences. The results we obtain are twofold. We first show that the metrics of semantic origin, i.e., the denotational and interactive ones, lie *in between* the observational and equational metrics and that in some cases, these inclusions are strict. Then, we give a result about the relationship between the denotational and interactive metrics, revealing that the former is less discriminating than the latter. All our results are given for a linear lambda-calculus, and some of them can be generalized to calculi with graded comonads, in the style of Fuzz.

2012 ACM Subject Classification Theory of computation → Program semantics

Keywords and phrases Metrics, Lambda Calculus, Linear Types

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.20

Related Version *Extended Version*: `https://arxiv.org/abs/2302.05022`

Funding The first and third authors are partially supported by the ERC CoG DIAPASoN, GA 818616. The second author is supported by JSPS KAKENHI Grant Number JP21K11762.

1 Introduction

Program equivalence is one of the most important concepts in the semantics of programming languages: every way of giving semantics to programs induces a notion of equivalence, and the various notions of equivalence available for the same language, even when very different from each other, help us understanding the deep nature of the language itself. Indeed, there is not *one* single, preferred way to construct a notion of equivalence for programs. The latter is especially true in presence of higher-order types or in scenarios in which programs have a fundamentally interactive behavior, e.g. in process algebras. For example, the relationship between observational equivalence, the most coarse-grained congruence relation among those which are coherent with the underlying notion of observation, and denotational semantics has led in some cases to so-called full-abstraction results (e.g. [20, 13]), which are known to hold only for *some* denotational models and in *some* programming languages. A similar argument applies to applicative bisimilarity, which, e.g., is indeed fully abstract in presence of *probabilistic* effects [5, 8] but not so in presence of *nondeterministic* effects [23].

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 20; pp. 20:1–20:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$$\begin{tikzcd}
& d^{\mathrm{obs}} = d^{\mathrm{log}} & \\
d^{\mathrm{den}} \arrow[ur] & & d^{\mathrm{int}} \arrow[ul, thick] \arrow[ll, thick] \\
& d^{\mathrm{equ}} \arrow[ul, thick] \arrow[ur] &
\end{tikzcd}$$

Figure 1 Illustration of our comparison results for program metrics: an arrow $d^{\mathbf{a}} \to d^{\mathbf{b}}$ indicates that $d^{\mathbf{b}}$ is *coarser* (i.e. less discriminating) than $d^{\mathbf{a}}$. Thick arrows indicate *strict* domination.

Equivalences, although central to the theory of programming languages, do not allow us to say anything about all those pairs of programs which, while qualitatively exhibiting different behaviors, behave *similarly* in a quantitative sense. This has led to the study of notions of *distance* between programs, which often take the form of (pseudo-)metrics on the space of programs or of their denotations. In this sense we can distinguish at least three defining styles:

- First, observational equivalence can be generalized to a metric, maintaining the intrinsic quantification across all contexts, but observing a difference rather than an equality [6, 7].
- There is also an approach obtained by generalizing equational logic, recently introduced by Mardare et al. [25], which has been adapted to higher-order computations with both linear [9] and non-linear [11] types.
- Finally, linear calculi admit a denotational interpretation in the category of metric complete partial orders [2], and this is well-known to work well in presence of graded comonads.

In other words, various definitional styles for program equivalences for higher-order calculi have been proved to have a meaningful metric counterpart, at least when the underlying type system is based on linear or graded types. Actually, metric semantics for non-linear [15, 28] as well as effectful [14, 10] higher-order calculi have also been recently explored. However, there is a missing tale in this picture, namely the one provided by interactive semantic models akin to game semantics and the geometry of interaction [17], which were key ingredients towards the aforementioned full-abstraction results. Moreover, the relationship between the various notions of distance in the literature has been studied only superficially, and the overall situation is currently less clear than for program equivalences.

The aim of this work is to shed light on the landscape about metrics in higher-order programs. Notably, a new metric between programs inspired by Girard's geometry of interaction [17] is defined, being obtained by applying the so-called Int-construction [21, 1] to the category of metric complete partial orders. The result is a denotational model, which, while fundamentally different from existing metric models, provides a natural way to measure the distance between programs, which we call the *interactive metric*. In the interactive metric, differences between two programs can be observed incrementally, by interacting with the underlying denotational interpretation in the question-answer protocol typical of game semantics and the geometry of interaction.

Technically, the main contribution of the paper is an in-depth study of the relationships between the various metrics existing in the literature, including the interactive metric. Overall, the result of this analysis is the one in Figure 1. The observational metric remains the least discriminating, while the equational metric is proved to be the one assigning the greatest distances to (pairs of) programs. The two metrics of a semantic nature, namely the denotational one and the interactive one, stand in between the two metrics mentioned above, with the interactive metric being more discriminating than the denotational one.

The remainder of this manuscript is structured as follows. After recalling some basic facts about metric spaces in Section 2, in Section 3 we introduce a basic linear programming language over the reals and its associated notion of program metrics; in Section 4, we discuss the logical relation metric and the observational metric; in Section 5, we discuss the equational metric; in Section 6, we introduce the two denotational metrics; Sections 7 and 8 contain our main comparison results, and in Section 9, we shortly discuss the case of graded exponentials. Some proofs are omitted and can be found in an extended version of this paper [12].

2 Preliminaries

In this section, we recall the notions of extended pseudo-metric spaces and non-expansive functions. Let $\mathbb{R}^{\infty}_{\geq 0}$ be the set $\{a \in \mathbb{R} \mid a \geq 0\} \cup \{\infty\}$ of non-negative real numbers and infinity. An *extended pseudo-metric space* X consists of a set $|X|$ and a function $d_X : |X| \times |X| \to \mathbb{R}^{\infty}_{\geq 0}$ satisfying the following conditions:

- For all $x \in |X|$, we have $d_X(x, x) = 0$;
- For all $x, y \in |X|$, we have $d_X(x, y) = d_X(y, x)$;
- For all $x, y, z \in |X|$, we have $d_X(x, z) \leq d_X(x, y) + d_X(y, z)$.

In the sequel, we simply refer to extended pseudo-metric spaces as metric spaces, and we denote the underlying set $|X|$ by X.

For metric spaces X and Y, a function $f\colon X \to Y$ is said to be *non-expansive* when for all $x, y \in X$, we have $d_Y(fx, fy) \leq d_X(x, y)$. We write **Met** for the category of metric spaces and non-expansive functions. The category **Met** has a symmetric monoidal closed structure $(1, \otimes, \multimap)$ where the metric of the tensor product $X \otimes Y$ is given by

$$d_{X\otimes Y}((x, y), (z, w)) = d_X(x, z) + d_Y(y, w).$$

We suppose that the monoidal product is left associative, and we denote the n-fold monoidal product of X by $X^{\otimes n}$. In the sequel, $\mathbb{R}$ denotes the metric space of real numbers equipped with the absolute distance $d_{\mathbb{R}}(a, b) = |a - b|$.

3 A Linear Programming Language

3.1 Syntax and Operational Semantics

We introduce our target language that is a linear lambda calculus equipped with constant symbols for real numbers and non-expansive functions. We fix a set S of non-expansive functions $f\colon \mathbb{R}^{\otimes n} \to \mathbb{R}$ with $n \geq 1$. We call n the *arity* of f. For example, S may include addition $+\colon \mathbb{R} \otimes \mathbb{R} \to \mathbb{R}$ and trigonometric functions such as $\sin, \cos\colon \mathbb{R} \to \mathbb{R}$. We assume function symbols $\overline{f}$ for $f \in S$ and constant symbols $\overline{a}$ for real numbers $a \in \mathbb{R}$.

Our language, denoted by Λ_S, is given as follows. Types and environments are given by

$$\text{Types} \quad \tau, \sigma := \mathbf{R} \mid \mathbf{I} \mid \tau \multimap \sigma \mid \tau \otimes \sigma, \qquad \text{Environments} \quad \Gamma, \Delta := \varnothing \mid \Gamma, x : \tau.$$

We denote the set of types by **Ty** and denote the set of environments by **Env**. We always suppose that every variable appears at most once in any environment. For environments Γ and Δ that do not share any variable, we write $\Gamma \# \Delta$ for a *merge* [3, 18] of Γ and Δ, that is an environment obtained by shuffling variables in Γ and Δ preserving the order of variables in Γ and the order of variables in Δ. For example, $(x : \tau, y : \sigma, y' : \sigma', x' : \tau')$ is a merge of $(x : \tau, x' : \tau')$ and $(y : \sigma, y' : \sigma')$. When we write $\Gamma \# \Delta$, we implicitly suppose that no variable is shared by Γ and Δ. Terms, values and contexts are given by the following BNF.

$$\frac{}{x:\tau\vdash x:\tau} \qquad \frac{a\in\mathbb{R}}{\vdash \overline{a}:\mathbf{R}} \qquad \frac{}{\vdash *:\mathbf{I}} \qquad \frac{f\in S \quad \Gamma_1\vdash M_1:\mathbf{R} \quad \dots \quad \Gamma_{\mathrm{ar}(f)}\vdash M_{\mathrm{ar}(f)}:\mathbf{R}}{\Gamma_1\#\cdots\#\Gamma_{\mathrm{ar}(f)}\vdash \overline{f}(M_1,\dots,M_{\mathrm{ar}(f)}):\mathbf{R}}$$

$$\frac{\Gamma,x:\sigma\vdash M:\tau}{\Gamma\vdash\lambda x:\sigma.\,M:\sigma\multimap\tau} \qquad \frac{\Gamma\vdash M:\sigma\multimap\tau \quad \Delta\vdash N:\sigma}{\Gamma\#\Delta\vdash M\,N:\tau} \qquad \frac{\Gamma\vdash M:\tau \quad \Delta\vdash N:\sigma}{\Gamma\#\Delta\vdash M\otimes N:\tau\otimes\sigma}$$

$$\frac{\Gamma\vdash M:\mathbf{I} \quad \Delta\vdash N:\tau}{\Gamma\#\Delta\vdash \mathbf{let}\ *\ \mathbf{be}\ M\ \mathbf{in}\ N:\tau} \qquad \frac{\Gamma\vdash M:\sigma_1\otimes\sigma_2 \quad \Delta,x:\sigma_1,y:\sigma_2\vdash N:\tau}{\Gamma\#\Delta\vdash \mathbf{let}\ x\otimes y\ \mathbf{be}\ M\ \mathbf{in}\ N:\tau}$$

Figure 2 Typing Rules.

$$\frac{}{V\hookrightarrow V} \qquad \frac{M_1\hookrightarrow\overline{a_1} \quad \dots \quad M_n\hookrightarrow\overline{a_n}}{\overline{f}(M_1,\dots,M_n)\hookrightarrow\overline{f(a_1,\dots,a_n)}} \qquad \frac{M\hookrightarrow\lambda x:\tau.\,L \quad N\hookrightarrow V \quad L[V/x]\hookrightarrow U}{M\,N\hookrightarrow U}$$

$$\frac{M\hookrightarrow V \quad N\hookrightarrow U}{M\otimes N\hookrightarrow V\otimes U} \qquad \frac{M\hookrightarrow * \quad N\hookrightarrow V}{\mathbf{let}\ *\ \mathbf{be}\ M\ \mathbf{in}\ N\hookrightarrow V} \qquad \frac{M\hookrightarrow V\otimes U \quad N[V/x,U/y]\hookrightarrow W}{\mathbf{let}\ x\otimes y\ \mathbf{be}\ M\ \mathbf{in}\ N\hookrightarrow W}$$

Figure 3 Evaluation Rules.

$$\begin{array}{ll}
\text{Terms} & M,N := x\in\mathbf{Var}\mid\overline{a}\mid *\mid\overline{f}(M_1,\dots,M_{\mathrm{ar}(f)})\mid M\,N\mid\lambda x:\tau.\,M\mid \\
& \qquad M\otimes N\mid\mathbf{let}\ *\ \mathbf{be}\ M\ \mathbf{in}\ N\mid\mathbf{let}\ x\otimes y\ \mathbf{be}\ M\ \mathbf{in}\ N \\
\text{Values} & V,U := \overline{a}\mid *\mid\lambda x:\tau.\,M\mid V\otimes U \\
\text{Contexts} & C[-] := [-]\mid\overline{f}(M,\dots,M',C[-],N',\dots,N)\mid C[-]\,M\mid M\,C[-]\mid\lambda x:\tau.\,C[-]\mid \\
& \qquad C[-]\otimes M\mid M\otimes C[-]\mid\mathbf{let}\ *\ \mathbf{be}\ C[-]\ \mathbf{in}\ M\mid\mathbf{let}\ *\ \mathbf{be}\ M\ \mathbf{in}\ C[-]\mid \\
& \qquad \mathbf{let}\ x\otimes y\ \mathbf{be}\ C[-]\ \mathbf{in}\ M\mid\mathbf{let}\ x\otimes y\ \mathbf{be}\ M\ \mathbf{in}\ C[-]
\end{array}$$

Here, a ranges over $\mathbb{R}$, f ranges over S, and x ranges over a countably infinite set $\mathbf{Var}$ of variables. We write $\Gamma\vdash M:\tau$ when the typing judgement is derived from the rules given in Figure 2. Evaluation rules are given in Figure 3. Since Λ_S is a purely linear programming language, for any closed term $\vdash M:\tau$, there is a value $\vdash V:\tau$ such that $M\hookrightarrow V$. For an environment Γ and a type τ, we define $\mathbf{Term}(\Gamma,\tau)$ to be the set of all terms M such that $\Gamma\vdash M:\tau$, and we define $\mathbf{Value}(\tau)$ to be the set of closed values of type τ. We simply write $\mathbf{Term}(\tau)$ for $\mathbf{Term}(\varnothing,\tau)$, that is the set of closed terms of type τ. For a context $C[-]$, we write $C[-]\colon(\Gamma,\tau)\to(\Delta,\sigma)$ when for all terms $\Gamma\vdash M:\tau$, we have $\Delta\vdash C[M]:\sigma$.

We adopt Church-style lambda abstraction so that every type judgement $\Gamma\vdash M:\tau$ has a unique derivation, which makes it easier to define denotational semantics for Λ_S. Except for this point, our language can be understood as a fragment of Fuzz [29] – the typing judgment $x:\sigma,\dots,y:\rho\vdash M:\tau$ corresponds to $x:_1\sigma,\dots,y:_1\rho\vdash M:\tau$ in Fuzz. In Section 9, we discuss extending our results in this paper to a richer language, closer to the one from [29].

3.2 Equational Theory

In this paper, we consider an equational theory for Λ_S, which will turn out to be instrumental to define a notion of well-behaving family of metrics for Λ_S called admissibility (Section 3.3) and to give a quantitative equational theory for Λ_S (Section 5). In both cases, if two terms are to be considered equal, then the distance between them is required to be 0. Here, we adopt the standard equational theory for the linear lambda calculus [24] extended with the following axiom

$$\frac{f\in S \quad f(a_1,\dots,a_{\mathrm{ar}(f)})=b}{\vdash\overline{f}(\overline{a_1},\dots,\overline{a_{\mathrm{ar}(f)}})=\overline{b}:\tau}\ .$$

For terms $\Gamma\vdash M:\tau$ and $\Gamma\vdash N:\tau$, we write $\Gamma\vdash M=N:\tau$ when the equality is derivable.

We may add some other axioms to the equational theory as long as the axioms are valid when we interpret function symbols $\overline{f}$ as f and constant symbols $\overline{a}$ as a. For example, when $\mathrm{add}\colon \mathbb{R} \otimes \mathbb{R} \to \mathbb{R}$ is in S, we may add the commutativity law $x : \mathbf{R}, y : \mathbf{R} \vdash \overline{\mathrm{add}}(x, y) = \overline{\mathrm{add}}(y, x) : \mathbf{R}$ to the equational theory. The rest of this paper is not affected by such extensions to the equational theory.

3.3 Admissibility

Let us call a family $\{d_{\Gamma,\tau}\}_{\Gamma\in\mathbf{Env},\tau\in\mathbf{Ty}}$ in which $d_{\Gamma,\tau}$ is a metric on $\mathbf{Term}(\Gamma, \tau)$ a *metric on* Λ_S. We introduce a class of metrics on Λ_S, which is the object of study of this paper.

▶ **Definition 1** (Admissible Metric). *Let $\{d_{\Gamma,\tau}\}_{\Gamma\in\mathbf{Env},\tau\in\mathbf{Ty}}$ be a metric on Λ_S. We say that $\{d_{\Gamma,\tau}\}_{\Gamma\in\mathbf{Env},\tau:\mathbf{Ty}}$ is* admissible *when the following conditions hold:*

(A1) *For any environment Γ, any type τ, any pair of terms $\Gamma \vdash M : \tau$, $\Gamma \vdash N : \tau$ and any context $C[-]\colon (\Gamma, \tau) \to (\Delta, \sigma)$, we have $d_{\Delta,\sigma}(C[M], C[N]) \le d_{\Gamma,\tau}(M, N)$.*

(A2) *For all $a, b \in \mathbb{R}$, we have $d_{\varnothing,\mathbf{R}}(a, b) = |a - b|$.*

(A3) *For all $a_1, \dots, a_n, b_1, \dots, b_n \in \mathbb{R}$ and all closed values $\vdash V : \tau$ and $\vdash U : \tau$, we have*

$$d_{\varnothing,\mathbf{R}^{\otimes n}\otimes\tau}\left(\overline{a_1} \otimes \cdots \otimes \overline{a_n} \otimes V, \overline{b_1} \otimes \cdots \otimes \overline{b_n} \otimes U\right) \ge |a_1 - b_1| + \cdots + |a_n - b_n|.$$

(A4) *If $\Gamma \vdash M = N : \tau$, then $d_{\Gamma,\tau}(M, N) = 0$.*

The first condition (A1) states that all contexts are non-expansive, and the second condition (A2) states that the metric on $\mathbf{R}$ coincides with the absolute metric on $\mathbb{R}$. (A3) states that the distance between two terms $\overline{a_1} \otimes \cdots \otimes \overline{a_n} \otimes V$ and $\overline{b_1} \otimes \cdots \otimes \overline{b_n} \otimes U$ is bounded (from below) by the distance between their "observable fragments" $d_{\mathbb{R}^{\otimes n}}((a_1, \dots, a_n), (b_1, \dots, b_n))$. The last condition (A4) states that $d_{\Gamma,\tau}$ subsumes the equational theory for Λ_S.

The definition of admissibility is motivated by the study of Fuzz [29], which is a linear type system for verifying differential privacy [4]. There, Reed and Pierce introduce a syntactically defined metric on Fuzz using a family of relations called metric relations, and they prove that all programs are non-expansive with respect to the syntactic metric (Theorem 6.4 in [29]). (A1) is motivated by this result. Furthermore, in the definition of the metric relation, the tensor product of types is interpreted as the monoidal product of metric spaces, and the type of real numbers is interpreted as $\mathbb{R}$ with the absolute distance. (A2) and (A3) are motivated by these definitions. In fact, given an admissible metric $\{d_{\Gamma,\tau}\}_{\Gamma\in\mathbf{Env},\tau\in\mathbf{Ty}}$ on Λ_S, we can show that $d_{\varnothing,\mathbf{R}^{\otimes n}}$ coincides with the metric of $\mathbb{R}^{\otimes n}$:

▶ **Lemma 2.** *If a metric $\{d_{\Gamma,\tau}\}_{\Gamma\in\mathbf{Env},\,\tau\in\mathbf{Ty}}$ is admissible, then for all $a_1, b_1, \dots, a_n, b_n \in \mathbb{R}$,*

$$d_{\varnothing,\mathbf{R}^{\otimes n}}(\overline{a_1} \otimes \cdots \otimes \overline{a_n}, \overline{b_1} \otimes \cdots \otimes \overline{b_n}) = |a_1 - b_1| + \cdots + |a_n - b_n|. \tag{1}$$

The reason that we do not take (1) as the third condition of admissibility and instead rely on the stronger condition (A3) above is that requiring (1) would not allow us to characterize the observational metric (Section 4.2) as the least admissible metric on Λ_S.

4 Logical Metric and Observational Metric

We give two syntactically defined metrics on Λ_S: one is based on logical relations, and the other is given in the style of Morris observational equivalence [27]. We then show that the two metrics coincide. This can be seen as a metric variant of Milner's context lemma [26].

4.1 Logical Metric

The first metric on Λ_S is given by means of a quantitative form of logical relations [29] called *metric logical relations*. Here, we directly define metric logical relations, and then, we define the induced metric on Λ_S. The metric logical relations

$$\{(-) \simeq_r^\tau (-) \subseteq \mathbf{Term}(\tau) \times \mathbf{Term}(\tau)\}_{\tau \in \mathbf{Ty},\, r \in \mathbb{R}_{\geq 0}^\infty}$$

are given by induction on τ as follows.

$$\begin{aligned}
M \simeq_r^{\mathbf{R}} N &\iff M \hookrightarrow \overline{a} \text{ and } N \hookrightarrow \overline{b} \text{ and } |a-b| \leq r\\
M \simeq_r^{\mathbf{I}} N &\iff M \hookrightarrow * \text{ and } N \hookrightarrow *\\
M \simeq_r^{\tau\otimes\sigma} N &\iff M \hookrightarrow V \otimes V' \text{ and } N \hookrightarrow U \otimes U' \text{ and}\\
&\qquad \exists s, s' \in \mathbb{R}_{\geq 0}^\infty,\ V \simeq_s^\tau U \text{ and } V' \simeq_{s'}^\sigma U' \text{ and } s + s' \leq r\\
M \simeq_r^{\tau\multimap\sigma} N &\iff M \hookrightarrow \lambda x : \tau.\, M' \text{ and } N \hookrightarrow \lambda x : \tau.\, N' \text{ and}\\
&\qquad \forall V, U \in \mathbf{Value}(\tau),\text{ if } V \simeq_s^\tau U,\ \text{ then } M'[V/x] \simeq_{r+s}^\sigma N'[U/x]
\end{aligned}$$

Then for an environment $\Gamma = (x : \sigma, \ldots, y : \rho)$ and a pair of terms $\Gamma \vdash M : \tau$ and $\Gamma \vdash N : \tau$, we define $d_{\Gamma,\tau}^{\log}(M, N) \in \mathbb{R}_{\geq 0}^\infty$ by

$$d_{\Gamma,\tau}^{\log}(M, N) = \inf\{r \in \mathbb{R}_{\geq 0}^\infty \mid \lambda x : \sigma. \cdots \lambda y : \rho.\, M \simeq_r^{\sigma\multimap\cdots\multimap\rho\multimap\tau} \lambda x : \sigma. \cdots \lambda y : \rho.\, N\}.$$

We give a consequence of our results in this paper, namely, Theorem 6 and Theorem 18. For the detail of the proof of Proposition 3, see [12].

▶ **Proposition 3.** *For any environment Γ and any type τ, the function $d_{\Gamma,\tau}^{\log}$ is a metric on $\mathbf{Term}(\Gamma, \tau)$. Furthermore, $\{d_{\Gamma,\tau}^{\log}\}_{\Gamma\in\mathbf{Env},\tau\in\mathbf{Ty}}$ is admissible.*

We call $d^{\log}$ *logical metric.* We note that we can directly check that $d^{\log}$ satisfies (A2) and (A3), and we need Theorem 6 and Theorem 18 to show that $d^{\log}$ is a metric on Λ_S and satisfies (A1) and (A4).

▶ **Example 4.** In this example, we suppose that the addition add: $\mathbb{R} \otimes \mathbb{R} \to \mathbb{R}$ is in S. Let M be $\lambda k : \mathbf{R} \multimap \mathbf{R}.\, k\,\overline{1}$, and let N be $\lambda k : \mathbf{R} \multimap \mathbf{R}.\, \overline{\mathrm{add}}(\overline{-2}, k\,\overline{0})$. Then

$$d_{\varnothing,\mathbf{R}\multimap\mathbf{R}}^{\log}(M, N) \leq 1 + 2 = 3$$

because we have $V\,\overline{1} \simeq_{s+1}^{\mathbf{R}} U\,\overline{0}$ for any pair $V \simeq_s^{\mathbf{R}\multimap\mathbf{R}} U$. In fact, we have $d_{\varnothing,\mathbf{R}\multimap\mathbf{R}}^{\log}(M, N) = 3$, which follows from Theorem 6. See Example 7. $\lrcorner$

▶ **Example 5.** For $a \in \mathbb{R}$, we define a term M_a to be

$$\vdash \overline{a} \otimes \overline{a} \otimes V : \mathbf{R} \otimes \mathbf{R} \otimes ((\mathbf{R} \otimes \mathbf{R} \multimap \mathbf{R}) \multimap \mathbf{R}) \qquad \text{where } V = \lambda k : \mathbf{R} \otimes \mathbf{R} \multimap \mathbf{R}.\, k\,(\overline{0} \otimes \overline{0}).$$

Since $d_{\varnothing,\mathbf{R}}^{\log}(\overline{0}, \overline{1}) = 1$, we obtain $d_{\varnothing,\mathbf{R}\otimes\mathbf{R}\otimes((\mathbf{R}\otimes\mathbf{R}\multimap\mathbf{R})\multimap\mathbf{R})}^{\log}(M_0, M_1) = 1 + 1 + 0 = 2.$ $\lrcorner$

4.2 Observational Metric

We next give a metric, which we call the *observational metric*, that measures distances between terms by observing concrete values produced by any possible context. For terms $\Gamma \vdash M : \tau$ and $\Gamma \vdash N : \tau$, we define $d_{\Gamma,\tau}^{\mathrm{obs}}(M, N) \in \mathbb{R}_{\geq 0}^\infty$ by

$$d_{\Gamma,\tau}^{\mathrm{obs}}(M, N) = \sup_{(n,\sigma,C[-])\in\mathcal{K}(\Gamma,\tau)} \left\{ |a_1 - b_1| + \cdots + |a_n - b_n| \;\middle|\; \begin{array}{l} C[M] \hookrightarrow \overline{a_1} \otimes \cdots \otimes \overline{a_n} \otimes V \\ \text{and } C[N] \hookrightarrow \overline{b_1} \otimes \cdots \otimes \overline{b_n} \otimes U \end{array} \right\}$$

where $(n, \sigma, C[-]) \in \mathcal{K}(\Gamma, \tau)$ if and only if $C[-]$ is a context from (Γ, τ) to $(\varnothing, \mathbf{R}^{\otimes n} \otimes \sigma)$.

$$\frac{\Gamma \vdash M = N : \tau}{\Gamma \vdash M \approx_0 N : \tau} \qquad \frac{\Gamma \vdash M \approx_r N : \tau}{\Gamma \vdash N \approx_r M : \tau} \qquad \frac{\Gamma \vdash M \approx_r N : \tau \quad \Gamma \vdash N \approx_s L : \tau}{\Gamma \vdash M \approx_{r+s} L : \tau}$$

$$\frac{|a - b| \leq r}{\vdash \overline{a} \approx_r \overline{b} : \mathbf{R}} \qquad \frac{\Gamma \vdash M \approx_r N : \tau \quad C[-] \colon (\Gamma, \tau) \to (\Delta, \sigma)}{\Delta \vdash C[M] \approx_r C[N] : \sigma}$$

Figure 4 Derivation Rules for $\Gamma \vdash M \approx_r N : \tau$.

▶ **Theorem 6.** *For any environment Γ and any type τ, we have $d^{\mathrm{obs}}_{\Gamma,\tau} = d^{\mathrm{log}}_{\Gamma,\tau}$.*

This theorem follows from coincidence of the metric logical relations with the metric relations [29] and the fundamental lemma for metric logical relations.

▶ **Example 7.** Let M and N be terms given in Example 4. By observing these terms by the context $[-]\,(\lambda x : \mathbf{R}.\, x)$, we see that $d^{\mathrm{log}}_{\varnothing,(\mathbf{R} \multimap \mathbf{R}) \multimap \mathbf{R}}(M, N) \geq 3$. By Theorem 6 and Example 4, we obtain $d^{\mathrm{log}}_{\varnothing,(\mathbf{R} \multimap \mathbf{R}) \multimap \mathbf{R}}(M, N) = 3$. ⌟

▶ **Example 8.** We consider the term $\vdash M_a : \tau$ given in Example 5 again. By observing M_0 and M_1 by the trivial context $[-]$, we can directly check that $d^{\mathrm{obs}}_{\varnothing,\mathbf{R}\otimes\mathbf{R}\otimes((\mathbf{R}\otimes\mathbf{R}\multimap\mathbf{R})\multimap\mathbf{R})}(M_0, M_1) \geq 2$. (In fact, it follows from Theorem 6 that the distance is equal to 2.) The purpose of the auxiliary type σ in the definition of $\mathcal{K}(\Gamma, \tau)$ is to enable observations of this type. In this case, while the logical metric distinguishes M_0 from M_1, we can not observationally distinguish M_0 from M_1 by means of observations at types $\mathbf{R}^{\otimes n}$ when S is empty. This is because there is no closed term of type $\mathbf{R} \otimes \mathbf{R} \multimap \mathbf{R}$ when S is empty. For a detailed explanation, see [12]. ⌟

5 Equational Metric

We give another syntactic metric on Λ_S, which we call the *equational metric*. This is essentially the quantitative equational theory from [9] without the rules called **weak**, **join** and **arch**. We exclude these rules since they do not affect the equational metric d^{equ} given below.

For terms $\Gamma \vdash M : \tau$ and $\Gamma \vdash N : \tau$, and for $r \in \mathbb{R}^{\infty}_{\geq 0}$, we write

$$\Gamma \vdash M \approx_r N : \tau$$

when we can derive the judgement from the rules in Figure 4. Then, for terms $\Gamma \vdash M : \tau$ and $\Gamma \vdash N : \tau$ we define $d^{\mathrm{equ}}_{\Gamma,\tau}(M, N) \in \mathbb{R}^{\infty}_{\geq 0}$ by

$$d^{\mathrm{equ}}_{\Gamma,\tau}(M, N) = \inf\{r \in \mathbb{R}^{\infty}_{\geq 0} \mid \Gamma \vdash M \approx_r N : \tau\}.$$

▶ **Proposition 9.** *For any environment Γ and any type τ, the function $d^{\mathrm{equ}}_{\Gamma,\tau}$ is a metric on* $\mathbf{Term}(\Gamma, \tau)$*. Furthermore, $\{d^{\mathrm{equ}}_{\Gamma,\tau}\}_{\Gamma \in \mathbf{Env}, \tau \in \mathbf{Ty}}$ is admissible.*

▶ **Example 10.** The equational metric measures differences between terms by comparing their subterms. For example, we have $\vdash \overline{2} \approx_1 \overline{3} : \mathbf{R}$, and therefore, $k : \mathbf{R} \multimap \mathbf{R} \vdash k\,\overline{2} \approx_1 k\,\overline{3} : \mathbf{R}$ holds. From this, we see that $d^{\mathrm{equ}}_{(k:\mathbf{R}\multimap\mathbf{R}),\mathbf{R}}(k\,\overline{2}, k\,\overline{3}) \leq 1$. In fact, this is an equality. This follows from $d^{\mathrm{obs}}_{(k:\mathbf{R}\multimap\mathbf{R}),\mathbf{R}}(k\,\overline{2}, k\,\overline{3}) \geq 1$, which is easy to check, and Theorem 18 below. ⌟

In general, we have $d^{\mathrm{obs}}_{\Gamma,\tau}(M, N) < d^{\mathrm{equ}}_{\Gamma,\tau}(M, N)$, i.e., the equational metric is strictly more discriminating than the observational metric (Theorem 18), which is proved by semantically inspired metrics in the next section.

6 Models of Λ_S and Associated Metrics

Now, we move our attention to semantically derived metrics on Λ_S. We first give a notion of models of Λ_S based on **Met**-enriched symmetric monoidal closed categories. **Met**-enriched symmetric monoidal closed categories are studied in [9] as models of quantitative equational theories for the linear lambda calculus. Then, we give two examples of semantic metrics on Λ_S: one is based on domain theory, and the other is based on the Geometry of Interaction.

6.1 Met-enriched Symmetric Monoidal Closed Category

We say that a symmetric monoidal closed category $(\mathcal{C}, I, \otimes, \multimap)$ is **Met**-*enriched* when every hom-set $\mathcal{C}(X, Y)$ has the structure of a metric space subject to the following conditions:

- the composition is a morphism in **Met** from $\mathcal{C}(X,Y) \otimes \mathcal{C}(Z,X)$ to $\mathcal{C}(Z,Y)$; and
- the tensor is a morphism in **Met** from $\mathcal{C}(X,Y) \otimes \mathcal{C}(Z,W)$ to $\mathcal{C}(X \otimes Z, Y \otimes W)$; and
- the currying operation is an isomorphism in **Met** from $\mathcal{C}(X \otimes Y, Z)$ to $\mathcal{C}(X, Y \multimap Z)$.

For morphisms $f, g \colon X \to Y$ in $\mathcal{C}$, we write $d(f,g)$ for the distance between f and g.

▶ **Definition 11.** *A* pre-model $\mathcal{M} = (\mathcal{C}, I, \otimes, \multimap, \lfloor - \rfloor)$ *of* Λ_S *is a* **Met**-*enriched symmetric monoidal closed category* $(\mathcal{C}, I, \otimes, \multimap)$ *equipped with an object* $\lfloor \mathbf{R} \rfloor \in \mathcal{C}$ *and families of morphisms* $\{\lfloor a \rfloor \colon I \to \lfloor \mathbf{R} \rfloor\}_{a \in \mathbb{R}}$ *and* $\{\lfloor f \rfloor \colon \lfloor \mathbf{R} \rfloor^{\otimes \mathrm{ar}(f)} \to \lfloor \mathbf{R} \rfloor\}_{f \in S}$.

For a pre-model $\mathcal{M} = (\mathcal{C}, I, \otimes, \multimap, \lfloor - \rfloor)$ of Λ_S, we interpret types as follows:

$$\llbracket \mathbf{R} \rrbracket^{\mathcal{M}} = \lfloor \mathbf{R} \rfloor, \quad \llbracket \mathbf{I} \rrbracket^{\mathcal{M}} = I, \quad \llbracket \tau \otimes \sigma \rrbracket^{\mathcal{M}} = \llbracket \tau \rrbracket^{\mathcal{M}} \otimes \llbracket \sigma \rrbracket^{\mathcal{M}}, \quad \llbracket \tau \multimap \sigma \rrbracket^{\mathcal{M}} = \llbracket \tau \rrbracket^{\mathcal{M}} \multimap \llbracket \sigma \rrbracket^{\mathcal{M}}.$$

For an environment $\Gamma = (x : \tau, \dots, y : \sigma)$, we define $\llbracket \Gamma \rrbracket^{\mathcal{M}}$ to be $\llbracket \tau \rrbracket^{\mathcal{M}} \otimes \cdots \otimes \llbracket \sigma \rrbracket^{\mathcal{M}}$. Then, the interpretation $\llbracket \Gamma \vdash M : \tau \rrbracket^{\mathcal{M}} \colon \llbracket \Gamma \rrbracket^{\mathcal{M}} \to \llbracket \tau \rrbracket^{\mathcal{M}}$ in $\mathcal{M}$ is given in the standard manner following [24], and constants are interpreted as follows: $\llbracket \vdash \overline{a} : \mathbf{R} \rrbracket^{\mathcal{M}} = \lfloor a \rfloor$,

$$\llbracket \Gamma \# \cdots \# \Delta \vdash \overline{f}(M, \dots, N) : \mathbf{R} \rrbracket^{\mathcal{M}} = \lfloor f \rfloor \circ (\llbracket M \rrbracket^{\mathcal{M}} \otimes \cdots \otimes \llbracket N \rrbracket^{\mathcal{M}}) \circ \theta$$

where $\theta \colon \llbracket \Gamma \# \Delta \rrbracket^{\mathcal{M}} \xrightarrow{\cong} \llbracket \Gamma \rrbracket^{\mathcal{M}} \otimes \llbracket \Delta \rrbracket^{\mathcal{M}}$ swaps objects following the merge $\Gamma \# \Delta$.

▶ **Definition 12.** *We say that a pre-model* $\mathcal{M} = (\mathcal{C}, I, \otimes, \multimap, \lfloor - \rfloor)$ *of* Λ_S *is a* model *of* Λ_S *if* $\mathcal{M}$ *satisfies the following three conditions.*

- *(M1) For any* $f \in S$, *if* $f(a_1, \dots, a_{\mathrm{ar}(f)}) = b$, *then* $\llbracket \overline{f}(\overline{a_1}, \dots, \overline{a_n}) \rrbracket^{\mathcal{M}} = \llbracket \overline{b} \rrbracket^{\mathcal{M}}$.
- *(M2) For all* $a, b \in \mathbb{R}$, $d(\lfloor a \rfloor, \lfloor b \rfloor) = |a - b|$.
- *(M3) For all* $x, y \colon I \to X$ *in* $\mathcal{C}$ *and all finite sequences* $a_1, \dots, a_n, b_1, \dots, b_n \in \mathbb{R}$, *we have*

$$d(\lfloor a_1 \rfloor \otimes \cdots \otimes \lfloor a_n \rfloor \otimes x, \lfloor b_1 \rfloor \otimes \cdots \otimes \lfloor b_n \rfloor \otimes y) \geq |a_1 - b_1| + \cdots + |a_n - b_n|.$$

The first condition corresponds to the reduction rule for function symbols and is necessary to prove soundness for models of Λ_S. The remaining conditions are for admissibility of the metric derived from models of Λ_S.

▶ **Proposition 13** (Soundness)**.** *Let* $\mathcal{M}$ *be a model of* Λ_S. *For any term* $M \in \mathbf{Term}(\tau)$ *and any value* $V \in \mathbf{Value}(\tau)$, *if* $M \hookrightarrow V$, *then* $\llbracket M \rrbracket^{\mathcal{M}} = \llbracket V \rrbracket^{\mathcal{M}}$.

Let $\mathcal{M} = (\mathcal{C}, I, \otimes, \multimap, \lfloor - \rfloor)$ be a model of Λ_S. For an environment Γ and a type τ, we define $d^{\mathcal{M}}_{\Gamma,\tau}$ to be the function $d(\llbracket - \rrbracket^{\mathcal{M}}, \llbracket - \rrbracket^{\mathcal{M}})$ from $\mathbf{Term}(\Gamma, \tau) \times \mathbf{Term}(\Gamma, \tau)$ to $\mathbb{R}^{\infty}_{\geq 0}$.

▶ **Proposition 14.** *For any environment* Γ *and any type* τ, *the function* $d^{\mathcal{M}}_{\Gamma,\tau}$ *is a metric on* $\mathbf{Term}(\Gamma, \tau)$. *Furthermore,* $\{d^{\mathcal{M}}_{\Gamma,\tau}\}_{\Gamma \in \mathbf{Env}, \tau \in \mathbf{Ty}}$ *is admissible.*

▶ **Example 15.** The symmetric monoidal closed category **Met** of metric spaces and non-expansive functions can be extended to a model $(\mathbf{Met}, I, \otimes, \multimap, \lfloor - \rfloor)$ of Λ_S where we define $\lfloor \mathbf{R} \rfloor \in \mathbf{Met}$ to be $\mathbb{R}$, and for $f \in S$, we define $\lfloor f \rfloor \colon \mathbb{R}^{\otimes \mathrm{ar}(f)} \to \mathbb{R}$ to be f. ⌟

6.2 Denotational Metric

In this section, we recall the notion of metric cpos introduced in [2] as a denotational model of Fuzz, and we give a model of Λ_S based on metric cpos. While we do not need the domain-theoretic nature of metric cpos to model Λ_S, we believe that the category of metric cpos is a good place to explore how metrics from denotational models and metrics from interactive semantic models are related. This is because the domain theoretic structure of the category of metric cpos directly gives rise to an interactive semantic model via Int-construction as we show in Section 6.3.2.

Let us recall the notion of (pointed) metric cpos [2].

▶ **Definition 16.** *A* (pointed) metric cpo X *consists of a metric space* $(|X|, d_X)$ *with a partial order* $\leq_X$ *on* $|X|$ *such that* $(|X|, \leq_X)$ *is a (pointed) cpo, and for all ascending chains* $(x_i)_{i\in\mathbb{N}}$ *and* $(x'_i)_{i\in\mathbb{N}}$ *on* X*, we have* $d_X\left(\bigvee_{i\in\mathbb{N}} x_i, \bigvee_{i\in\mathbb{N}} x'_i\right) \leq \bigvee_{i\in\mathbb{N}} d_X(x_i, x'_i)$.

For metric cpos X and Y, a function $f\colon |X| \to |Y|$ is said to be continuous and non-expansive when f is a continuous function from $(|X|, \leq_X)$ to $(|Y|, \leq_Y)$ and is a non-expansive function from $(|X|, d_X)$ to $(|Y|, d_Y)$. Below, we simply write X for the underlying set $|X|$.

Pointed metric cpos and continuous and non-expansive functions form a category, which is denoted by **MetCppo**. The unit object I of **MetCppo** is the unit object of **Met** equipped with the trivial partial order. The tensor product $X \otimes Y$ is given by the tensor product of metric spaces with the componentwise order. The hom-object $X \multimap Y$ is given by the set of continuous and non-expansive functions equipped with the pointwise order and

$$d_{X\multimap Y}(f, g) = \sup_{x\in X} d_Y(fx, gx).$$

We associate **MetCppo** with the structure of a model of Λ_S as follows. We define $R = \lfloor \mathbf{R} \rfloor$ to be $(\mathbb{R} \cup \{\perp\}, d_R, \leq_R)$ where d_R is the extension of the metric on $\mathbb{R}$ given by $d_R(a, \perp) = \infty$ for all $a \in \mathbb{R}$, and $(\mathbb{R} \cup \{\perp\}, \leq_R)$ is the lifting of the discrete cpo $\mathbb{R}$. For $f \in S$, we define $\lfloor f \rfloor \colon R^{\otimes \mathrm{ar}(f)} \to R$ to be the function satisfying $\lfloor f \rfloor(x_1, \ldots, x_{\mathrm{ar}(f)}) = y \in \mathbb{R}$ if and only if $x_1, \ldots, x_{\mathrm{ar}(f)} \in \mathbb{R}$ and $f(x_1, \ldots, x_{\mathrm{ar}(f)}) = y$. In the sequel, we denote the metric on Λ_S induced by this model by d^{den}, and we call the metric d^{den} the *denotational metric.*

6.3 Interactive Metric

We describe another model of Λ_S, which we call the *interactive semantic model.* Categorically speaking, the construction is based on the notion of *trace operator* and on the related *Int-construction* [21]. Below, we first sketch how terms are interpreted, and then, we formally describe the construction of the interactive semantic model.

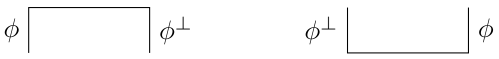
6.3.1 How Terms are Interpreted, Informally

Via the Curry-Howard correspondence, our target language can be considered as the (intuitionistic) multiplicative fragment of linear logic equipped with an atomic proposition ρ and derivation rules

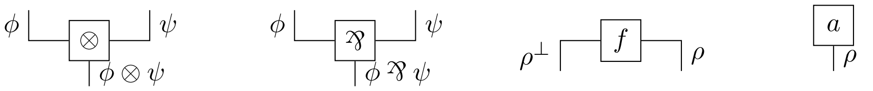
$$\frac{}{\vdash \rho^{\perp}, \ldots, \rho^{\perp}, \rho}\, f\, , \qquad \frac{}{\vdash \rho}\, a$$

for each $f \in S$ and each $a \in \mathbb{R}$. Roughly speaking, the metric induced by the interactive semantic model measures distances between terms by measuring distances between the proof structures [16] associated to the corresponding proofs in this extension of linear logic; and the distances between proof structures are given by the distances between the non-expansive functions associated to proof structures. We will use the Int-construction as a categorical machinery to associate non-expansive functions to proof structures. Below, without going into categorical detail, we illustrate how we measure distances between terms using concrete examples.

For simplicity, we suppose that S is the set of all non-expansive functions from $\mathbb{R}$ to $\mathbb{R}$. Then the proof structures associated to proofs are generated by the axiom links and the cut-links:



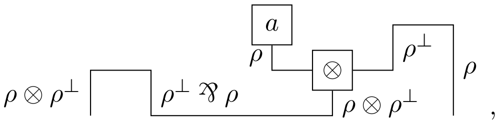
with the following nodes:



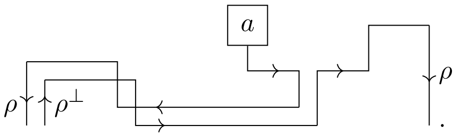
labeled by $f\colon \mathbb{R} \to \mathbb{R}$ in S and $a \in \mathbb{R}$. The first two nodes are called tensor-node and par-node, respectively. The latter two nodes corresponds to derivation rules for $f \in S$ and $a \in \mathbb{R}$. Labels $\phi, \psi, \ldots$ on edges are formulae of the multiplicative fragment of linear logic given by: $\phi, \psi ::= \rho \mid \rho^\perp \mid \phi \otimes \psi \mid \phi \parr \psi$. As usual, $(-)^\perp$ is an involutive operator inductively defined by $\rho^{\perp\perp} = \rho$, $(\phi \otimes \psi)^\perp = \phi^\perp \parr \psi^\perp$, and $(\phi \parr \psi)^\perp = \phi^\perp \otimes \psi^\perp$.

As an example, let us consider $M_a = \lambda k : \mathbf{R} \multimap \mathbf{R}.\, k\,\overline{a}$ for $a \in \mathbf{R}$. The proof and the proof structure corresponding to M_a are

$$\dfrac{\dfrac{}{\vdash \rho \otimes \rho^\perp, \rho^\perp \parr \rho} \qquad \dfrac{\dfrac{}{\vdash \rho}\,a \qquad \dfrac{}{\vdash \rho^\perp, \rho}}{\vdash \rho \otimes \rho^\perp, \rho}}{\vdash \rho \otimes \rho^\perp, \rho} \quad ,$$



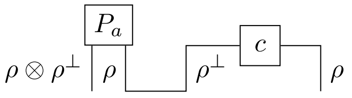
and the interactive semantic model associates this proof structure with a non-expansive function $f_a\colon \mathbb{R} \to \mathbb{R} \otimes \mathbb{R}$ given by $f_a(x) = (a, x)$. Intuitively, this non-expansive function f represents information flow on the proof structure. In this case, the information flow on the proof structure can be visualized by replacing edges labeled by $\rho \otimes \rho^\perp$ and $\rho^\perp \parr \rho$ with pairs of directed wires and removing the tensor node:



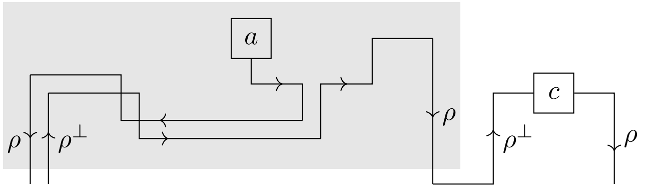
Here, we have one incoming edge labeled by $\rho^\perp$, and given an input x to this edge, we will obtain a pair of outputs: one is a from the left outgoing edge and the other is x from the right outgoing edge. This way, we obtain the function $f_a(x) = (a, x)$. Technically speaking, this graph transformation is precisely what the Int-construction does: the Int-construction provides a way to represent undirected graphs as bidirectional information flows. Finally, we can compute the distance between M_a and M_b by comparing f_a and f_b. Since $d_{\mathbb{R} \multimap \mathbb{R} \otimes \mathbb{R}}(f_a, f_b) = |a - b|$, the distance between M_a and M_b in the interactive semantic model is $|a - b|$.

An important feature of the interactive semantic model is that it provides an intensional view. For example, for $a \in \mathbb{R}$, let L_a be $\lambda k : \mathbf{R} \multimap \mathbf{R}.\, \overline{c}(k\,\overline{a})$ where $c \in S$ is the constant function given by $c(x) = 0$. Then, for all $a, b \in \mathbb{R}$, while L_a and L_b are extensionally equivalent, the interactive semantic model distinguishes L_a from L_b if $a \neq b$. Below, we explain how the interactive semantic model distinguishes these terms. The proof and the proof structure corresponding to L_a are

$$\dfrac{\dfrac{\Pi_a}{\vdash \rho \otimes \rho^{\perp}, \rho} \qquad \dfrac{}{\vdash \rho^{\perp}, \rho}\, c}{\vdash \rho \otimes \rho^{\perp}, \rho} \quad ,$$



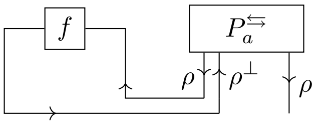
where Π_a in the left hand side denotes the proof associated to M_a, and P_a in the right hand side denotes the proof structure associated to M_a. By replacing edges in P_a with bidirectional edges (depicted in the gray region in the following graph), we obtain



.

This graph represents $g_a \colon \mathbb{R} \to \mathbb{R} \otimes \mathbb{R}$ given by $g_a(x) = (0, a)$. Because $d_{\mathbb{R} \multimap \mathbb{R} \otimes \mathbb{R}}(g_a, g_b) = |a - b|$, we see that the distance between L_a and L_b in the interactive semantic model is $|a - b|$. In other words, the interactive semantic model distinguishes L_a and L_b by looking at their computational processes, namely, applying either a or b to its argument and then returning 0, rather than their extensional behavior.

Finally, we give a remark on the name *interactive* semantic model. Interaction in the interactive semantic model can be found when we consider terms that have β-redexes. For example, the bidirectional graph associated to $M_a\ (\lambda x : \mathbf{R}.\overline{f}(x))$ is



,

where $P_a^{\leftrightarrows}$ is the bidirectional graph obtained from P_a. In this graph, we can find interaction between the node f and the node $P_a^{\leftrightarrows}$.

6.3.2 The Interactive Semantic Model, Formally

In order to formally describe the interactive semantic model, we first observe that the category **MetCppo** has a trace operator, which is necessary to apply the Int-construction to **MetCppo**. For $f \colon X \otimes Z \to Y \otimes Z$ in **MetCppo**, we define $\mathrm{tr}^{Z}_{X,Y}(f) \colon X \to Y$ by

$$\mathrm{tr}^{Z}_{X,Y}(f)(x) = \text{the first component of } f(x, z)$$

where z is the least fixed point of the continuous function $f(x, -) \colon Z \to Z$. When we ignore metric enrichment, the definition of $\mathrm{tr}^{Z}_{X,Y}(f)$ coincides with the definition of the trace operator associated to the least fixed point operator on the category of pointed cpos and continuous functions [18]. Hence, in order to show that $\mathrm{tr}^{Z}_{X,Y}$ is a trace operator, it is enough to check non-expansiveness of $\mathrm{tr}^{Z}_{X,Y}(f)$.

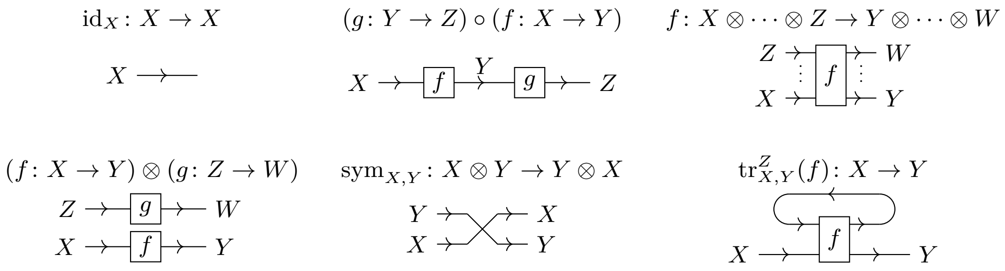

$\mathrm{id}_X : X \to X$
X
$(g \colon Y \to Z) \circ (f \colon X \to Y)$
X
f
Y
g
Z
$f \colon X \otimes \cdots \otimes Z \to Y \otimes \cdots \otimes W$
Z
W
f
X
Y
$(f \colon X \to Y) \otimes (g \colon Z \to W)$
Z
g
W
X
f
Y
$\mathrm{sym}_{X,Y} \colon X \otimes Y \to Y \otimes X$
Y
X
X
Y
$\mathrm{tr}^Z_{X,Y}(f) \colon X \to Y$
f
X
Y

Figure 5 String Diagrams for the Traced Symmetric Monoidal Structure.

▶ **Proposition 17.** *The symmetric monoidal category* $(\mathbf{MetCppo}, I, \otimes)$ *equipped with the family of operators* $\{\mathrm{tr}^Z_{X,Y}\}_{X,Y,Z \in \mathbf{MetCppo}}$ *is a traced symmetric monoidal category.*

Now, we can apply the Int-construction to **MetCppo** and obtain a symmetric monoidal *closed* category $\mathbf{Int}(\mathbf{MetCppo})$. (In fact, what we obtain is a *compact closed category*, and we only need its symmetric monoidal closed structure to interpret Λ_S.) Objects in $\mathbf{Int}(\mathbf{MetCppo})$ are pairs $X = (X_+, X_-)$ consisting of objects X_+ and X_- in **MetCppo**, and a morphism from X to Y in $\mathbf{Int}(\mathbf{MetCppo})$ is a morphism from $X_+ \otimes Y_-$ to $X_- \otimes Y_+$ in **MetCppo**. The identity on (X_+, X_-) is the symmetry $X_+ \otimes X_- \cong X_- \otimes X_+$, and the composition of $f \colon (X_+, X_-) \to (Y_+, Y_-)$ is given by

$$\mathrm{tr}^{Y_- \otimes Y_+}_{X_+ \otimes Z_-, X_- \otimes Z_+}((X_- \otimes \theta) \circ (f \otimes g) \circ (X_+ \otimes \theta'))$$

where $\theta \colon Y_+ \otimes Y_- \otimes Z_+ \to Z_+ \otimes Y_- \otimes Y_+$ and $\theta' \colon Y_- \otimes Y_+ \otimes Z_- \to Z_- \otimes Y_- \otimes Y_+$ are the canonical isomorphisms, and we omit some coherence isomorphisms. The symmetric monoidal closed structure of $\mathbf{Int}(\mathbf{MetCppo})$ is given as follows. The tensor unit is (I, I), and the tensor product $X \otimes Y$ is $(X_+ \otimes Y_+, X_- \otimes Y_-)$. The hom-object $X \multimap Y$ is $(X_- \otimes Y_+, X_+ \otimes Y_-)$. For more details on the categorical structure of $\mathbf{Int}(\mathbf{MetCppo})$, see [21, 30].

We associate $\mathbf{Int}(\mathbf{MetCppo})$ with the structure of a model of Λ_S as follows. We define $\lfloor \mathbf{R} \rfloor$ to be (R, I), and for each $f \in S$, we define $\lfloor f \rfloor \colon (R, I)^{\otimes \mathrm{ar}(f)} \to (R, I)$ by

$$R^{\otimes \mathrm{ar}(f)} \otimes I \xrightarrow{\cong} R^{\otimes \mathrm{ar}(f)} \xrightarrow{\text{the interpretation of } \overline{f} \text{ in } \mathbf{MetCppo}} R \xrightarrow{\cong} I^{\otimes \mathrm{ar}(f)} \otimes R.$$

We write d^{int} for the metric on Λ_S induced by the interactive semantic model, and we call d^{int} the *interactive metric.*

In Figure 6, we describe the interpretation of Λ_S in $\mathbf{Int}(\mathbf{MetCppo})$ in terms of string diagrams. Here, we write $\llparenthesis \tau \rrparenthesis_+$ and $\llparenthesis \tau \rrparenthesis_-$ for the positive part and the negative part of the interpretation of τ, and we write $\llparenthesis \Gamma \vdash M : \tau \rrparenthesis$ for the interpretation of a term $\Gamma \vdash M : \tau$. See Figure 5 (and [30]) for the meaning of string diagrams. The interpretation $\llparenthesis \vdash * : \mathbf{I} \rrparenthesis$ is not in Figure 6 since $\llparenthesis \vdash * : \mathbf{I} \rrparenthesis$ is the identity on the unit object I, which is presented by zero wires. In the interpretation of $\overline{f}(M_1, \dots, M_{\mathrm{ar}(f)})$, we suppose that $\mathrm{ar}(f) = 2$ for legibility.

7 Finding Your Way Around the Zoo

We describe how the admissible metrics on Λ_S considered in this paper are related. Below, for metrics $d = \{d_{\Gamma,\tau}\}_{\Gamma \in \mathbf{Env}, \tau \in \mathbf{Ty}}$ and $d' = \{d'_{\Gamma,\tau}\}_{\Gamma \in \mathbf{Env}, \tau \in \mathbf{Ty}}$ on Λ_S, we write $d \leq d'$ when for all terms $\Gamma \vdash M : \tau$ and $\Gamma \vdash N : \tau$, we have $d_{\Gamma,\tau}(M, N) \leq d'_{\Gamma,\tau}(M, N)$. We write $d < d'$ when we have $d \leq d'$ and $d \neq d'$. Our main results are about the relationships between the various metrics on Λ_S, as from Figure 1.

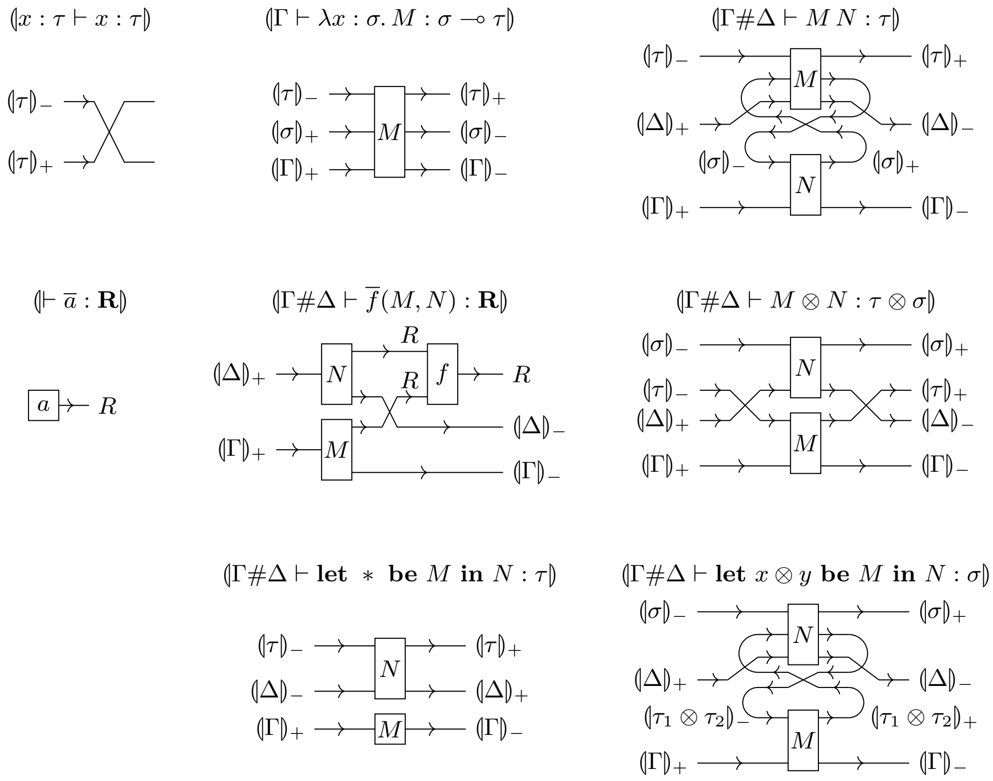


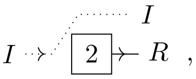
Figure 6 The Interpretation of Λ_S in $\mathbf{Int}(\mathbf{MetCppo})$.

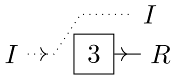
▶ **Theorem 18.** *The following inclusions hold.*

1. *For any admissible metric d on Λ_S, we have $d^{\mathrm{log}} = d^{\mathrm{obs}} \leq d \leq d^{\mathrm{equ}}$.*
2. *$d^{\mathrm{log}} = d^{\mathrm{obs}} \leq d^{\mathrm{den}} < d^{\mathrm{int}} \leq d^{\mathrm{equ}}$.*

The first claim in the above theorem states that the observational metric is the least admissible metric and the equational metric is the greatest admissible metric. In the proof, the conditions (A1), (A2), (A3) and (A4) in the definition of admissibility play different roles. While $d^{\mathrm{obs}} \leq d$ follows from (A1), (A3) and (A4), $d \leq d^{\mathrm{equ}}$ follows from (A1), (A2) and (A4). In the long version [12], we also show the converse of this statement. Namely, if a metric d on Λ_S satisfies (A1) and $d^{\mathrm{obs}} \leq d \leq d^{\mathrm{equ}}$, then d is admissible. This implies the notion of admissibility captures reasonable class of metrics on Λ_S. The second claim in the main theorem is what illustrated in Figure 1. The inequalities $d^{\mathrm{obs}} \leq d^{\mathrm{den}}$ and $d^{\mathrm{int}} \leq d^{\mathrm{equ}}$ follow from the first claim; the proof of the strict inequality $d^{\mathrm{den}} < d^{\mathrm{int}}$ is deferred to the next section.

Concrete metrics in-between d^{obs} and d^{equ} are useful to approximately compute d^{obs} and d^{equ}. For example, it is not easy to *directly* prove $d^{\mathrm{equ}}_{(k:\mathbf{R}\multimap\mathbf{I}),\mathbf{I}}(k\,\overline{2}, k\,\overline{3}) \geq 1$ since we need to know that *whenever* $k : \mathbf{R} \multimap \mathbf{I} \vdash k\,\overline{2} \approx_r k\,\overline{3} : \mathbf{I}$ is derivable, we have $r \geq 1$. Let us give a semantic proof for the inequality $d^{\mathrm{equ}}_{(k:\mathbf{R}\multimap\mathbf{I}),\mathbf{I}}(k\,\overline{2}, k\,\overline{3}) \geq 1$. Here, we use the interactive semantic model. The interpretations of these terms in the interactive semantic model are





where we can *directly* see the values applied to k. Hence, we obtain $d^{\mathrm{int}}_{(k:\mathbf{R}\multimap\mathbf{I}),\mathbf{I}}(k\,\overline{2}, k\,\overline{3}) = 1$. Then, the claim follows from $d^{\mathrm{int}} \leq d^{\mathrm{equ}}$.

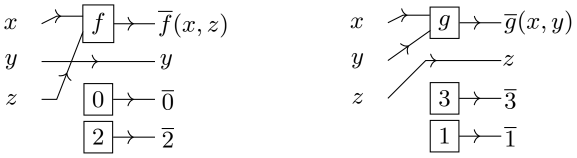


Figure 7 String diagrams with Int-terms for $M = \overline{f}(x(y\overline{0}), z\overline{2})$ and $N = \overline{g}(x(z\overline{1}), y\overline{3})$.

8 Comparing the Two Denotational Viewpoints

We show that by switching from **MetCppo** to the interactive semantic model via the Int-construction, one obtains a more discriminating metric. In other words, our goal is to establish that $d^{\mathrm{den}} < d^{\mathrm{int}}$. In this section, besides the standard equational theory from Section 3, we will also make reference to the standard β-reduction relation on Λ_S.

Let us start by making the interactive semantic metric more explicit. Notably, in the case of β-normal terms, computing distances in $\mathbf{Int}(\mathbf{MetCppo})$ can be reduced to computing distances in **MetCppo** as follows: a morphism from Γ to σ in $\mathbf{Int}(\mathbf{MetCppo})$ is a morphism in **MetCppo** from $\llparenthesis\Gamma\rrparenthesis_+ \otimes \llparenthesis\sigma\rrparenthesis_+$ to $\llparenthesis\Gamma\rrparenthesis_- \otimes \llparenthesis\sigma\rrparenthesis_+$, where these two objects correspond to tensors of the form $\mathbf{U} \otimes \cdots \otimes \mathbf{U}$, with $\mathbf{U} \in \{\mathbf{I}, \mathbf{R}\}$, More precisely, with any list of types Γ one can associate two natural numbers Γ^+, Γ^- defined inductively as $(\emptyset)^+ = (\emptyset)^- = 0$, $(\mathbf{U} * \Gamma)^+ = 1 + \Gamma^+$, $(\mathbf{U} * \Gamma)^- = \Gamma^-$, $(\sigma \multimap \tau * \Gamma)^+ = \sigma^- + \tau^+ + \Gamma^+$, $(\sigma \multimap \tau * \Gamma)^- = \sigma^+ + \tau^- + \Gamma^-$, $(\sigma \otimes \tau * \Gamma)^+ = \sigma^+ + \tau^+ + \Gamma^+$, $(\sigma \otimes \tau * \Gamma)^- = \sigma^- + \tau^- + \Gamma^-$. Then one has the following:

▶ **Proposition 19** (First-Order Int-Terms)**.** *Let M, N be β-normal terms such that $\Gamma \vdash M, N : \sigma$ and let $m = \Gamma^+ + \sigma^-$, $n = \Gamma^- + \sigma^+$. Then there exist* first-order *linear terms $H_1^M, \dots, H_n^M$, depending on variables $x_1, \dots, x_m$, and a partition $I_1, \dots, I_m$ of $\{1, \dots, m\}$ such that:*

- $\Gamma_j \vdash H_j^M : \mathbf{U}$*, for all $j = 1, \dots, n$, where $\Gamma_j = \{x_l : \mathbf{U} \mid l \in I_j\}$, with $\mathbf{U} \in \{\mathbf{I}, \mathbf{R}\}$;*
- $[\![M]\!]^{\mathbf{Int}(\mathbf{MetCppo})} = \bigotimes_j [\![H_j^M]\!]^{\mathbf{MetCppo}}$.

Intuitively, the variables occurring in the left-hand of $\Gamma_j \vdash H_j^M : \mathbf{U}$ correspond to the left-hand "wires" of the string diagram representation of $[\![M]\!]^{\mathbf{Int}(\mathbf{MetCppo})}$, and the first-order term H_j^M describes what exits from i-th right-hand "wire" of $[\![M]\!]^{\mathbf{Int}(\mathbf{MetCppo})}$.

▶ **Example 20.** Let $M = \overline{f}(x(y\overline{0}), z\overline{2})$ and $N = \overline{g}(x(z\overline{1}), y\overline{3})$, so that $\Gamma \vdash M, N : \mathbf{R}$, where $\Gamma = \{x : \mathbf{R} \multimap \mathbf{R}, y : \mathbf{R} \multimap \mathbf{R}, z : \mathbf{R} \multimap \mathbf{R}\}$. The string diagram representations of M and N, with the associated Int-terms, are illustrated in Fig. 7. ⌟

From Proposition 19 we can now deduce the following:

▶ **Corollary 21.** *For all β-normal terms M, N, $d^{\mathrm{int}}(M, N) = \sum_{j=1}^{n} d^{\mathrm{den}}(H_j^M, H_j^N)$.*

For instance, in the case of Example 20, the distance $d^{\mathrm{int}}(M, N)$ coincides with the sum of the distances, computed in **MetCppo**, between the Int-terms illustrated in Fig. 7.

We can use Corollary 21 to show that the equality $d^{\mathrm{int}} = d^{\mathrm{den}}$ cannot hold. For instance, while $d^{\mathrm{den}}_{(k:\mathbf{R}\multimap\mathbf{I}),\mathbf{I}}(k\overline{2}, k\overline{3}) = 0$, by computing the Int-terms $H_1^{k\overline{2}}(x) = H_1^{k\overline{3}}(x) = x$, $H_2^{k\overline{3}} = \overline{2}$, $H_2^{k\overline{3}} = \overline{3}$ we deduce $d^{\mathrm{int}}_{(k:\mathbf{R}\multimap\mathbf{I}),\mathbf{I}}(k\overline{2}, k\overline{3}) = 0 + 1 = 1$.

It remains to prove then that $d^{\mathrm{den}} \leq d^{\mathrm{int}}$.

▶ **Theorem 22.** *For all M, N such that $\Gamma \vdash M, N : \sigma$ holds, $d^{\mathrm{den}}(M, N) \leq d^{\mathrm{int}}(M, N)$.*

Proof sketch. It suffices to prove the claim for M, N β-normal, using the fact that, if M^* and N^* are the β-normal forms of M, N, then $d^{\mathrm{den}}(M, M^*) = d^{\mathrm{den}}(N, N^*) = 0$, and moreover $d^{\mathrm{int}}(M^*, N^*) \leq d^{\mathrm{int}}(M, N)$, as a consequence of the non-expansiveness of the trace operator. Recall that

$$d^{\mathrm{den}}(M, N) = \sup\{d^{\mathrm{den}}_{\sigma}([\![M]\!]^{\mathbf{MetCppo}}(\vec{a}), [\![N]\!]^{\mathbf{MetCppo}}(\vec{a})) \mid \vec{a} \in [\![\Gamma]\!]^{\mathbf{MetCppo}}\},$$

$$d^{\mathrm{int}}(M, N) = \sup\left\{ \sum_{i=1}^{n} d^{\mathrm{den}}_{\mathbf{R}}(H^M_i[\vec{r}], H^N_i[\vec{r}]) \;\middle|\; \vec{r} \in \mathbb{R}^m \right\}.$$

For fixed $\vec{a} \in [\![\Gamma]\!]^{\mathbf{MetCppo}}$ we will construct reals $\vec{r} \in \mathbb{R}^m$, a sequence of terms $M = M_0, \dots, M_k = N$, where $k = \Gamma^- + \sigma^+$, and a bijection $\rho : \{1, \dots, k\} \to \{1, \dots k\}$ such that the distance between $M_i[\vec{a}]$ and $M_{i+1}[\vec{a}]$ is bounded by the distance between the Int-terms $H^M_{\rho(i+1)}[\vec{r}]$ and $H^N_{\rho(i+1)}[\vec{r}]$. In this way we can conclude by a finite number of applications of the triangular law that

$$\begin{aligned} d^{\mathrm{den}}_{\sigma}(M[\vec{a}], N[\vec{a}]) &\leq d^{\mathrm{den}}_{\sigma}(M_0[\vec{a}], M_1[\vec{a}]) + \cdots + d^{\mathrm{den}}_{\sigma}(M_{k-1}[\vec{a}], M_k[\vec{a}]) \\ &\leq d^{\mathrm{den}}_{\mathbf{R}}(H^M_{\rho(1)}[\vec{r}], H^N_{\rho(1)}[\vec{r}]) + \cdots + d^{\mathrm{den}}_{\mathbf{R}}(H^M_{\rho(k)}[\vec{r}], H^N_{\rho(k)}[\vec{r}]) \leq d^{\mathrm{int}}(M, N). \end{aligned}$$

We observe (see [12] for more details) that:

- the bound or free variables x_i of M are bijectively associated with the subterms $\phi_i M$ of M of the form $x_i\vec{Q}$ and with first-order variables α_i;
- the Int-terms $H^M_i[\alpha_{i_1}, \dots, \alpha_{i_s}]$ are bijectively associated with the subterms $\psi_i M$ of M of the form $H^M_i[\phi_{i_1} M, \dots, \phi_{i_s} M]$.

Similar observations hold for N, and ρ is defined so that, whenever $\psi_i N$ is a subterm of $\psi_j N$, $\phi(j) \leq \phi(i)$. Let us set $r_{\alpha_j} := \phi_j M[\vec{a}]$. The desired sequence is defined by letting $M_0 = M$ and M_{i+1} be obtained from M_i by replacing the subterm $\psi_{\rho(i)} M = H^M_{\rho(i)}[\phi_{i_1} M, \dots, \phi_{i_s} M]$ by $H^N_{\rho(i)}[\phi_{i_1} M, \dots, \phi_{i_s} M]$. Using the properties of ρ, one can check that this replacement is well-defined at each step, and that M_k actually coincides with N. Moreover, at each step the passage from M_i to M_{i+1} is bounded in distance by

$$\begin{aligned} d^{\mathrm{den}}(H^M_{\rho(i)}[\dots \phi_j M[\vec{a}] \dots], H^N_{\rho(i)}[\dots \phi_j M[\vec{a}] \dots]) &= d^{\mathrm{int}}(H^M_{\rho(i)}[\dots r_{a_j} \dots], H^N_{\rho(i)}[\dots r_{a_j} \dots]) \\ &\leq d^{\mathrm{int}}(H^M_{\rho(i)}, H^N_{\rho(i)}). \end{aligned}$$ ◀

▶ **Example 23.** For the terms M and N from Example 20, the procedure just sketched defines the sequence: $M = \overline{f}(x(y\overline{0}), z\overline{2}) \overset{\overline{f}(x,z)\mapsto\overline{g}(x,y)}{\to} \overline{g}(x(y\overline{0}), y\overline{0}) \overset{y\mapsto z}{\to} \overline{g}(x(z\overline{2}), y\overline{0}) \overset{\overline{0}\mapsto\overline{3}}{\to} \overline{g}(x(z\overline{2}), y\overline{3}) \overset{\overline{2}\mapsto\overline{1}}{\to} \overline{g}(x(z\overline{1}), y\overline{3}) = N$, where at each step the replacement is of the form $H^M_i[\dots \varphi_j M \dots] \mapsto H^N_i[\dots \varphi_j M \dots]$. ⌟

While the argument above holds in the linear case, it does not seem to scale to graded exponentials, and in this last case we are not even sure if a result like Theorem 22 may actually hold (see also the discussion in the next section).

9 On Graded Exponentials

One of the (original) motivations of this paper was to study distances between programs in Fuzz [29], which is a linear type system designed to track *program sensitivity* . A function $f : \mathbb{R} \to \mathbb{R}$ is said to be r-sensitive for $r \in \mathbb{R}^{\infty}_{\geq 0}$ when f is a non-expansive function from $!_r\mathbb{R} \to \mathbb{R}$ where $!_r\mathbb{R}$ is the metric space of real numbers with $d_{!_r\mathbb{R}}(a, b) = r|a - b|$. Sensitivity tells how much a function depends on its arguments. Since Fuzz adopts the scaling modalities

$!_r(-)$ as type constructors, when we are to extend our results in this paper to Fuzz, we need to take care of the scaling modalities. In this section, we give some hints about which ones of our results can be extended to Fuzz.

As for Theorem 6, we can show that the metric given in [29] in terms of metric logical relations equals the observational metric $d^{\mathrm{obs}}_{\Gamma,\tau}(M,N)$ given by

$$\sup_{C[-]\colon(\Gamma,\tau)\to(\varnothing,\mathbf{R})} \inf\left\{ r\in\mathbb{R}^{\infty}_{\geq 0} \;\middle|\; \begin{array}{l}\text{if } C[M]\hookrightarrow a,\ \text{then } C[N]\hookrightarrow b \text{ and } |a-b|\leq r,\\ \text{and if } C[N]\hookrightarrow a,\ \text{then } C[M]\hookrightarrow b \text{ and } |a-b|\leq r\end{array} \right\}$$

as long as we equip Fuzz with unary multiplications $r\times(-)\colon !_r\mathbf{R}\to\mathbf{R}$ for all $r\in\mathbb{R}_{\geq 0}$. For Fuzz *without* unary multiplications, we need to define observational metric by observations at types $!_{r_1}\mathbf{R}\otimes\cdots\otimes !_{r_n}\mathbf{R}$ in order to prove that the observational metric coincides with the logical metric. We can also extend the over-approximation of the observational metric by the denotational metric to Fuzz. This follows from the adequacy theorem of **MetCppo** with respect to Fuzz shown in [2]. In **MetCppo**, the scaling modalities $!_r(-)$ of Fuzz are interpreted as scaling operators $r\cdot(-)$: for any metric space X, $r\cdot X$ is a metric space defined to be $|X|$ with $d_{r\cdot X}(x,y)=rd_X(x,y)$. Unfortunately, generalizing the over-approximation of the observational metric by the interactive semantic metric to Fuzz is not done yet. The main difficulty lies in the interpretation of the scaling modalities $!_r(-)$. Since the scaling modalities can be understood as a graded variant of the linear exponential comonad in linear logic [22], it is reasonable to explore graded variants of Abramsky and Jagadeesan's a model of linear logic based on **Int**(**Cppo**) [1]. However, at this point, we could only accommodate grades as non-negative possibly infinite *integers* [12]. We believe that this restriction is not so strong because, for example, closed terms of type $!_{k/n}\mathbf{R}\multimap !_{h/m}\mathbf{R}$ in Fuzz are "definable" as closed terms of type $!_{km}\mathbf{R}\multimap !_{hn}\mathbf{R}$.

10 Conclusion

In this paper, we study quantitative reasoning about linearly typed higher-order programs. We introduce a notion of admissibility for families of metrics on a purely linear programming language Λ_S, and among them, we investigate five notions of program metrics and how these are related, namely the logical, observational, equational, denotational, and interactive metrics. Some of our results can be seen as quantitative analogues of well-known results about program equivalences: the observational metric is never more discriminating than the semantic metrics, and non-definable functionals in the semantics are the source of inclusions. We list some open problems:

- Does the denotational metric coincide with the observational metric?
- Does the interactive metric coincide with the equational metric?
- We may define another observational metric d^{obsbase} where we observe terms only at $\mathbf{R}$. This observational metric d^{obsbase} is less than or equal to the logical metric and depends on the choice of S. For which choices of S does d^{obsbase} coincide with the logical metric?
- We have another semantic metric obtained from the category of metric spaces. How is the metric related to the denotational metric?
- There is a symmetric monoidal coreflection between **Int**(**MetCppo**) and **MetCppo** [19]. This is a strong connection between the two models. However, we do not know whether this categorical structure sheds any light on their relationship at the level of higher-order programs.

On the last point, we can say that our study reveals the intrinsic difficulty of comparing denotational models with interactive semantic models obtained by applying the Int-construction. Indeed, their relationship is not trivial already at the level of program equivalences.

Some of our results can be extended to a fragment of Fuzz where grading is restricted to extended natural numbers. Providing a quantitative equational theory and an interactive metric for full Fuzz is another very interesting topic for future work. There are some notions of metric that we have not taken into account in this paper. In [14], Gavazzo gives coinductively defined metrics for an extension of Fuzz with algebraic effects and recursive types, which we do not consider here. The so-called observational quotient [20] can be seen as a way to construct less discriminating program metrics from fine-grained ones. A thorough comparison of these notions of program distance with the ones we introduce here is another intriguing problem on which we plan to work in the future.

References

1 S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. *Information and Computation*, 111(1):53–119, 1994. doi:10.1006/inco.1994.1041.

2 Arthur Azevedo de Amorim, Marco Gaboardi, Justin Hsu, Shin-ya Katsumata, and Ikram Cherigui. A semantic account of metric preservation. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL 2017)*, pages 545–556, New York, NY, USA, 2017. Association for Computing Machinery.

3 Andrew Barber and Gordon D. Plotkin. Dual intuitionistic linear logic, 1996. Unpublished draft. An early version appeared as a technical report ECS-LFCS-96-347, LFCS, University of Edinburgh.

4 Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy: The sulq framework. In *Proceedings of the 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2005)*, pages 128–138, New York, NY, USA, 2005. Association for Computing Machinery. doi:10.1145/1065167.1065184.

5 Raphaëlle Crubillé and Ugo Dal Lago. On probabilistic applicative bisimulation and call-by-value λ-calculi. In Zhong Shao, editor, *Proceedings of the 23rd European Symposium on Programming Languages and Systems (ESOP 2014)*, volume 8410 of *Lecture Notes in Computer Science*, pages 209–228. Springer, 2014. doi:10.1007/978-3-642-54833-8_12.

6 Raphaëlle Crubillé and Ugo Dal Lago. Metric reasoning about λ-terms: The affine case. In *Proceedings of the 30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2015)*, pages 633–644, USA, 2015. IEEE Computer Society. doi:10.1109/LICS.2015.64.

7 Raphaëlle Crubillé and Ugo Dal Lago. Metric reasoning about λ-terms: The general case. In Hongseok Yang, editor, *Proceedings of the 26th European Symposium on Programming Languages and Systems (ESOP 2017)*, Lecture Notes in Computer Science, pages 341–367, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg. doi:10.1007/978-3-662-54434-1_13.

8 Raphaëlle Crubillé, Ugo Dal Lago, Davide Sangiorgi, and Valeria Vignudelli. On applicative similarity, sequentiality, and full abstraction. In Roland Meyer, André Platzer, and Heike Wehrheim, editors, *Proceedings of Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Oldenburg, Germany, September 8-9, 2015*, volume 9360 of *Lecture Notes in Computer Science*, pages 65–82. Springer, 2015. doi:10.1007/978-3-319-23506-6_7.

9 Fredrik Dahlqvist and Renato Neves. An internal language for categories enriched over generalised metric spaces. In Florin Manea and Alex Simpson, editors, *Proceedings of the 30th EACSL Annual Conference on Computer Science Logic, (CSL 2022)*, volume 216 of *LIPIcs*, pages 16:1–16:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.CSL.2022.16.

10 Ugo Dal Lago and Francesco Gavazzo. Effectul program distancing. *Proceedings of the ACM on Programming Languages*, 6(POPL):1–30, 2022. doi:10.1145/3498680.

11 Ugo Dal Lago, Furio Honsell, Marina Lenisa, and Paolo Pistone. On Quantitative Algebraic Higher-Order Theories. In Amy P. Felty, editor, *Proceedings of the 7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*, volume 228 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSCD.2022.4.

12 Ugo Dal Lago, Naohiko Hoshino, and Paolo Pistone. On the lattice of program metrics, 2023. URL: https://arxiv.org/abs/2302.05022.

13 Thomas Ehrhard, Michele Pagani, and Christine Tasson. Full abstraction for probabilistic PCF. *Journal of the ACM*, 65(4), 2018. doi:10.1145/3164540.

14 Francesco Gavazzo. Quantitative behavioural reasoning for higher-order effectful programs: Applicative distances. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2018)*, pages 452–461, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3209108.3209149.

15 Guillaume Geoffroy and Paolo Pistone. A partial metric semantics of higher-order types and approximate program transformations. In *Proceedings of the 29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, volume 183 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 35:1–35:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2021.23.

16 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

17 Jean-Yves Girard. Geometry of interaction 1: Interpretation of System F. In R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo, editors, *Proceedings of the Logic Colloquium '88*, volume 127 of *Studies in Logic and the Foundations of Mathematics*, pages 221–260. North-Holland, 1989.

18 Masahito Hasegawa. *Models of Sharing Graphs: A Categorical Semantics of Let and Letrec.* Springer-Verlag, Berlin, Heidelberg, 1999.

19 Masahito Hasegawa. On traced monoidal closed categories. *Mathematical Structures in Computer Science*, 19(2):217–244, 2009. doi:10.1017/S0960129508007184.

20 J. M. E. Hyland and C. H. L. Ong. On Full Abstraction for PCF: I, II, and III. *Information and Computation*, 163(2):285–408, 2000. doi:10.1006/inco.2000.2917.

21 André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):447–468, 1996.

22 Shin-ya Katsumata. A double category theoretic analysis of graded linear exponential comonads. In Christel Baier and Ugo Dal Lago, editors, *Proceedings of the 21st International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2018)*, pages 110–127, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-89366-2_6.

23 Soren Bogh Lassen. *Relational Reasoning about Functions and Nondeterminism.* PhD thesis, University of Aarhus, available at https://www.brics.dk/DS/98/2/BRICS-DS-98-2.pdf, 1998.

24 Ian Mackie, Leopoldo Román, and Samson Abramsky. An internal language for autonomous categories. *Applied Categorical Structures*, 1:311–343, 1993.

25 Radu Mardare, Prakash Panangaden, and Gordon Plotkin. Quantitative algebraic reasoning. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2016)*, pages 1–10, New York, NY, USA, 2016.

26 Robin Milner. Fully abstract models of typed λ-calculi. *Theoretical Computer Science*, 4(1):1–22, 1977. doi:10.1016/0304-3975(77)90053-6.

27 James Hiram Morris. *Lambda-calculus models of programming languages.* PhD thesis, Massachusetts Institute of Technology, available at https://dspace.mit.edu/handle/1721.1/64850, 1969.

28 Paolo Pistone. On generalized metric spaces for the simply typed λ-calculus. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2021)*, pages 1–14. IEEE Computer Society, 2021. doi:10.1109/LICS52264.2021.9470696.

29 Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: A calculus for differential privacy. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming (ICFP 2010)*, volume 45, pages 157–168, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1932681.1863568.

30 Peter Selinger. A survey of graphical languages for monoidal categories. In Bob Coecke, editor, *New Structures for Physics*, volume 813 of *Lecture Notes in Physics*, pages 289–355. Springer-Verlag Berlin Heidelberg, 2011.

Unifying Graded Linear Logic and Differential Operators

Flavien Breuvart
LIPN, USPN, Villetaneuse, France

Marie Kerjean
CNRS, LIPN, USPN, Villetaneuse, France

Simon Mirwasser
LIPN, USPN, Villetaneuse, France

Abstract

Linear Logic refines Classical Logic by taking into account the resources used during the proof and program computation. In the past decades, it has been extended to various frameworks. The most famous are indexed linear logics which can describe the resource management or the complexity analysis of a program. From another perspective, Differential Linear Logic is an extension which allows the linearization of proofs. In this article, we merge these two directions by first defining a differential version of Graded linear logic: this is made by indexing exponential connectives with a monoid of differential operators. We prove that it is equivalent to a graded version of previously defined extension of finitary differential linear logic. We give a denotational model of our logic, based on distribution theory and linear partial differential operators with constant coefficients.

2012 ACM Subject Classification Theory of computation → Linear logic; Mathematics of computing → Partial differential equations; Theory of computation → Denotational semantics

Keywords and phrases Linear Logic, Denotational Semantics, Functional Analysis, Graded Logic

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.21

Funding *Flavien Breuvart*: Supported by the project ANR-18-CE25-0001.
Marie Kerjean: Supported by the projects ANR-20-CE48-0002 and ANR-20-CE48-0014.

Acknowledgements We thank the anonymous reviewers for their kind and constructive comments on the first version of the paper, which greatly contributed to its improvement.

1 Introduction

Linear logic (LL) [20] and its differential counterpart [14] give a framework to study resource usages of proofs and programs. These logics were invented by enriching the syntax of proofs with new constructions observed in denotational models of λ-calculus [21, 11]. The exponential connective ! introduces non-linearity in the context of linear proofs and encapsulate the notion of resource usage. This notion was refined into *parametrised exponentials* [22, 13, 17, 19], where exponential connectives are indexed by annotations specifying different behaviors. Our aim here is to follow Kerjean's former works [25] by indexing formulas of Linear Logic with Differential Operators. Thanks to the setting of Bounded Linear Logic, we formalize and deepen the connection between Differential Linear Logic and Differential Operators.

The fundamental linear decomposition of LL is the decomposition of the usual non-linear implication $\Rightarrow$ into a linear one $\multimap$ from a set of resources represented by the new connective !: $(A \Rightarrow B) \equiv (!A \multimap B)$. Bounded Linear Logic (BLL) [22] was introduced as the first attempt to use typing systems for complexity analysis. But our interest for this logic stems from the fact that it extends LL with several exponential connectives which are indexed by *polynomially bounded intervals*. Since then, some other indexations of LL have been developed for many

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 21; pp. 21:1–21:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

purposes, for example IndLL [13] where the exponential modalities are indexed by some functions, or the graded logic $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$ [6, 19, 29] where they are indexed by the elements of a semiring $\mathcal{S}$. This theoretical development finds applications in programming languages [1, 16].

Differential linear logic [14] (DiLL) consists in an a priori distinct approach to linearity, and is based on the denotational semantics of linear proofs in terms of linear functions. In the syntax of LL, the dereliction rule states that if a proof is linear, one can then forget its linearity and consider it as non-linear. To capture differentiation, DiLL is based on a codereliction rule which is the syntactical opposite of the dereliction. It states that from a non-linear proof (or a non-linear function) one can extract a linear approximation of it, which, in terms of functions, is exactly the differential (one can notice that here, the analogy with resources does not work). Then, models of DiLL interpret the codereliction by different kinds of differentiation [10, 3].

A first step towards merging the graded and the differential extension of LL was made by Kerjean in 2018 [25]. In this paper, she defines an extension of DiLL, named D-DiLL, in which the exponential connectives ? and ! are indexed with a *fixed* linear partial differential operator with constant coefficients (LPDOcc) D. There, formulas $!_D A$ and $?_D A$ are respectively interpreted in a denotational model as spaces of functions or distributions which are solutions of the differential equation induced by D. The dereliction and codereliction rules then represent respectively the resolution of a differential equation and the application of a differential operator. This is a significant step forward in our aim to make the theory of programming languages and functional analysis closer, with a Curry-Howard perspective. In this work, we will generalize D-DiLL to a logic indexed by a monoid of LPDOcc.

Contributions. This work considerably generalizes, corrects and consolidates the extention of DiLL to differential operators sketched in [25]. It extends D-DiLL in the sense that the logic is now able to deal with all LPDOcc and combine their action. It corrects D-DiLL as the denotational interpretation of indexed exponential $?_D$ and $!_D$ are changed, leaving the interpretation of *inference rules* unchanged but reversing their type in a way that is now compatible with graded logics. Finally, this work consolidates D-DiLL by proving a cut-elimination procedure in the graded case, making use of an algebraic property on the monoid of LPDOcc.

Outline. We begin this paper in Section 2 by reviewing Differential Linear Logic and its semantics in terms of functions an distributions. We also recall the definition of $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$. Section 3 focuses on the definition of an extension of $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$, where we construct a finitary differential version for it and prove a cut-elimination theorem. The cut-elimination procedure mimicks partly the one of DiLL or $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$, but also deals with completely new interactions with inference rules. Then, Section 4 generalizes D-DiLL into a framework with several indexes and shows that it corresponds to our finitary differential $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$ indexed by a monoid of LPDOcc. It formally constructs a denotational model for it. This gives in particular a new semantics for $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$. Finally, Section 5 discusses the addition of an indexed promotion to differential $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$ and possible definitions for a semiring of differential operators.

2 Linear logic and its extensions

Linear Logic refines Classical Logic by introducing a notion of linear proofs. Formulas are defined according to the following grammar (omitting neutral elements which do not play a role here):

$$A, B := A \otimes B \mid A \parr B \mid A \,\&\, B \mid A \oplus B \mid ?A \mid !A \mid \cdots .$$

The linear negation $(\_)^{\perp}$ of a formula is *defined* on the syntax and is involutive, with in particular $(!A)^{\perp} := ?(A)^{\perp}$. The connector ! enjoys structural rules, respectively called weakening w, contraction c, dereliction d and promotion p:

$$\frac{\Gamma \vdash \Delta}{\Gamma, !A \vdash \Delta}\,\mathsf{w} \qquad \frac{\Gamma, !A, !A \vdash \Delta}{\Gamma, !A \vdash \Delta}\,\mathsf{c} \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma, !A \vdash \Delta}\,\mathsf{d} \qquad \frac{!\Gamma \vdash A}{!\Gamma \vdash !A}\,\mathsf{p}$$

These structural rules can be understood in terms of resources: a proof of $A \vdash B$ uses exactly once the hypothesis A while a proof of $!A \vdash B$ might use A an arbitrary number of times. Notice that the dereliction allows to forget the linearity of a proof by making it non-linear.

▶ Remark 1. The exponential rules for LL are recalled here in a two-sided flavour, making their denotational interpretation in Section 2.1 easier. However, we always consider a *classical* sequent calculus, and the new $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$ will be introduced later in a one-sided flavour to lightens the formalism.

Differentiation is then introduced through a "codereliction" rule $\bar{\mathsf{d}}$, which is symmetrical to d and allows to linearize a non-linear proof [14]. To express the cut-elimination with the promotion rule, other costructural rules are needed, which find a natural interpretation in terms of differential calculus. Note that the first version of DiLL, called DiLL_0, does not feature the promotion rule, which was introduced in later versions [30]. The exponential rules of DiLL_0 are then $\mathsf{w}, \mathsf{c}, \mathsf{d}$ with the following coweakening $\bar{\mathsf{w}}$, cocontraction $\bar{\mathsf{c}}$ and codereliction $\bar{\mathsf{d}}$ rules, given here in a one-sided flavour.

$$\frac{}{\vdash !A}\,\bar{\mathsf{w}} \qquad \frac{\vdash \Gamma, !A \qquad \vdash \Delta, !A}{\vdash \Gamma, \Delta, !A}\,\bar{\mathsf{c}} \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, !A}\,\bar{\mathsf{d}}$$

In the rest of the paper, as a support for the semantical interpretation of DiLL, we denote by $D_a(f)$ the differential of a function f at a point a, that is:

$$D_a f : v \mapsto \lim_{h \to 0} \frac{f(a + hv) - f(a)}{h}$$

2.1 Distribution theory as a semantical interpretation of DiLL

DiLL originates from vectorial refinements of models of LL [11], which mainly keep their discrete structure. However, the exponential connectives and rules of DiLL can also be understood as operations on smooth functions or distributions [31]. In the whole paper, $(\_)' := \mathcal{L}(\_, \mathbb{R})$ is the dual of a (topological) vector space, and *distributions with compact support* are by definition linear continuous maps on the space of smooth scalar maps, that is elements of $(\mathcal{C}^{\infty}(\mathbb{R}^n, \mathbb{R}))'$. Distributions are sometimes described as "generalized functions"[1] Let us recall the notation for Dirac operator, which is a distribution with compact support and used a lot in the rest of the paper: $\delta : v \in \mathbb{R}^n \mapsto (f \mapsto f(v)) \in (\mathcal{C}^{\infty}(\mathbb{R}^n, \mathbb{R}))'$.

Recently, Kerjean [25] gave an interpretation of the connective ? by a space of smooth scalar functions, while ! is interpreted as the space of linear maps acting on those functions, that is a space of *distributions*:

$$[\![?A]\!] := \mathcal{C}^{\infty}([\![A]\!]', \mathbb{R}) \qquad\qquad [\![!A]\!] := \mathcal{C}^{\infty}([\![A]\!], \mathbb{R})'.$$

[1] Indeed, any function with compact support $g \in \mathcal{C}^{\infty}(\mathbb{R}^n, \mathbb{R})$ acts as a distribution $T_g \in (\mathcal{C}^{\infty}(\mathbb{R}^n, \mathbb{R}))'$ with compact support, through integration: $T_g : f \mapsto \int gf$. It is indeed a distribution, as it acts linearly (and continuously) on smooth functions.

Note that the language of distributions applies to all models of DiLL as noticed by Ehrhard on Köthe spaces [10]. The focus of this model was to find smooth infinite dimensional models of DiLL, whose objects were invariant under double negation, that is a model of *classical* DiLL. This is an intricate issue, see [8], and a simple solution is to consider models of *polarized* calculus. Polarized Linear Logic LL_{pol} [27] separates formulas in two classes:

Negative Formulas: $N, M := a \mid ?P \mid \uparrow P \mid N \parr M \mid \perp \mid N \,\&\, M \mid \top$.
Positive Formulas: $P, Q := a^{\perp} \mid !N \mid \downarrow N \mid P \otimes Q \mid 0 \mid P \oplus Q \mid 1$.

We interpret formulas of LL_{pol} by Nuclear topological vector spaces, and add the condition that the spaces are Fréchet or DF according to the polarity of the formulas. Positive formulas (left stable by $\otimes$!) are interpreted as Nuclear DF spaces while Negative formulas (left stable by $\parr$?) are interpreted by Nuclear Fréchet spaces. We will not dive into the details of these definitions, see [24] for more details, but the reader should keep in mind that the formulas are always interpreted as *reflexive topological vector spaces*, that is spaces E which are isomorphic to their double dual E''. The model of functions and distribution is thus a model of *classical* DiLL, in which $[\![(\_)^{\perp}]\!] := (\_)'$.

Nicely, every exponential rule of DiLL has an interpretation in terms of functions and distributions, through the following natural transformations. In the whole paper, E and F denote topological vector spaces, which will represent the interpretation $[\![A]\!]$ and $[\![B]\!]$ of formulas A, B of DiLL. For the sake of readability, we will denote the natural transformations (*e.g.* $\mathsf{d}, \bar{\mathsf{d}}$) by the same label as the deriving rule they interpret, and likewise for connectors (*e.g.* $?, \otimes, !$) and their associated functors.

- The weakening $\mathsf{w} : \mathbb{R} \to ?E$ maps $1 \in \mathbb{R}$ to the constant function at 1, while the coweakening $\bar{\mathsf{w}} : \mathbb{R} \to !E$ maps $1 \in \mathbb{R}$ to Dirac distribution at 0: $\delta_0 : f \mapsto f(0)$.
- The dereliction $\mathsf{d} : E' \to ?(E')$ maps a linear function to itself while the codereliction $\bar{\mathsf{d}} : E \to !E$ maps a vector v to the distribution mapping a function to its differential at 0 according to the vector v:

$$\mathsf{d} : \ell \mapsto \ell \qquad \bar{\mathsf{d}} : v \mapsto (D_0(\_)(v) : f \mapsto D_0(f)(v))\,.$$

- The contraction $\mathsf{c} : ?E \otimes ?E \to ?E$ maps two scalar functions f, g to their scalar multiplication $f.g$ while the cocontraction $\bar{\mathsf{c}} : !E \otimes !E \to !E$ maps two distributions ψ and ϕ to their *convolution product* $\psi * \phi : f \mapsto \psi\,(x \mapsto \phi(y \mapsto f(x+y)))$, which is a commutative operation over distributions.

These interpretations are natural, while trying to give a semantics of a model with smooth functions and distributions. The dereliction is the one from LL, and the codereliction is the differentiation at 0, which is what differential linear logic provides. The fact that the contraction is interpreted by the scalar product comes from the kernel theorem, and the weakening is the neutral element for this operation. The cocontraction is interpreted by the convolution product, as the natural monoidal operation on distributions, with its neutral element to interpret the coweakening: the dirac operator at 0.

The natural transformations $\mathsf{w}, \bar{\mathsf{w}}, \mathsf{d}, \bar{\mathsf{d}}$ can also be directly constructed from the biproduct on topological vector spaces and Schwartz' Kernel Theorem expressing Seely isomorphisms.

2.2 Differential operators as an extension of DiLL

A first advance in merging the graded and the differential extensions of LL was made by Kerjean in 2018 [25]. In this paper, she defines an extension of DiLL named $\mathsf{D\text{-}DiLL}$. This logic is based on a *fixed single* linear partial differential operator D, which appears as a single index in exponential connectives $!_D$ and $?_D$.

The abstract interpretation of ? and ! as spaces of functions and distributions respectively allows to generalize them to spaces of solutions and parameters of differential equations. To do so, we generalize the action of $D_0(\_)$ in the interpretation of $\bar{\mathsf{d}}$ to another differential operator D. The interpretation of $\bar{\mathsf{d}}$ then corresponds to the application of a differential operator while the interpretation of d corresponds to the resolution of a differential equation (which is ℓ itself when the equation is $D_0(\_) = \ell$, but this is specifically due to the involutivity of D_0).

In D-DiLL, the exponential connectives can be indexed by a fixed differential operator. It admits a denotational semantics for a specific class of those, whose resolution is particularly easy thanks to the existence of a fundamental solution. A *Linear Partial Differential Operator with constant coefficients (LPDOcc)* acts linearly on functions $f \in \mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R})$, and by duality acts also on distributions. In what follows, each a_α will be an element of $\mathbb{R}$. By definition, only a finite number of such a_α are non-zero.

$$D : f \mapsto \left(z \mapsto \sum_{\alpha \in \mathbb{N}^n} a_\alpha \frac{\partial^{|\alpha|} f}{\partial x^\alpha}(z) \right) \qquad \hat{D} : f \mapsto \left(z \mapsto \sum_{\alpha \in \mathbb{N}^n} (-1)^{|\alpha|} a_\alpha \frac{\partial^{|\alpha|} f}{\partial x^\alpha}(z) \right) \tag{1}$$

▶ Remark 2. The coefficients $(-1)^{|\alpha|}$ in equation 1 originates from the intuition of distributions as generalized functions. With this intuition, it is natural to want that for each smooth function f, $D(T_f) = T_{D(f)}$, where T_f stands for the distribution generalizing the function f. When computing $T_{D(f)}$ on a function g with partial integration one shows that $T_{D(f)}(g) = \int D(f)g = \int f(\hat{D}(g)) = T_f \circ \hat{D}$, hence the definition.

We make D act on distributions through the following equation:

$$D(\phi) := \left(\phi \circ \widehat{D} : f \mapsto \phi(\widehat{D}(f)) \right) \in \mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R})'. \tag{2}$$

Thanks to the involutivity of D, we have $\hat{D}(\phi) = \phi \circ D$.

▶ **Definition 3.** *Let D be a LPDOcc. A* fundamental solution *of D is a distribution $\Phi_D \in \mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R})'$ such that $D(\Phi_D) = \delta_0$.*

▶ **Proposition 4** (Hormander, 1963)**.** *LPDOcc* distribute over convolution*, meaning that $D(\phi * \psi) = D(\phi) * \psi = \phi * D(\psi)$ for any $\phi, \psi \in\, !E$.*

The previous proposition is easy to check and means that knowing the fundamental solution of D gives access to the solution $\psi * \Phi_D$ of the equation $D(\_) = \psi$. It is also the reason why indexation with several differential operators is possible. Luckily for us, LPDOcc are particularly well-behaved and always have a fundamental solution. The proof of the following well-known theorem can for example be found in [23, 3.1.1].

▶ **Theorem 5** (Malgrange-Ehrenpreis)**.** *Every linear partial differential operator with constant coefficients admits exactly one fundamental solution.*

Using this result, D-DiLL gives new definitions for d and $\bar{\mathsf{d}}$, depending of a LPDOcc D:

$$\mathsf{d}_D : f \mapsto \Phi_D * f \qquad \bar{\mathsf{d}}_D : \phi \mapsto \phi \circ D.$$

These new definitions came from the following ideas. Through the involutory duality, each $v \in E$ corresponds to a unique $\delta_v \in E'' \simeq E$, and $\bar{\mathsf{d}}_D$ is then interpreted as $\phi \in E'' \mapsto \phi \circ D_0$. Then Kerjean considered that $E'' = (D_0(?(E'), \mathbb{R}))'$ and generalized it by replacing D_0 with D, defining $?_D E := D(\mathcal{C}^\infty(E', \mathbb{R}))$. This gave types $\mathsf{d}_D : ?_D E' \to ?E'$ and $\bar{\mathsf{d}}_D : !_D E \to !E$.

The reader should note that these definitions only work for finite dimensional vector spaces: one is able to apply a LPDOcc to a smooth function from $\mathbb{R}^n$ to $\mathbb{R}$ using partial differentiation on each dimension, but this is completely different if the function has an infinite dimensional domain. The exponential connectives indexed by a LPDOcc therefore only apply to *finitary* formulas: that are the formulas with no exponentials.

2.3 Indexed linear logics: resources, effects and coeffects

Since Girard's original BLL [22], several systems have implemented indexed exponentials to keep track of resource usage [9, 15]. More recently, several authors [19, 17, 6] have defined a modular (but a bit less expressive) version $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$ where the exponentials are indexed (more specifically "graded", as in graded algebras) by elements of a given semiring $\mathcal{S}$.

▶ **Definition 6.** *A* semiring $(\mathcal{S}, +, 0, \times, 1)$ *is given by a set $\mathcal{S}$ with two associative binary operations on $\mathcal{S}$: a sum $+$ which is commutative and has a neutral element $0 \in \mathcal{S}$ and a product $\times$ which is distributive over the sum and has a neutral element $1 \in \mathcal{S}$.*
Such a semiring is said to be commutative *when the product is commutative.*
An ordered semiring *is a semiring endowed with a partial order $\leq$ such that the sum and the product are monotonic.*

This type of indexation, named *grading*, has been used in particular to study effects and coeffects, as well as resources [6, 5, 17]. The main feature is to use this grading in a type system where some types are indexed by elements of the semiring. This is exactly what is done in the logic $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$, where $\mathcal{S}$ is an ordered semiring. The exponential rules of $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$ are adapted from those of LL, and agree with the intuitions that the index x in $!_x A$ is a witness for the usage of resources of type A during the proof/program.

$$\frac{\Gamma \vdash B}{\Gamma, !_0 A \vdash B}\,\mathsf{w} \qquad \frac{\Gamma, !_x A, !_y A \vdash B}{\Gamma, !_{x+y} A \vdash B}\,\mathsf{c} \qquad \frac{\Gamma, A \vdash B}{\Gamma, !_1 A \vdash B}\,\mathsf{d} \qquad \frac{!_{x_1} A_1, \dots, !_{x_n} A_n \vdash B}{!_{x_1 \times y} A_1, \dots, !_{x_n \times y} A_n \vdash !_y B}\,\mathsf{p}$$

Finally, a subtyping rule is also added, which uses the order of $\mathcal{S}$. In Section 3, we will use an order induced by the additive rule of $\mathcal{S}$, and this subtyping rule will stand for a generalized dereliction.

$$\frac{\Gamma, !_x A \vdash B \quad x \leq y}{\Gamma, !_y A \vdash B}\,\mathsf{d}_I$$

3 A differential $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$

In this section, we extend a graded linear logic with indexed coexponential rules. We define and prove correct a cut-elimination procedure.

Formulas and proofs

We define a differential version of $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$ by extending its set of exponential rules. Here, we will restrict ourselves to a version without promotion, as it has been done for DiLL originally. Following the ideas behind DiLL, we add *costructural* exponential rules: a coweakening $\bar{\mathsf{w}}$, a cocontraction $\bar{\mathsf{c}}$, an indexed codereliction $\bar{\mathsf{d}}_I$ and a codereliction $\bar{\mathsf{d}}$. The set of exponential rules of our new logic $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$ is given in Figure 1. Note that by doing so we study a *classical* version of $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$, with an involutive linear duality.

$$\frac{\vdash \Gamma}{\vdash \Gamma, ?_0 A}\,\mathsf{w} \qquad \frac{\vdash \Gamma, ?_x A, ?_y A}{\vdash \Gamma, ?_{x+y} A}\,\mathsf{c} \qquad \frac{\vdash \Gamma, ?_x A \qquad x \leq y}{\vdash \Gamma, ?_y A}\,\mathsf{d}_I \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\,\mathsf{d}$$

$$\frac{}{\vdash !_0 A}\,\bar{\mathsf{w}} \qquad \frac{\vdash \Gamma, !_x A \qquad \vdash \Delta, !_y A}{\vdash \Gamma, \Delta, !_{x+y} A}\,\bar{\mathsf{c}} \qquad \frac{\vdash \Gamma, !_x A \qquad x \leq y}{\vdash \Gamma, !_y A}\,\bar{\mathsf{d}}_I \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, !A}\,\bar{\mathsf{d}}$$

Figure 1 Exponential rules of $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$.

▶ Remark 7. In $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$, we consider a semiring $\mathcal{S}$ as a set of indices. With $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$, we do not need a semiring: since this is a promotion-free version, only one operation (the sum) is important. Hence, in $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$, $\mathcal{S}$ will only be a monoid. This modification requires two precisions:

- The indexed dereliction uses the fact that $\mathcal{S}$ is an ordered semiring. Here, the order will *always* be defined through the sum: $\forall x, y \in \mathcal{S},\ x \leq y \iff \exists x' \in \mathcal{S},\ x + x' = y$. This is due to the fact that for compatiblity with coexponential rules, we always need $\forall x, 0 \leq x$.
- In $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$, the dereliction is indexed by 1, the neutral element of the product. In $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$, we will remove this index since we do not have a product operation and simply use ! and ? instead of $!_1$ and $?_1$.

Since every element of $\mathcal{S}$ is greater than 0, we have two admissible rules which will appear in the cut elimination procedure: an indexed weakening w_I and an indexed coweakening $\bar{\mathsf{w}}_I$:

$$\frac{\vdash \Gamma}{\vdash \Gamma, ?_x A}\,\mathsf{w}_I \quad := \quad \cfrac{\cfrac{\vdash \Gamma}{\vdash \Gamma, ?_0 A}\,\mathsf{w}}{\vdash \Gamma, ?_x A}\,\mathsf{d}_I \qquad\qquad \frac{}{\vdash !_x A}\,\bar{\mathsf{w}}_I \quad := \quad \cfrac{\cfrac{}{\vdash !_0 A}\,\bar{\mathsf{w}}}{\vdash !_x A}\,\bar{\mathsf{d}}_I\ .$$

Definition of the cut elimination procedure

Since this work is done with a Curry-Howard perspective, a crucial point is the definition of a cut-elimination procedure. The cut rule is the following one

$$\frac{\vdash \Gamma, A \qquad \vdash A^{\perp}, \Delta}{\vdash \Gamma, \Delta}\,\mathit{cut}$$

which represents the composition of proofs/programs. Defining its elimination, corresponds to express explicitly how to rewrite a proof with cuts into a proof without any cut. It represents exaclty the calculus of our logic.

In order to define the cut elimination procedure of $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$, we have to consider the cases of cuts after each costructural rule that we have been introduced, since the cases of cuts after MALL rules or after w, c, d_I and d are already known. An important point is that we will use the formerly introduced indexed (co)weakening rather than the usual one.

Before giving the formal rewriting of each case, we will divide them into three groups. Since $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$ is highly inspired from DiLL, one can try to adapt the cut-elimination procedure from DiLL. This adaptation would mean that the structure of the rewriting is exactly the same, but the exponential connectives have to be indexed. For most cases, this method works and there is exactly one possible way to index these connectives, since w_I, $\bar{\mathsf{w}}_I$, c, $\bar{\mathsf{c}}$, d and $\bar{\mathsf{d}}$ do not require a choice of the index (at this point, one can think that there is a choice in the indexing of w_I and $\bar{\mathsf{w}}_I$, but this is a forced choice thanks to the other rules).

However, the case of the cut between a contraction and a cocontraction will require some work on the indexes because these two rules use the addition of the monoid. The index of the principal formula x (resp. x') of a contraction (resp. cocontraction) rule is the sum of two indexes x_1 and x_2 (resp. x_3 and x_4). But $x{=}x'$ does not imply that $x_1{=}x_3$ and $x_2{=}x_4$. We will then have to use a technical algebraic notion to decorate the indexes of the cut elimination between c and $\bar{\mathsf{c}}$ in DiLL: the additive splitting.

▶ **Definition 8.** *A monoid* $(\mathcal{M}, +, 0)$ *is* additive splitting *if for each* $x_1, x_2, x_3, x_4 \in \mathcal{M}$ *such that* $x_1 + x_2 = x_3 + x_4$*, there are elements* $x_{1,3},\ x_{1,4},\ x_{2,3},\ x_{2,4} \in \mathcal{M}$ *such that*

$$x_1 = x_{1,3} + x_{1,4} \qquad x_2 = x_{2,3} + x_{2,4} \qquad x_3 = x_{1,3} + x_{2,3} \qquad x_4 = x_{1,4} + x_{2,4}.$$

This notion appears in [5], for describing particular models of $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$, based on the relational model. Here the purpose is different: it appears from a syntactical point of view. In the rest of this section, we will not only require $\mathcal{S}$ to be a monoid, but to be additive splitting as well.

Now that we have raised some fundamental difference in a possible cut-elimination procedure, one can note that we do not have mentioned how to rewrite the cuts following an indexed (co)dereliction. This is because the procedure from DiLL cannot be adapted at all in order to eliminate those cuts, as d_I and $\bar{\mathsf{d}}_I$ have nothing in common with the exponential rules of DiLL. The situation is even worse: these cuts cannot be eliminated since these rules are not deterministic because of the use of the order relation. These considerations lead to the following division between the cut elimination cases.

Group 1: The cases where DiLL can naively be decorated. These will be cuts involving two exponential rules, with at least one being an indexed (co)weakening or a non-indexed (co)dereliction.

Group 2: The case where DiLL can be adapted using algebraic technicality, which is the cut between a contraction and a cocontraction.

Group 3: The cases highly different from DiLL. Those are the ones involving an indexed dereliction or an indexed codereliction.

The formal rewritings for the cases of groups 1 and 2 are given in Figure 2. The cut-elimination for contraction and a cocontraction uses the additive splitting property with the notations of Definition 8.

Finally, the last possible case of an occurrence of a cut in a proof is the one where d_I or $\bar{\mathsf{d}}_I$ is applied before the cut: the group 3. The following definition introduces rewritings where these rules go up in the derivation tree, and which will be applied before the cut elimination procedure. This technique is inspired from subtyping ideas, which make sense since d_I is originally defined as a subtyping rule.

▶ **Definition 9.** *The rewriting procedures* $\rightsquigarrow_{\mathsf{d}_I}$ *and* $\rightsquigarrow_{\bar{\mathsf{d}}_I}$ *are defined on proof trees of* $DB_{\mathcal{S}}LL$*.*

1. *When* d_I *(resp.* $\bar{\mathsf{d}}_I$*) is applied after a rule* r *and* r *is either from* $MALL$ *(except the axiom) or* r *is* $\bar{\mathsf{w}}_I$*,* $\bar{\mathsf{c}}$*,* $\bar{\mathsf{d}}_I$ *(resp.* w_I*,* c*,* d_I*),* $\bar{\mathsf{d}}$ *or* d*, the rewriting* $\rightsquigarrow_{\mathsf{d}_I,1}$ *(resp.* $\rightsquigarrow_{\bar{\mathsf{d}}_I,1}$*) exchanges* r *and* d_I *(resp.* $\bar{\mathsf{d}}_I$*) which is possible since* r *and* d_I *do not have the same principal formula.*
2. *When* d_I *or* $\bar{\mathsf{d}}_I$ *is applied after a (co)contraction, the rewriting is*

$$\dfrac{\dfrac{\begin{array}{c}\Pi\\ \vdash \Gamma, ?_{x_1}A, ?_{x_2}A\end{array}}{\vdash \Gamma, ?_{x_1+x_2}A}\,\mathsf{c}}{\vdash \Gamma, ?_{x_1+x_2+x_3}A}\,\mathsf{d}_I \qquad \rightsquigarrow_{\mathsf{d}_I,2} \qquad \dfrac{\dfrac{\dfrac{\begin{array}{c}\Pi\\ \vdash \Gamma, ?_{x_1}A, ?_{x_2}A\end{array}}{\vdash \Gamma, ?_{x_1+x_2}A}\,\mathsf{c}}{\vdash \Gamma, ?_{x_1+x_2}A, ?_{x_3}A}\,\mathsf{w}_I}{\vdash \Gamma, ?_{x_1+x_2+x_3}A}\,\mathsf{c}$$

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ \vdash \Gamma\end{array}}{\vdash \Gamma, ?_x A}\,\mathsf{w}_I \qquad \dfrac{}{\vdash !_x A^\perp}\,\bar{\mathsf{w}}_I}{\vdash \Gamma}\,\mathit{cut} \qquad \rightsquigarrow_{cut} \qquad \begin{array}{c}\Pi_1\\ \vdash \Gamma\end{array}$$

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ \vdash \Gamma, A\end{array}}{\vdash \Gamma, ?A}\,\mathsf{d} \qquad \dfrac{\begin{array}{c}\Pi_2\\ \vdash \Delta, A^\perp\end{array}}{\vdash \Delta, !A^\perp}\,\bar{\mathsf{d}}}{\vdash \Gamma, \Delta}\,\mathit{cut} \qquad \rightsquigarrow_{cut} \qquad \dfrac{\begin{array}{c}\Pi_1\\ \vdash \Gamma, A\end{array} \qquad \begin{array}{c}\Pi_2\\ \vdash \Delta, A^\perp\end{array}}{\vdash \Gamma, \Delta}\,\mathit{cut}$$

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ \vdash \Gamma, ?_x A, ?_y A\end{array}}{\vdash \Gamma, ?_{x+y} A}\,\mathsf{c} \qquad \dfrac{}{\vdash !_{x+y} A^\perp}\,\bar{\mathsf{w}}_I}{\vdash \Gamma}\,\mathit{cut} \qquad \rightsquigarrow_{cut} \qquad \dfrac{\dfrac{\begin{array}{c}\Pi_1\\ \vdash \Gamma, ?_x A, ?_y A\end{array} \qquad \dfrac{}{\vdash !_y A^\perp}\,\bar{\mathsf{w}}_I}{\vdash \Gamma, ?_x A}\,\mathit{cut} \qquad \dfrac{}{\vdash !_x A^\perp}\,\bar{\mathsf{w}}_I}{\vdash \Gamma}\,\mathit{cut}$$

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ \vdash \Gamma, !_x A\end{array} \qquad \begin{array}{c}\Pi_2\\ \vdash \Delta, !_y A\end{array}}{\Gamma, \Delta, !_{x+y} A}\,\bar{\mathsf{c}} \qquad \dfrac{\begin{array}{c}\Pi_3\\ \vdash \Xi\end{array}}{\vdash \Xi, ?_{x+y} A^\perp}\,\mathsf{w}_I}{\vdash \Gamma, \Delta, \Xi}\,\mathit{cut} \qquad \rightsquigarrow_{cut}$$

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ \vdash \Gamma, !_x A\end{array} \qquad \dfrac{\begin{array}{c}\Pi_3\\ \vdash \Xi\end{array}}{\vdash \Xi, ?_x A^\perp}\,\mathsf{w}_I}{\vdash \Gamma, \Xi}\,\mathit{cut}}{\vdash \Gamma, \Xi, ?_y A^\perp}\,\mathsf{w}_I \qquad \begin{array}{c}\Pi_2\\ \vdash \Delta, !_y A\end{array}}{\vdash \Gamma, \Xi, \Delta}\,\mathit{cut}$$

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ \vdash \Gamma, ?_{x_1} A^\perp, ?_{x_2} A^\perp\end{array}}{\vdash \Gamma, ?_{x_1+x_2} A^\perp}\,\mathsf{c} \qquad \dfrac{\begin{array}{c}\Pi_2\\ \vdash \Delta, !_{x_3} A\end{array} \qquad \begin{array}{c}\Pi_3\\ \vdash \Xi, !_{x_4} A\end{array}}{\vdash \Delta, \Xi, !_{x_3+x_4=x_1+x_2} A}\,\bar{\mathsf{c}}}{\vdash \Gamma, \Delta, \Xi}\,\mathit{cut} \qquad \rightsquigarrow_{cut}$$

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\Pi_b\\ \vdash \Gamma, ?_{x_{1,4}} A^\perp, ?_{x_{2,4}} A^\perp, ?_{x_3} A^\perp\end{array} \qquad \begin{array}{c}\Pi_2\\ \vdash \Delta, !_{x_3} A\end{array}}{\vdash \Gamma, \Delta, ?_{x_{1,4}} A^\perp, ?_{x_{2,4}} A^\perp}\,\mathit{cut}}{\vdash \Gamma, \Delta, ?_{x_4} A^\perp}\,\mathsf{c} \qquad \begin{array}{c}\Pi_3\\ \Xi, !_{x_4} A\end{array}}{\vdash \Gamma, \Delta, \Xi}\,\mathit{cut}$$

in which Π_a and Π_b are as follows:

$$\Pi_a = \qquad \dfrac{\dfrac{\dfrac{}{\vdash ?_{x_{2,3}} A^\perp, !_{x_{2,3}} A}\,\mathit{ax} \qquad \dfrac{}{\vdash ?_{x_{2,4}} A^\perp, !_{x_{2,4}} A}\,\mathit{ax}}{\vdash ?_{x_{2,3}} A^\perp, ?_{x_{2,4}} A^\perp, !_{x_2} A}\,\bar{\mathsf{c}} \qquad \begin{array}{c}\Pi_1\\ \vdash \Gamma, ?_{x_1} A^\perp, ?_{x_2} A^\perp\end{array}}{\vdash \Gamma, ?_{x_{2,3}} A^\perp, ?_{x_{2,4}} A^\perp, ?_{x_1} A^\perp}\,\mathit{cut}$$

$$\Pi_b = \qquad \dfrac{\dfrac{\begin{array}{c}\Pi_a\\ \vdash \Gamma, ?_{x_{2,3}} A^\perp, ?_{x_{2,4}} A^\perp, ?_{x_1} A^\perp\end{array} \qquad \dfrac{\dfrac{}{\vdash ?_{x_{1,3}} A^\perp, !_{x_{1,3}} A}\,\mathit{ax} \qquad \dfrac{}{\vdash ?_{x_{1,4}} A^\perp, !_{x_{1,4}} A}\,\mathit{ax}}{?_{x_{1,3}} A^\perp, ?_{x_{1,4}} A^\perp, !_{x_1} A}\,\bar{\mathsf{c}}}{\vdash \Gamma, ?_{x_{2,3}} A^\perp, ?_{x_{2,4}} A^\perp, ?_{x_{1,3}} A^\perp, ?_{x_{1,4}} A^\perp}\,\mathit{cut}}{\vdash \Gamma, ?_{x_{1,4}} A^\perp, ?_{x_{2,4}} A^\perp, ?_{x_3} A^\perp}\,\mathsf{c}$$

Figure 2 Cut elimination for $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$: group 1 and group 2.

$$\dfrac{\dfrac{\overset{\Pi_1}{\vdash \Gamma, !_{x_1}A} \qquad \overset{\Pi_2}{\vdash \Delta, !_{x_2}A}}{\vdash \Gamma, \Delta, !_{x_1+x_2}A}\,\bar{\mathsf{c}}}{\vdash \Gamma, \Delta, !_{x_1+x_2+x_3}A}\,\bar{\mathsf{d}}_I \quad \rightsquigarrow_{\bar{\mathsf{d}}_I,2} \quad \dfrac{\dfrac{\overset{\Pi_1}{\vdash \Gamma, !_{x_1}A} \qquad \overset{\Pi_2}{\vdash \Delta, !_{x_2}A}}{\vdash \Gamma, \Delta, !_{x_1+x_2}A}\,\bar{\mathsf{c}} \qquad \dfrac{\vdash}{\vdash !_{x_3}A}\,\bar{\mathsf{w}}_I}{\vdash \Gamma, \Delta, !_{x_1+x_2+x_3}A}\,\bar{\mathsf{c}}$$

3. *If it is applied after an indexed (co)weakening, the rewriting is*

$$\dfrac{\dfrac{\overset{\Pi}{\vdash \Gamma}}{\vdash \Gamma, ?_xA}\,\mathsf{w}_I}{\vdash \Gamma, ?_{x+y}A}\,\mathsf{d}_I \quad \rightsquigarrow_{\mathsf{d}_I,3} \quad \dfrac{\overset{\Pi}{\vdash \Gamma}}{\vdash \Gamma, ?_{x+y}A}\,\mathsf{w}_I \qquad\qquad \dfrac{\dfrac{\overset{\Pi}{\vdash}}{\vdash !_xA}\,\bar{\mathsf{w}}_I}{\vdash !_{x+y}A}\,\bar{\mathsf{d}}_I \quad \rightsquigarrow_{\bar{\mathsf{d}}_I,3} \quad \dfrac{\overset{\Pi}{\vdash}}{\vdash !_{x+y}A}\,\bar{\mathsf{w}}_I$$

4. *And if it is after an axiom, we define*

$$\dfrac{\dfrac{}{\vdash !_xA, ?_xA^\perp}\,ax}{\vdash !_xA, ?_{x+y}A^\perp}\,\mathsf{d}_I \quad \rightsquigarrow_{\mathsf{d}_I,4} \quad \dfrac{\dfrac{\dfrac{}{\vdash !_xA, ?_xA^\perp}\,ax}{\vdash !_xA, ?_xA^\perp, ?_yA^\perp}\,\mathsf{w}_I}{\vdash !_xA, ?_{x+y}A^\perp}\,\mathsf{c}$$

$$\dfrac{\dfrac{}{\vdash !_xA, ?_xA^\perp}\,ax}{\vdash !_{x+y}A, ?_xA^\perp}\,\bar{\mathsf{d}}_I \quad \rightsquigarrow_{\bar{\mathsf{d}}_I,4} \quad \dfrac{\dfrac{}{\vdash !_xA, ?_xA^\perp}\,ax \qquad \dfrac{\vdash}{\vdash !_yA}\,\bar{\mathsf{w}}_I}{\vdash !_{x+y}A, ?_xA^\perp}\,\bar{\mathsf{c}}$$

One defines $\rightsquigarrow_{\mathsf{d}_I}$ (resp. $\rightsquigarrow_{\bar{\mathsf{d}}_I}$) as the transitive closure of the union of the $\rightsquigarrow_{\mathsf{d}_I,i}$ (resp. $\rightsquigarrow_{\bar{\mathsf{d}}_I,i}$).

Even if this definition is non-deterministic, this is not a problem. Every indexed (co)dereliction goes up in the tree, without meeting another one. This implies that this rewriting is confluent: the result of the rewriting does not depend on the choices made.

▶ Remark 10. It is easy to define a forgetful functor U, which transforms a formula (resp. a proof) of $\mathsf{DB}_\mathcal{S}\mathsf{LL}$ into a formula (resp. a proof) of DiLL. For a formula A of $\mathsf{DB}_\mathcal{S}\mathsf{LL}$, $U(A)$ is A where each $!_x$ (resp. $?_x$) is transformed into $!$ (resp. $?$), which is a formula of DiLL. For a proof-tree without any d_I and $\bar{\mathsf{d}}_I$, the idea is the same: when an exponential rule of $\mathsf{DB}_\mathcal{S}\mathsf{LL}$ is applied in a proof-tree Π, the same rule but not indexed is applied in $U(\Pi)$, which is a proof-tree in DiLL. Moreover, we notice that if $\Pi_1 \rightsquigarrow_{cut} \Pi_2$, $U(\Pi_1) \rightsquigarrow_{\mathsf{DiLL}} U(\Pi_2)$ where $\rightsquigarrow_{\mathsf{DiLL}}$ is the cut-elimination in [12].

We can now define a cut-elimination procedure:

▶ **Definition 11.** *The rewriting $\rightsquigarrow$ is defined on derivation trees. For a tree Π, we apply $\rightsquigarrow_{\mathsf{d}_I}$, $\rightsquigarrow_{\bar{\mathsf{d}}_I}$ and $\rightsquigarrow_{cut}$ as long as it is possible. When there are no more cuts, the rewriting ends.*

▶ **Theorem 12.** *The rewriting procedure $\rightsquigarrow$ terminates on each derivation tree, and reaches an equivalent tree with no cut.*

In order to prove this theorem, we first need to prove a lemma, which shows that the (co)dereliction elimination is well defined.

▶ **Lemma 13.** *For each derivation tree Π, if we apply $\rightsquigarrow_{\mathsf{d}_I}$ and $\rightsquigarrow_{\bar{\mathsf{d}}_I}$ to Π, this procedure terminates such that $\Pi \rightsquigarrow_{\mathsf{d}_I} \Pi_1 \rightsquigarrow_{\bar{\mathsf{d}}_I} \Pi_2$ without any d_I and $\bar{\mathsf{d}}_I$ in Π_2.*

Proof. Let Π be a proof-tree. Each rule has a height (using the usual definition for nodes in a tree). We define the *depth* of a node as the height of the tree minus the height of this node. The procedure $\rightsquigarrow_{\mathsf{d}_I}$ terminates on Π: let $a(\Pi)$ be the number of indexed derelictions in Π and $b(\Pi)$ be the sum of the depth of each indexed derelictions in Π. Now, we define $H(\Pi) = (a(\Pi), b(\Pi))$ and $<_{lex}$ as the lexicographical order on $\mathbb{N}^2$. For each step of $\rightsquigarrow_{\mathsf{d}_I}$ such that $\Pi_i \rightsquigarrow_{\mathsf{d}_I} \Pi_j$, we have $H(\Pi_i) <_{lex} H(\Pi_j)$:

1. If $\Pi_i \rightsquigarrow_{\mathsf{d}_I,1} \Pi_j$, the number of d_I does not change and the sum of depths decreases by 1. Hence, $H(\Pi_i) <_{lex} H(\Pi_j)$.
2. If $\Pi_i \rightsquigarrow_{\mathsf{d}_I,k} \Pi_j$ with $2 \leq k \leq 4$, the number of derelictions decreases, so $H(\Pi_i) <_{lex} H(\Pi_j)$.

Using this property and the fact that $<_{lex}$ is a well-founded order on $\mathbb{N}^2$, this rewriting procedure has to terminates on a tree Π_1. Moreover, if there is an indexed dereliction in Π_1, this dereliction is below an other rule, so $\rightsquigarrow_{\mathsf{d}_I,i}$ for $1 \leq i \leq 4$ can be applied which leads to a contradiction with the definition of Π_1. Then, there is no indexed dereliction in Π_1.

Using similar arguments, the rewriting procedure $\rightsquigarrow_{\bar{\mathsf{d}}_I}$ on Π_1 ends on a tree Π_2 where there is no codereliction (and no dereliction because the procedure $\rightsquigarrow_{\bar{\mathsf{d}}_I}$ does not introduce any derelictions). ◀

Proof of Theorem 12. If we apply our procedure $\rightsquigarrow$ on a tree Π we will, using Lemma 13, have a tree $\Pi_{\mathsf{d}_I,\bar{\mathsf{d}}_I}$ such that $\Pi \rightsquigarrow_{\mathsf{d}_I} \Pi_{\mathsf{d}_I} \rightsquigarrow_{\bar{\mathsf{d}}_I} \Pi_{\mathsf{d}_I,\bar{\mathsf{d}}_I}$ and there is no dereliction and no codereliction in $\Pi_{\mathsf{d}_I,\bar{\mathsf{d}}_I}$. Hence, the procedure $\rightsquigarrow$ applied on Π gives a rewriting

$$\Pi \rightsquigarrow_{\mathsf{d}_I} \Pi_{\mathsf{d}_I} \rightsquigarrow_{\bar{\mathsf{d}}_I} \left(\Pi_{\mathsf{d}_I,\bar{\mathsf{d}}_I} = \Pi_0\right) \rightsquigarrow_{cut} \Pi_1 \rightsquigarrow_{cut} \dots$$

Applying the forgetful functor U from Remark 10 on each tree Π_i (for $i \in \mathbb{N}$), the cut-elimination theorem of DiLL [30] implies that this rewriting terminates at a rank n, because the cut-elimination rules of $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$ which are used in Π_0 are those of DiLL when the indexes are removed. Then, $\Pi \rightsquigarrow^* \Pi_n$ where Π_n is cut-free. ◀

▶ Remark 14. Notice that while DiLL is famous for introducing *formal sums of proofs* with its cut-elimination, we have none of that here. The syntactical reason is that, as exponential are labelled with indices, there is no non-deterministic choices to make here. The semantical reason is that sum is introduced while operating a cut between codereliction and contraction (differentiating a scalar multiplication of functions) or a cut between a dereliction and cocontraction (applying a convolution product of distributions to a linear map). As detailed in Section 4, LPDOcc do not behave like this and fundamental solutions or differential operators are painlessly propagated into the first argument of a distribution or function.

4 An indexed differential linear logic

In the previous section, we have defined a logic $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$ as the syntactical differential of an indexed linear logic $\mathsf{B}_{\mathcal{S}}\mathsf{LL}$, with its cut elimination procedure. It is a syntactical differentiation of BLL, as it uses the idea that differentiation is expressed through co-structural rules that mirror the structural rules of LL. Here we will take a semantical point of view: starting from differential linear logic, we will index it with LPDOcc into a logic named IDiLL, and then study the relation between $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$ and IDiLL.

4.1 IDiLL: a generalization of D-DiLL

As we saw in Section 2, Kerjean generalized $\bar{\mathsf{d}}$ and d in previous work [25], with the idea that in DiLL, the codereliction corresponds to the application of the differential operator D_0 whereas the dereliction corresponds to the resolution of the differential equation associated to D_0. This led to a logic D-DiLL, where $\bar{\mathsf{d}}$ and d have the same effect but with a LPDOcc D instead of D_0, and where the exponential connectives are indexed by this operator D. One would expect that this work could be connected to $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$, but these definitions clash with the traditional intuitions of graded logics. The first reason is syntactical: in graded logics, the exponential connectives are indexed by elements of an algebraic structure, whereas in

$$\frac{\vdash \Gamma}{\vdash \Gamma, ?_D A}\ \mathsf{w}_I \qquad \frac{\vdash \Gamma, ?_{D_1} A, ?_{D_2} A}{\vdash \Gamma, ?_{D_1 \circ D_2} A}\ \mathsf{c} \qquad \frac{\vdash \Gamma, ?_{D_1} A}{\vdash \Gamma, ?_{D_1 \circ D_2} A}\ \mathsf{d}_I$$

$$\frac{}{\vdash !_D A}\ \bar{\mathsf{w}}_I \qquad \frac{\vdash \Gamma, !_{D_1} A \qquad \vdash \Delta, !_{D_2} A}{\vdash \Gamma, \Delta, !_{D_1 \circ D_2} A}\ \bar{\mathsf{c}} \qquad \frac{\vdash \Gamma, !_{D_1} A}{\vdash \Gamma, !_{D_1 \circ D_2} A}\ \bar{\mathsf{d}}_I$$

Figure 3 Exponential rules of IDiLL.

D-DiLL only one operator is used as an index. We then change the logic D-DiLL into a logic IDiLL, which is much closer to what is done in the graded setting. In this new framework, we will consider the composition of two LPDOcc as our monoidal operation. Indeed, thanks to Proposition 4, we have that $D_1(\phi) * D_2(\psi) = (D_1 \circ D_2)(\phi * \psi)$. The convolution $*$ being the interpretation of the cocontraction rule $\bar{\mathsf{c}}$, the composition is the monoidal operation on the set of LPDOcc that we are looking for. Moreover, the composition of LPDOcc is commutative, which is a mandatory property for the monoidal operation in a graded framework. We describe the exponential rules of IDiLL in Figure 3.

The indexed rules d_D and $\bar{\mathsf{d}}_D$ are generalized to rules d_I and $\bar{\mathsf{d}}_I$ involving a variety of LPDOcc, while rules d and $\bar{\mathsf{d}}$ are ignored for now (see the first discussion of section 5). The interpretations of $?_D A$ and $!_D A$, and hence the typing of d_I and $\bar{\mathsf{d}}_I$ are changed from what D-DiLL would have directly enforced (see remark 15). Our new interpretations for $?_D A$ and $!_D A$ are now compatible with the intuition that in graded logics, rules are supposed to add information.

$$\llbracket ?_D A \rrbracket := \{g \mid \exists f \in \llbracket ?A \rrbracket,\ D(g) = f\} \qquad \llbracket !_D A \rrbracket := \llbracket ?_D A^{\perp} \rrbracket' = \hat{D}(\llbracket !A \rrbracket)$$
$$\mathsf{d}_I : \llbracket ?_{D_1} A \rrbracket \to \llbracket ?_{D_1 \circ D_2} A \rrbracket \qquad \bar{\mathsf{d}}_I : \llbracket !_{D_1} A \rrbracket \to \llbracket !_{D_1 \circ D_2} A \rrbracket$$

The reader might note that these new definitions have another benefit: they ensure that the dereliction (resp. the codereliction) is well typed when it consists in solving (resp. applying) a differential equation. This will be detailed in Section 4.3.

Notice that a direct consequence of Proposition 4 is that for two LPDOcc D_1 and D_2, $\Phi_{D_1 \circ D_2} = \Phi_{D_1} * \Phi_{D_2}$. It expresses that our monoidal law is also well-defined w.r.t. the interpretation of the indexed dereliction.

▶ Remark 15. Our definition for indexed connectives and thus for the types of d_D and $\bar{\mathsf{d}}_D$ differs from the original one in D-DiLL [25]. Kerjean gave types $\mathsf{d}_D : ?_{D,old} E' \to ?E'$ and $\bar{\mathsf{d}}_D : !_{D,old} E \to !E$. However, graded linear logic carries different intuitions: indices are here to keep track of the operations made through the inference rules. As such, d_D and $\bar{\mathsf{d}}_D$ should introduces indices D and not delete it. Compared with work in [25], we then change the interpretation of $?_D A$ and $!_D A$, and the types of d_D and $\bar{\mathsf{d}}_D$. Thanks to this change, we will see in the rest of the paper D-DiLL as a particular case of $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$.

4.2 Grading linear logic with differential operators

In this section, we will show that IDiLL consists of admissible rules of $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$ for the monoid of LPDOcc. In order to connect IDiLL with our results from Section 3, we have to study the algebraic struture of the set of linear partial differential operators with constant coefficients $\mathcal{D}$. More precisely, our goal is to prove the following theorem.

▶ **Theorem 16.** *The set $\mathcal{D}$ of LPDOcc is an additive splitting monoid under composition, with the identity operator id as the identity element.*

To prove this result, we will use multivariates polynomials: $\mathbb{R}[X^{(\omega)}] := \bigcup_{n\in\mathbb{N}} \mathbb{R}[X_1, \dots, X_n]$. It is well known that $(\mathbb{R}[X^{(\omega)}], +, \times, 0, 1)$ is a commutative ring. Its monoidal restriction is isomorphic to $(\mathcal{D}, \circ, id)$, the LPDOcc endowed with composition, through the following monoidal isomorphism

$$\chi\colon \begin{cases} (\mathcal{D}, \circ) \to (\mathbb{R}[X^{(\omega)}], \times) \\ \displaystyle\sum_{\alpha\in\mathbb{N}^n} a_\alpha \frac{\partial^{|\alpha|}(\_)}{\partial x^\alpha} \mapsto \sum_{\alpha\in\mathbb{N}^n} a_\alpha X^{\alpha_1} \dots X_n^{\alpha_n} \end{cases}$$

The following proposition is crucial in the indexation of $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$ by differential operators, since the monoid in $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$ has to be additive splitting.

▶ **Proposition 17.** *The monoid* $(\mathbb{R}[X^{(\omega)}], \times, 1)$ *is additive splitting.*

The proof requires some algebraic definitions to make it more readable.

▶ **Definition 18.** *Let $\mathcal{R}$ be a non-zero commutative ring.*

1. *$\mathcal{R}$ is an integral domain if for each $x, y \in \mathcal{R}\backslash\{0\}$, $xy \neq 0$.*
2. *An element $u \in \mathcal{R}$ is a unit if there is $v \in \mathcal{R}$ such that $uv = 1$.*
3. *Two elements $x, y \in \mathcal{R}$ are associates if x divides y and y divides x.*
4. *$\mathcal{R}$ is a factorial ring if it is an integral domain such that for each $x \in \mathcal{R}\backslash\{0\}$ there is a unit $u \in \mathcal{R}$ and $p_1, \dots, p_n \in \mathcal{R}$ irreducible elements such that $x = up_1 \dots p_n$ and for every other decomposition $vq_1 \dots q_m = up_1 \dots p_n$ (with v unit and q_i irreducible for each i) we have $n = m$ and a bijection $\sigma : \{1, \dots, n\} \to \{1, \dots, n\}$ such that p_i and $q_{\sigma(i)}$ are associated for each i.*

Proof of Proposition 17. For each integer n, the ring $\mathbb{R}[X_1, \dots, X_n]$ is factorial. This classical proposition is for example proved in [4, 2.7 Satz 7].

Let us take four polynomials P_1, P_2, P_3 and P_4 in $\mathbb{R}[X^{(\omega)}]$ such that $P_1 \times P_2 = P_3 \times P_4$. There is $n \in \mathbb{N}$ such that $P_1, P_2, P_3, P_4 \in \mathbb{R}[X_1, \dots, X_n]$.

If $P_1 = 0$ or $P_2 = 0$, then $P_3 = 0$ or $P_4 = 0$, since $\mathbb{R}[X_1, \dots, X_n]$ has integral domain. If for example $P_1 = 0$ and $P_3 = 0$, one can define

$$P_{1,3} = 0 \qquad P_{1,4} = P_4 \qquad P_{2,3} = P_2 \qquad P_{2,4} = 1$$

which gives a correct decomposition. And we can reason symmetrically for the other cases.

Now, we suppose that each polynomials P_1, P_2, P_3 and P_4 are non-zero. By factoriality of $\mathbb{R}[X_1, \dots, X_n]$, we have a decomposition

$$P_i = u_i Q_{n_{i-1}+1} \times \dots Q_{n_i} \qquad \text{(for each } 1 \leq i \leq 4)$$

where $n_0 = 0 \leq n_1 \dots \leq n_4$, u_i are units and Q_i are irreducible. Then, the equality $P_1P_2 = P_3P_4$ gives

$$u_1u_2Q_1 \dots Q_{n_2} = u_3u_4Q_{n_2+1} \dots Q_{n_4}.$$

Since u_1u_2 and u_3u_4 are units, the factoriality implies that $n_2 = n_4 - n_2$ and that there is a bijection $\sigma : \{1, \dots, n_2\} \to \{n_2 + 1, \dots, n_4\}$ such that Q_i and $Q_{\sigma(i)}$ are associates for each $1 \leq i \leq n_2$. It means that for each $1 \leq i \leq n_2$, there is a unit v_i such that $Q_{\sigma(i)} = v_iQ_i$.

Hence, defining two sets $A_3 = \sigma^{-1}(\{n_2+1,\dots,n_3\})$ and $A_4 = \sigma^{-1}(\{n_3+1,\dots,n_4\})$ we can rewrite our polynomials P_1 and P_2 using:

$$\begin{array}{lll}
A_{1,3} = A_3 \cap \{1,\dots,n_1\} = p_1,\dots,p_{m_1} & R_{1,3} = Q_{p_1}\dots Q_{p_{m_1}} & v_{1,3} = v_{p_1}\dots v_{p_{m_1}} \\
A_{1,4} = A_4 \cap \{1,\dots,n_1\} = q_1,\dots,q_{m_2} & R_{1,4} = Q_{q_1}\dots Q_{q_{m_2}} & v_{1,4} = v_{q_1}\dots v_{q_{m_2}} \\
A_{2,3} = A_3 \cap \{n_1+1,\dots,n_2\} = r_1,\dots,r_{m_3} & R_{2,3} = Q_{r_1}\dots Q_{r_{m_3}} & v_{2,3} = v_{r_1}\dots v_{r_{m_3}} \\
A_{2,4} = A_4 \cap \{n_1+1,\dots,n_2\} = s_1,\dots,s_{m_4} & R_{2,4} = Q_{s_1}\dots Q_{s_{m_4}} & v_{2,4} = v_{s_1}\dots v_{s_{m_4}}
\end{array}$$

which leads to

$$P_1 = u_1 R_{1,3} R_{1,4} \qquad P_2 = u_2 R_{2,3} R_{2,4} \qquad P_3 = u_3 v_{1,3} R_{1,3} v_{2,3} R_{2,3} \qquad P_4 = u_4 v_{1,4} R_{1,4} v_{2,4} R_{2,4}$$

Finally, we define our new polynomials

$$P_{1,3} = u_1 R_{1,3} \qquad P_{1,4} = R_{1,4} \qquad P_{2,3} = \frac{u_3 v_{1,3} v_{2,3}}{u_1} R_{2,3} \qquad P_{2,4} = \frac{u_1 u_2}{u_3 v_{1,3} v_{2,3}} R_{2,4}$$

gives the wanted decomposition: this is straightforward for P_1, P_2 and P_3 (the coefficients are chosen for that), and for P_4, it comes from the fact that $u_1 u_2 = u_3 u_4$ (which is in the definition of a factorial ring), and that $v_{1,a} v_{1,b} v_{2,a} v_{2,b} = 1$ which is easy to see using our new polynomials $R_{1,3}, R_{1,4}, R_{2,3}, R_{2,4}$ and the equality $P_1 P_2 = P_3 P_4$. ◀

This result ensures that $(\mathcal{D}, \circ, id)$ is an additive splitting monoid. Then, $\mathcal{D}$ induces a logic $\mathsf{DB}_{\mathcal{D}}\mathsf{LL}$. In this logic, since the order of the monoid is defined through the composition rule, for D_1 and D_2 in $\mathcal{D}$ we have

$$D_1 \leq D_2 \iff \exists D_3 \in \mathcal{D},\ D_2 = D_1 \circ D_3$$

which expresses that the rules d_I and $\bar{\mathsf{d}}_I$ from IDiLL and those from $\mathsf{DB}_{\mathcal{D}}\mathsf{LL}$ are exactly the same. In addition, the weakening and the coweakening from $\mathsf{DB}_{\mathcal{D}}\mathsf{LL}$ are rules which exists in IDiLL (the (co)weakening with $D = id$), and a weakening (resp. a coweakening) in IDiLL can be expressed in $\mathsf{DB}_{\mathcal{D}}\mathsf{LL}$ as an indexed weakening (resp. an indexed coweakening). In fact, this indexed weakening is the one that appears in the cut elimination procedure of $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$. Hence, this gives the following proposition.

▶ **Proposition 19.** *Each rule of IDiLL is admissible in $DB_{\mathcal{D}}LL$, and each rule of $DB_{\mathcal{D}}LL$ except* d *and* $\bar{\mathsf{d}}$ *is admissible in IDiLL.*

With this proposition, Theorem 12 ensures that IDiLL enjoys a cut elimination procedure, which is the same as the one defined for $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$. This procedure will even be easier in the case of IDiLL. One issue in the definition of the cut elimination of $\mathsf{DB}_{\mathcal{S}}\mathsf{LL}$ is to define w_I and $\bar{\mathsf{w}}_I$. This is no longer a problem in IDiLL because these rules already exist in this framework.

4.3 A concrete semantics for IDiLL

Now that we have defined the rules and the cut elimination procedure for a logic able to deal with the interaction between differential operators in its syntax, we should express how it semantically acts on smooth maps and distributions. For MALL formulas and rules, the interpretation is the same as the one for DiLL (or D-DiLL), given in Section 2. First, we give the interpretation of our indexed exponential connectives. Beware that we are still here in a *finitary setting*, in wich exponential connectives only apply to finite dimensional vector spaces, meaning that $[\![A]\!] = \mathbb{R}^n$ for some n in equation (3) below. This makes sense syntactically as long as we do not introduce a promotion rule, and corresponds to the denotational model exposed originally by Kerjean. As mentioned in the conclusion, we think that work in higher dimensional analysis should provide an higher-order interpretation for indexed exponential connectives [18].

Consider $D \in \mathcal{D}$. Then D applies independently to any $f \in \mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R})$ for any n, by injecting smoothly $\mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R}) \subseteq \mathcal{C}^\infty(\mathbb{R}^m, \mathbb{R})$ for any $m \geq n$. We give the following interpretation of graded exponential connectives:

$$\begin{aligned}
&[\![!_D A]\!] := (\{f \in \mathcal{C}^\infty([\![A]\!], \mathbb{R}) \mid \exists g \in \mathcal{C}^\infty([\![A]\!], \mathbb{R}),\ D(f) = g\})' = \hat{D}([\![!A]\!]) \\
&[\![?_D A]\!] := \{f \in \mathcal{C}^\infty([\![A]\!]', \mathbb{R}) \mid \exists g \in \mathcal{C}^\infty([\![A]\!]', \mathbb{R}),\ D(f) = g\} = D^{-1}([\![?A]\!]) \qquad (3)
\end{aligned}$$

We recall that $\hat{D}$ appears in the definition of the application of a LPDOcc to a distribution, see equation 2.

From this definition, one can note that when $D = id$, we get

$$[\![!_{id} A]\!] = (\mathcal{C}^\infty([\![A]\!], \mathbb{R}))' = [\![!A]\!] \qquad\qquad [\![?_{id} A]\!] = \mathcal{C}^\infty([\![A]\!]', \mathbb{R}) = [\![?A]\!].$$

▶ Remark 20. One can notice that, as differential equations always have solutions in our case, the space of solutions $[\![?_D A]\!]$ is *isomorphic* to the function space $[\![?A]\!]$. The isomorphism in question is plainly the dereliction $\mathsf{d}_D : f \mapsto \Phi_D * f$. While our setting might be seen as too simple from the point of view of analysis, it is a first and necessary step before extending IDiLL to more intricate differential equations. If we were to explore the abstract categorical setting for our model, these isomorphisms would be relevant in a *bicategorical* setting.

The next step is to give a semantical interpretation of the exponential rules. Most of these interpretations will be quite natural, in the sense that they will be based on the intuitions given in Section 4.1 and on the model of DiLL described in previous work [25]. However, the contraction rule will require some refinements. The contraction takes two formulas $?_{D_1} A$ and $?_{D_2} A$, and contracts them into a formula $?_{D_1 \circ D_2} A$. In our model, it corresponds to the contraction of two functions $f \in \mathcal{C}^\infty(E', \mathbb{R})$ such that $D_1(f) \in \mathcal{C}^\infty(E', \mathbb{R})$ and $g \in \mathcal{C}^\infty(E', \mathbb{R})$ such that $D_2(g) \in \mathcal{C}^\infty(E', \mathbb{R})$ into a function $h \in \mathcal{C}^\infty(E', \mathbb{R})$ such that $D_1 \circ D_2(h) \in \mathcal{C}^\infty(E', \mathbb{R})$. In differential linear logic, the contraction is interpreted as the pointwise product of functions. This is not possible here, since we do not know how to compute $D_1 \circ D_2(f.g)$. We will then use the fundamental solution, which has the property that $D(\Phi_D * f) = f$. This leads to the following definition.

▶ **Definition 21.** *We define the interpretation of each exponential rule of IDiLL by:*

$$\mathsf{w} : \begin{cases} \mathbb{R} \to ?_{id} E \\ 1 \mapsto cst_1 \end{cases} \qquad\qquad \bar{\mathsf{w}} : \begin{cases} \mathbb{R} \to !_{id} E \\ 1 \mapsto \delta_0 \end{cases}$$

$$\mathsf{c} : \begin{cases} ?_{D_1} E \mathbin{\hat{\otimes}} ?_{D_2} E \to ?_{D_1 \circ D_2} E \\ \qquad f \otimes g \mapsto \Phi_{D_1 \circ D_2} * (D_1(f).D_2(g)) \end{cases} \qquad \bar{\mathsf{c}} : \begin{cases} !_{D_1} E \mathbin{\hat{\otimes}} !_{D_2} E \to !_{D_1 \circ D_2} E \\ \qquad \psi \otimes \phi \mapsto \psi * \phi \end{cases}$$

$$\mathsf{d}_I : \begin{cases} ?_{D_1} E \to ?_{D_1 \circ D_2} E \\ \quad f \mapsto \Phi_{D_2} * f \end{cases} \qquad\qquad \bar{\mathsf{d}}_I : \begin{cases} !_{D_1} E \to !_{D_1 \circ D_2} E \\ \quad \psi \mapsto \psi \circ D_2 \end{cases}$$

▶ Remark 22. One can note that we only have defined the interpretation of the (co)weakening when it is indexed by the identity. This is because, as well as for $\mathsf{DB}_\mathcal{S}\mathsf{LL}$, the one of w_I and $\bar{\mathsf{w}}_I$ can be deduced from this one, using the definition of d_I and $\bar{\mathsf{d}}_I$. This leads to

$$\mathsf{w}_I : 1 \mapsto \Phi_D * cst_1 = cst_{\Phi_D(cst_1)} \qquad\qquad \bar{\mathsf{w}}_I : 1 \mapsto \delta_0 \circ D = (f \mapsto D(f)(0)).$$

The interpretation for $\bar{\mathsf{c}}$ and c is justified by the fact that in Nuclear Fréchet or Nuclear DF spaces [25], both the $⅋$ and $\otimes$ connectors of LL are interpreted by the same completed topological tensor product $\hat{\otimes}$. They however do not apply to the same kind of spaces, as $?E$ is Fréchet while $!E$ isn't. Thus, basic operations on the interpretation of $A ⅋ B$ or $A \otimes B$ are first defined on elements $a \otimes b$ on the tensor product, and then extended by linearity and completion.

In order to ensure that Definition 21 gives a correct model of IDiLL, we should verify the well-typedness of each morphism. First, this is obvious for the weakening and the coweakening. The function cst_1 defined on E is smooth, and δ_0 is the canonical example of a distribution. Moreover, we interpret w and $\bar{\mathsf{w}}$ in the same way as in the model of DiLL on which our intuitions are based. The indexed dereliction is well-typed, because for $f \in ?_{D_1}E$, there is $g \in \mathcal{C}^\infty(E', \mathbb{R})$ such that $D_1(f) = g$ by definition. Hence, $D_1 \circ D_2(\Phi_{D_2} * f) = D_1(f) = g \in \mathcal{C}^\infty(E', \mathbb{R})$ so $\mathsf{d}_I(f) \in ?_{D_1 \circ D_2}E$. For the contraction, if $f \in ?_{D_1}E$ and $g \in ?_{D_2}E$, $D_1(f)$ and $D_2(g)$ are in $\mathcal{C}^\infty(E', \mathbb{R})$, and so is their scalar product. Hence, $D_1 \circ D_2(\mathsf{c}(f \otimes g)) = D_1(f).D_2(g)$ which is in $\mathcal{C}^\infty(E', \mathbb{R})$. The indexed codereliction is also well-typed: for $\psi \in !_{D_1}E$, equation (3) ensures that $\psi = \hat{D_1}(\psi_1)$ with $\psi_1 \in !E$, so $\psi \circ D_2 = (\psi_1 \circ D_1) \circ D_2 \in !_{D_1 \circ D_2}E$. Finally, using similar arguments for the cocontraction, if $\psi \in !_{D_1}E$ and $\phi \in !_{D_2}E$, then $\psi = \hat{D_1}(\psi_1)$ and $\phi = \hat{D_2}(\phi_1)$, with $\psi_1, \phi_1 \in !E$. Hence,

$$\psi * \phi = (\psi_1 \circ D_1) * (\phi_1 \circ D_2) = (\psi_1 * \phi_1) \circ (D_1 \circ D_2) = \widehat{D_1 \circ D_2}(\psi_1 * \phi_1) \in !_{D_1 \circ D_2}E.$$

We have then proved the following proposition.

▶ **Proposition 23.** *Each morphism* $\mathsf{w}, \bar{\mathsf{w}}, \mathsf{c}, \bar{\mathsf{c}}, \mathsf{d}_I$ *and* $\bar{\mathsf{d}}_I$ *is well-typed.*

Another crucial point to study is the compatibility between this model and the cut elimination procedure $\rightsquigarrow$. In denotational semantics, one would expect that a model is invariant w.r.t. the computation. In our case, that would mean that for each step of rewriting of $\rightsquigarrow$, the interpretation of the proof-tree has the same value.

It is easy to see that this is true for the cut $\mathsf{w}_I/\bar{\mathsf{w}}_I$, since $D(\Phi_D * cst_1)(0) = cst_1(0) = 1$. For the cut between a contraction and an indexed coweakening, the interpretation before the reduction is $\delta_0(D_1 \circ D_2)(\Phi_{D_1 \circ D_2}(D_1(f).D_2(g))) = D_1(f)(0).D_2(g)(0)$, which is exactly the interpretation after the reduction[2].

Finally, proving the invariance of our semantics over the cut between a contraction or a weakening, and a cocontraction takes slightly more work. The weakening case is enforced by linearity of the distributions, while the contraction case relies on the density of $\{\delta_x \mid x \in E\}$ in $!E$.

▶ **Lemma 24.** *The interpretation of $DB_{\mathcal{S}}LL$ with $\mathcal{D}$ as indexes is invariant over the* $\mathsf{c}/\bar{\mathsf{c}}$ *and the* $\bar{\mathsf{c}}/\mathsf{w}_I$ *cut-elimination rules, as given in Figure 2.*

Proof. Before cut-elimination, the interpretation of the $\bar{\mathsf{c}}/\mathsf{w}$ as given in Figure 2 is:

$$\begin{aligned}
&(\psi * \phi)(\Phi_{D_1 \circ D_2} * cst_1)\\
&= \psi(x \mapsto \phi(y \mapsto \Phi_{D_1} * (\Phi_{D_2} * cst_1)(x + y)))\\
&= \psi(x \mapsto \phi(y \mapsto \Phi_{D_1}(z \mapsto \Phi_{D_2} * cst_1(x + y - z))))\\
&= \psi(x \mapsto \phi(y \mapsto \Phi_{D_1}(cst_{\Phi_{D_2}(cst_1)})))
\end{aligned}$$

[2] The scalar product $(\_.\_)$ appears as the tensor product in $\mathbb{R} \otimes \mathbb{R}$, and is transparent in sequent interpretation as $\mathbb{R} = [\![\perp]\!]$.

$$\begin{aligned}
&= \psi(x \mapsto \phi(y \mapsto \Phi_{D_1}(\Phi_{D_2}(cst_1).cst_1))) \\
&= \psi(x \mapsto \phi(y \mapsto \Phi_{D_2}(cst_1).\Phi_{D_1}(cst_1))) && \text{(by homogeneity of } \phi) \\
&= \psi(x \mapsto \phi(cst_{\Phi_{D_2}(cst_1).\Phi_{D_1}(cst_1)})) \\
&= \psi(x \mapsto \phi(\Phi_{D_1}(cst_1).cst_{\Phi_{D_2}(cst_1)})) \\
&= \psi(x \mapsto \Phi_{D_1}(cst_1).\phi(cst_{\Phi_{D_2}(cst_1)})) && \text{(by homogeneity of } \phi) \\
&= \psi(cst_{\Phi_{D_1}(cst_1).\phi(cst_{\Phi_{D_2}(cst_1)})}) \\
&= \psi(\phi(cst_{\Phi_{D_2}(cst_1)}).cst_{\Phi_{D_1}(cst_1)}) \\
&= \phi(cst_{\Phi_{D_2}(cst_1)}).\psi(cst_{\Phi_{D_1}(cst_1)}) && \text{(by homogeneity of } \psi)
\end{aligned}$$

which corresponds to the interpretation of the proof after cut-elimination.

Let us tackle now the $\bar{\mathsf{c}}/\mathsf{c}$ cut-elimination case. Suppose that we have $D_1, D_2, D_3, D_4 \in \mathcal{D}$ such that $D_1 \circ D_2 = D_3 \circ D_4$. By the additive splitting property we have $D_{1,3}, D_{1,4}, D_{2,3}, D_{2,4}$ such that

$$D_1 = D_{1,3} \circ D_{1,4} \qquad D_2 = D_{2,3} \circ D_{2,4} \qquad D_3 = D_{1,3} \circ D_{2,3} \qquad D_4 = D_{1,4} \circ D_{2,4}.$$

The diagrammatic translation of the cut-elimination rule in Figure 2 is the following.

$$\begin{tikzcd}
!_{D_1}E \otimes !_{D_2}E \arrow[r, "\mathsf{c}'_{D_{1,3},D_{1,4}} \otimes \mathsf{c}'_{D_{2,3},D_{2,4}}"] \arrow[d, "\bar{\mathsf{c}}_{D_1,D_2}"] & !_{D_{1,3}}E \otimes !_{D_{1,4}}E \otimes !_{D_{2,3}}E \otimes !_{D_{2,4}}E \arrow[dd] \\
!_{D_1 \circ D_2}E = !_{D_3 \circ D_4}E \arrow[d, "\mathsf{c}'_{D_3,D_4}"] & \\
!_{D_3}E \otimes !_{D_4}E & !_{D_{1,3}}E \otimes !_{D_{2,3}}E \otimes !_{D_{1,4}}E \otimes !_{D_{2,4}}E \arrow[l, "\bar{\mathsf{c}}_{D_{1,3},D_{2,3}} \otimes \bar{\mathsf{c}}_{D_{1,4},D_{2,4}}"']
\end{tikzcd}$$

As we interpret formulas by reflexive spaces, we can without loss of generality interpret contraction as a law $\mathsf{c}'_{D_a,D_b} : !_{D_a \circ D_b}E \to !_{D_a}E \otimes !_{D_b}E$. Because we are working on finite dimensional spaces E, an application of Hahn-Banach theorem gives us that the span of $\{\delta_x \mid x \in E\}$ is dense in $!E$. As such, the interpretation of c' can be restricted to elements of the form $\delta_x \circ D_a \circ D_b \in !_{D_a \circ D_b}E$, and one checks easily that the dual of c (Definition 21) is : $\mathsf{c}'_{D_a,D_b} : \delta_x \circ D_a \circ D_b \mapsto (\delta_x \circ D_a) \otimes (\delta_x \circ D_b)$. Remember that the convolution of Dirac operators is the Dirac of the sum of points, and as such we have : $\bar{\mathsf{c}}_{D_a,D_b} : (\delta_x \circ D_a) \otimes (\delta_y \circ D_b) \mapsto (\delta_{x+y} \circ D_b \circ D_a)$. Now one can compute easily that the diagram above commutes on elements $(\delta_x \circ D_1) \otimes (\delta_y \circ D_2)$ of $!_{D_1}E \otimes !_{D_2}E$, and as such commutes on all elements by density and continuity of $\bar{\mathsf{c}}$ and c'. ◀

In order to ensure that this model is fully compatible with $\rightsquigarrow$, it also has to be invariant by $\rightsquigarrow_{\mathsf{d}_I}$ and by $\rightsquigarrow_{\bar{\mathsf{d}}_I}$. For $\rightsquigarrow_{\mathsf{d}_I}$, the interpretation of the reduction step when the indexed dereliction meets a contraction is

$$\begin{aligned}
&\Phi_{D_3} * (\Phi_{D_1 \circ D_2} * (D_1(f).D_2(g))) \\
&= \Phi_{D_1 \circ D_2 \circ D_3} * ((D_1(f).D_2(g)).cst_1) \\
&= \Phi_{D_1 \circ D_2 \circ D_3} * ((D_1(f).D_2(g)).D_3(\Phi_{D_3} * cst_1)) \\
&= \Phi_{D_1 \circ D_2 \circ D_3} * (D_1 \circ D_2(\Phi_{D_1 \circ D_2} * (D_1(f).D_2(g))).D_3(\Phi_{D_3} * cst_1))
\end{aligned}$$

which is the interpretation after the application of $\rightsquigarrow_{\mathsf{d}_I,2}$. The case with a weakening translates the fact that $\Phi_{D_1 \circ D_2} = \Phi_{D_1} * \Phi_{D_2}$. Finally, the axiom rule introduces a distribution $\psi \in !_{D_1}E$ and a smooth map $f \in !_{D_1}E$, and $\rightsquigarrow_{\mathsf{d}_I,4}$ corresponds to the equality $\Phi_{D_1 \circ D_2} * D_1(f) = \Phi_{D_2} * f$.

The remaining case is the procedure $\rightsquigarrow_{\bar{\mathsf{d}}_I}$, which is quite similar to $\rightsquigarrow_{\mathsf{d}_I}$. The invariance of the model with the cocontraction case follows from Proposition 4. For the weakening, this is just the associativity of the composition, and the axiom works because δ_0 is the neutral element of the convolution product. We can finally deduce that our model gives an interpretation which is invariant by the cut elimination procedure of Section 3.

▶ **Proposition 25.** *Each morphism* $\mathsf{w}, \bar{\mathsf{w}}, \mathsf{c}, \bar{\mathsf{c}}, \mathsf{d}_I$ *and* $\bar{\mathsf{d}}_I$ *is compatible with the cut elimination procedure* $\rightsquigarrow$.

5 Promotion and higher-order differential operators

In the previous section, we have defined a differential extension of graded linear logic, which is interpreted thanks to exponentials indexed by a monoid of differential operators. This extension is done *up-to promotion*, meaning that we do not incorporate promotion in the set of rules. There are two reasons why it makes sense to leave promotion out of the picture:

- DiLL was historically introduced without it, with a then perfectly symmetric set of rules.
- Concerning semantics, LPDOcc are only defined when acting on functions with finite dimensional codomain: $D : \mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R}) \to \mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R})$. Introducing a promotion rule would mean extending the theory of LPDOcc to higher-order functions.

In this section, we sketch a few of the difficulties one faces when trying to introduce promotion and dereliction rules indexed by differential operators, and explore possible solutions.

Graded dereliction

Indexing the promotion goes hand-in-hand with indexing the *dereliction*. In Figure 1, we introduced a basic (not indexed) dereliction and codereliction rule d and $\bar{\mathsf{d}}$. The original intuition of DiLL is that codereliction computes the differentiation at 0 of some proof. Following the intuition of D-DiLL, dereliction computes a solution to the equation $D_0(\_) = \ell$ for some ℓ. Therefore, as indexes are here to *keep track* of the computations, and following equation (3), we should have (co)derelictions indexed by D_0 as below:

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, !A}\ \bar{\mathsf{d}} \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\ \mathsf{d} \qquad\qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, !_{D_0} A}\ \bar{\mathsf{d}}_{D_0} \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?_{D_0} A}\ \mathsf{d}_{D_0}$$

Mimicking what happens in graded logics, D_0 should be the identity element for the second law in the semiring interpreting the indices of exponentials in $\mathsf{DB}_\mathcal{S}\mathsf{LL}$. However, D_0 is *not* a linear partial differential operator (even less with constant coefficient). Let us briefly compare how a LPDOcc D and D_0 act on a function $f \in \mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R})$:

$$D : f \mapsto \left(y \in \mathbb{R}^n \mapsto \sum_{\alpha \in \mathbb{N}^n} a_\alpha \frac{\partial^{|\alpha|} f}{\partial x^\alpha}(y) \right) \qquad D_0 : f \mapsto \left(y \in \mathbb{R}^n \mapsto \sum_{0 \le i \le n} y_i \frac{\partial f}{\partial x_i}(0) \right)$$

where $(x_i)_i$ is the canonical base of $\mathbb{R}^n$, y_i is the i-th coordinate of y in the base $(x_i)_i$, and $a_\alpha \in \mathbb{R}$. To include LPDOcc and D_0 in a single semiring structure, one would need to consider global differential operators generated by:

$$\mathsf{D} : f \mapsto \left((y, v) \mapsto \sum_{\alpha \in \mathbb{N}^n} a_\alpha(v) \frac{\partial^{|\alpha|} f}{\partial x^\alpha}(y) \right), \text{ with } a_\alpha \in \mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R}).$$

The algebraic structure of such a set would be more complicated, and the composition in particular would not be commutative, and as such not suitable for the first law of a semi-ring which is essential since it ensures the symmetry of the contraction and the cocontraction.

Graded promotion

To introduce a promotion law in $\mathsf{DB}_\mathcal{S}\mathsf{LL}$, we need to define a multiplicative law $\odot$ on $\mathcal{D}$, with D_0 as a unit. We will write it under a digging form:

$$\frac{\vdash \Gamma, ?_{D_1}?_{D_2}A}{\vdash \Gamma, ?_{D_1 \odot D_2}A}\ \mathsf{dig}$$

This relates with recent work by Kerjean and Lemay [26], inspired by preexisting mathematical work in infinite dimensional analysis [18]. They show that in particular quantitative models, one can define the exponential of elements of $!A$, such that $e^{D_0} : \mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R}) \to \mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R})$ is the identity. It hints at a possible definition of the multiplicative law as $D_1 \odot D_2 := D_1 \circ e^{D_2}$.

Even if one finds a semi-ring structure on the set of all LPDOcc, the introduction of promotion in the syntax means higher-order functions in denotational models. Indexed exponential connectives are defined so-far thanks to the action of LPDOcc on functions with a finite number of variable. To make LPDOcc act on higher order function (*e.g.* elements of $\mathcal{C}^\infty(\mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R}), \mathbb{R})$ and not only $\mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R})$) one would need to find a definition of partial differential operators independent from any canonical base, which seems difficult. Moreover, contrarily to what happens regarding the differentiation of the composition of function, no higher-order version of the chain rule exists for the action of LPDOcc on the composition of functions. A possible solution could come from differentiable programming [7], in which differentials of first-order functions are propagated through higher-order primitives.

As a trick to bypass some of these issues, we could consider that the $!_D$ modalities are not composable. This is possible in a framework similar to the original BLL or that of IndLL [13], where indexes have a source and a target.

6 Conclusion

In this paper, we define a multi-operator version to $\mathsf{D\text{-}DiLL}$, which turns out to be the finitary differential version of Graded Linear Logic. We describe the cut-elimination procedure and give a denotational model of this calculus in terms of differential operators. This provides a new and unexpected semantics for Graded Linear Logic, and tighten the links between Linear Logic and Functional Analysis.

There are several directions to explore now that the proof theory of $\mathsf{DB}_\mathcal{S}\mathsf{LL}$ has been established. The obvious missing piece in our work is the *categorical axiomatization* of our model. In a version with promotion, that would consist in a differential version of bounded linear exponentials [6]. A first study based on with differential categories [2] was recently done by Pacaud-Lemay and Vienney [28]. While similarities will certainly exist in categorical models of $\mathsf{DB}_\mathcal{S}\mathsf{LL}$, differences between the dynamic of LPDOcc and of differentiation at 0 will certainly require adaptation. In particular, the treatment of the sum will require attention (proof do not need to be summed here while differential categories are additive). Finally, beware that our logic does not yet extend to higher-order and that without a concrete higher-model it might be difficult to design elegant categorical axioms.

Another line of research would consist in introducing more complex differential operators as indices of exponential connectives. Equations involving LPDOcc are extremely simple to manipulate as they are solved in a one step computation (by applying a convolution product with their fundamental solution). The vast majority of differential equations are difficult if not impossible to solve. One could introduce fixpoint operators within the theory of $\mathsf{DB}_\mathcal{S}\mathsf{LL}$, to try and modelize the resolution of differential equation by fixed point. This could also be combined with the study of particularly stable classes of differential operators, as *D-finite operators*. We would also like to understand the link between our model, where exponentials are graded with differential operators, with another new model of linear logic where morphisms corresponds to linear or non-linear differential operators [32].

References

1 Jean-Philippe Bernardy, Mathieu Boespflug, Ryan R. Newton, Simon Peyton Jones, and Arnaud Spiwack. Linear haskell: practical linearity in a higher-order polymorphic language. In *Principles of Programming Languages 2018 (POPL 2018)*. ACM, January 2018.

2 Rick Blute, Robin Cockett, and Robert Seely. Differential categories. *Mathematical Structures in Computer Science*, 16(6), 2006.

3 Rick Blute, Thomas Ehrhard, and Christne Tasson. A convenient differential category. *Les cahiers de topologie et de géométrie différentielle catégorique*, 2012.

4 Siegfried Bosch. *Algebra*. Springer-Lehrbuch. Springer, 2009.

5 Flavien Breuvart and Michele Pagani. Modelling Coeffects in the Relational Semantics of Linear Logic. In *Computer Science Logic (CSL)*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl, 2015.

6 Aloïs Brunel, Marco Gaboardi, Damiano Mazza, and Steve Zdancewic. A core quantitative coeffect calculus. In *Programming Languages and Systems*. Springer Berlin Heidelberg, 2014.

7 Aloïs Brunel, Damiano Mazza, and Michele Pagani. Backpropagation in the Simply Typed Lambda-calculus with Linear Negation. *Principles of Programming Languages*, 2020.

8 Y. Dabrowski and M. Kerjean. Models of Linear Logic based on the Schwartz epsilon product. *Theory and Applications of Categories*, 2020.

9 Ugo Dal Lago and Martin Hofmann. Bounded linear logic, revisited. In *Typed Lambda Calculi and Applications (TLCA)*. Springer Berlin Heidelberg, 2009.

10 Thomas Ehrhard. On Köthe Sequence Spaces and Linear Logic. *Mathematical Structures in Computer Science*, 12(5), 2002.

11 Thomas Ehrhard. Finiteness spaces. *Mathematical Structures in Computer Science*, 15(4), 2005.

12 Thomas Ehrhard. An introduction to differential linear logic: proof-nets, models and antiderivatives. *Mathematical Structures in Computer Science*, 28(7), 2018.

13 Thomas Ehrhard and Antonio Bucciarelli. On phase semantics and denotational semantics: the exponentials. *Annals of Pure and Applied Logic*, 109(3), 2001.

14 Thomas Ehrhard and Laurent Regnier. Differential interaction nets. *Theoretical Computer Science*, 364(2), 2006.

15 Yōji Fukihara and Shin-ya Katsumata. Generalized bounded linear logic and its categorical semantics. In *Foundations of Software Science and Computation Structures (FoSSaCS)*. Springer International Publishing, 2021.

16 Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin Pierce. Linear Dependent Types for Differential Privacy. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '13. ACM, 2013.

17 Marco Gaboardi, Shin-ya Katsumata, Dominic Orchard, Flavien Breuvart, and Tarmo Uustalu. Combining effects and coeffects via grading. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, International Conference on Functional Programming, ICFP. Association for Computing Machinery, 2016.

18 R. Gannoun, R. Hachaichi, H. Ouerdiane, and A. Rezgui. Un théorème de dualité entre espaces de fonctions holomorphes à croissance exponentielle. *Journal of Functional Analysis*, 171(1), 2000.

19 Dan Ghica and Alex I. Smith. Bounded linear types in a resource semiring. In *Programming Languages and Systems,*, European Symposium on Programming, (ESOP). Springer Berlin Heidelberg, 2014.

20 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1), 1987.

21 Jean-Yves Girard. Normal functors, power series and λ-calculus. *Annals of Pure and Applied Logic*, 1988.

22 Jean-Yves Girard, Andre Scedrov, and Philip Scott. Bounded linear logic. *Theoretical Computer Science*, 9, August 1991.

23 Lars Hormander. *Linear partial differential operators.* Springer Berlin, 1963.

24 Hans Jarchow. Locally convex spaces. B. G. Teubner Stuttgart, 1981. Mathematical Textbooks.

25 Marie Kerjean. A logical account for linear partial differential equations. In *Logic in Computer Science (LICS), Proceedings.* Association for Computing Machinery, 2018.

26 Marie Kerjean and Jean-Simon Pacaud Lemay. Taylor Expansion as a Monad in Models of Dill, 2023. preprint.

27 O. Laurent. *Etude de la polarisation en logique.* Thèse de Doctorat, Université Aix-Marseille II, March 2002.

28 Jean-Simon Pacaud Lemay and Jean-Baptiste Vienney. Graded differential categories and graded differential linear logic, 2023. preprint.

29 Paul-André Melliès. Parametric monads and enriched adjunctions, 2012. preprint.

30 Michele Pagani. The cut-elimination theorem for differential nets with promotion. In *International Conference on Typed Lambda Calculus and Applications*, 2009.

31 L. Schwartz. *Théorie des distributions.* Publications de l'Institut de Mathématique de l'Université de Strasbourg, No. IX-X. Hermann, Paris, 1966.

32 James Wallbridge. Jets and differential linear logic. *Mathematical Structures in Computer Science*, 30(8), 2020. `doi:10.1017/S0960129520000249`.

α-Avoidance

Samuel Frontull
Universität Innsbruck, Austria

Georg Moser
Universität Innsbruck, Austria

Vincent van Oostrom
Barendrecht, The Netherlands

Abstract

When substitutions and bindings interact, there is a risk of undesired side effects if the substitution is applied naïvely. The λ-calculus captures this phenomenon concretely, as β-reduction may require the renaming of bound variables to avoid variable capture. In this paper we introduce α-paths as an estimation for α-avoidance, roughly expressing that α-conversions are not required to prevent variable capture. These paths provide a novel method to analyse and predict the potential need for α in different calculi. In particular, we show how α-path characterises α-avoidance for several sub-calculi of the λ-calculus like (i) developments, (ii) affine/linear λ-calculi, (iii) the weak λ-calculus, (iv) μ-unfolding and (iv) finally the safe λ-calculus. Furthermore, we study the unavoidability of α-conversions in untyped and simply-typed λ-calculi and prove undecidability of the need of α-conversions for (leftmost–outermost reductions) in the untyped λ-calculus. To ease the work with α-paths, we have implemented the method and the tool is publicly available.

2012 ACM Subject Classification Theory of computation → Program analysis

Keywords and phrases λ-calculus, variable capture, α-conversion, developments, safe λ-calculus, undecidability

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.22

Supplementary Material *Software (Alpha Avoidance Tool)*: https://tcs-informatik.uibk.ac.at/tools/alpha/ archived at swh:1:dir:e2731de499d52916453f9ac5d90cf9db879e26c5

1 Introduction

Substitution is a fundamental concept in computer science. It is, for example, a core operation for computation in the λ-calculus, applied by compilers to optimise programs and, in general, key for reasoning with logical expressions. As is well-known, undesired side effects may arise when substitution and bindings interact, if the substitution is naïvely applied. Here, we study substitution and in particular the need for α-conversion in the context of the λ-calculus. We emphasise, however, that the same idea could be applied in different fields.

In the λ-calculus, the contraction of a redex by means of naïve substitution may cause *variable capture* where a variable originally occurring free ends up being bound in the resulting term due to a name collision. Variable captures may lead to inconsistent results and invalidate the confluence property. Such fallacies have occurred already quite early in the literature, for example in work from the 1940s by Newman [29]. As discovered by Schroer, and as presented by Rosser in his review [33], Newman's proof that the projection axioms were satisfied for the λI-calculus was erroneous. The purported proof contained an α-problem; cf. [30], [29, Remark 6.14(ii)] and [13, Sect. 5.2].[1]

[1] As far as we know this problem is the α-α-problem, that is, this is the first α-problem in the literature on the λ-calculus.

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 22; pp. 22:1–22:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Since the specific variable names are actually irrelevant (cf. [12]), the result of an evaluation should also not be influenced by the specific names. An option is to work with some kind of unique representatives of α-equivalence classes of λ-term, e.g. with De Bruijn's λ-terms with nameless dummies [12] (see below for more). Though that certainly is a possibility, here we stick to Church's original proposal and work with explicit α-conversion steps, enabling to state the main questions addressed in this paper: can α-conversion steps be avoided for a λ-term M, by suitably α-converting it up front, say to a term M' such that no α-conversion step needs to be invoked along any reduction from M'. Such a characterisation should allow for a more efficient reduction, based on naïve substitutions, that applies α-conversion (if at all) only on the initial term. In the sequel, by "avoiding α" we mean that we can α-convert a λ-term M to some λ-term M' so that subsequent α-conversions are not needed in any computation from M'.

Before proceeding let us relate the question addressed here to the so-called Variable Convention [6] stating that variables may be assumed to be suitably renamed apart in *a given context*. On the one hand, this convention has been widely adopted in the literature. On the other hand, examples as in Figure 1, where renaming apart in the initial term does not suffice, abound. From that perspective our investigation addresses the question in *what contexts* exactly does the Variable Convention work?

In the examples presented in Figure 1, α cannot be avoided, no matter how the variables are (re)named initially. Without the explicit α-conversion steps and substituting naïvely, would lead to variable capture and give rise to the λ-terms $\lambda zz.z\,z$ and $\lambda cxy.c\,x\,x$ respectively, which do not have the intended semantics. (Hence omitting the α-conversion steps would break the Church–Rosser property.) Note that though the example on the left in Figure 1 cannot be (simply) typed, the example on the right can, showing that type regimes in general do not guarantee that α can be avoided.

Contributions

- As already indicated in Figure 1, α-conversion may be unavoidable in the (untyped) λ-calculus. This motivates the question about the algorithmic feasibility of the problem. We establish that (for leftmost–outermost reductions) the problem is undecidable.
- We present a *sound* characterisation for α-avoidance via *α-paths*. *A*-paths give a novel perspective on α; they can be utilised as a tool to predict for a given λ-term M the *potential need* for α-conversion, i.e. the need for α-conversion in *any* step $N \to_\beta L$ after *any* β-reduction $M \to_\beta^* N$. To that end, α-paths combine two known ideas.

  Foremost, α-paths build on the notion of *legal* path, cf. [3], characterising the so-called *virtual* redexes of a term M, where a virtual redex of M is a redex that can arise in *any* term N along *any* reduction $M \to_\beta^* N$. Legal paths arose from Girard's geometry of interaction; see [2] for various characterisations of them attesting to their canonicity. The intuition for them employed here, is that a redex R in N is represented in the *graph* of N by a single-edge-*path* from an application node to an abstraction node, and that *pulling that path back* along the reduction $M \to_\beta^* N$ gives rise to a path in M, the legal path representing the redex R in N as a virtual redex in M.

  The intuition then is that α-conversion is *potentially needed* in M when there is a *virtual* redex in M, that is, a redex in N, whose contraction *needs* α-conversion. Since also the need for α-conversion can be characterised by means of paths, namely by so-called *binding-capturing chains* [17, 7], we arrive at our notion of α-path, combining legal paths with binding-capturing chains.
- To ease work with α-paths, we have implemented the method; the tool is publicly available.

$$\begin{array}{ll}
 & \underline{(\lambda x.x\,x)\,(\lambda yz.y\,z)}_{\mathsf{A}} \\
\to_\beta & \underline{(\lambda yz.yz)\,(\lambda yz.y\,z)}_{\mathsf{B}} \\
\to_\beta & \lambda z.(\lambda yz.y\,z)\,z \\
\to_\alpha & \lambda z.\underline{(\lambda yz'.y\,z')\,z}_{\mathsf{C}} \\
\to_\beta & \lambda zz'.z\,z'
\end{array}$$

A...duplication B...redex creation C...open redex contraction

$$\begin{array}{ll}
 & \underline{(\lambda fc.f\,(f\,c))\,(\lambda zxy.z\,y\,x)}_{\mathsf{A}} \\
\to_\beta & \lambda c.(\lambda zxy.z\,y\,x)\,\underline{((\lambda zxy.z\,y\,x)\,c)} \\
\to_\beta & \lambda c.\underline{(\lambda zxy.z\,y\,x)\,(\lambda xy.c\,y\,x)}_{\mathsf{B}} \\
\to_\beta & \lambda c.(\lambda xy.(\lambda xy.c\,y\,x)\,y\,x) \\
\to_\alpha & \lambda c.(\lambda xy.\underline{(\lambda xy'.c\,y'\,x)\,y}_{\mathsf{C}}\,x) \\
\to_\beta & \lambda cxy.\underline{(\lambda y'.c\,y'\,y)\,x} \\
\to_\beta & \lambda cxy.c\,x\,y
\end{array}$$

Figure 1 α is needed in the λ-calculus.

- We exemplify the versatility of α-paths through various important sub-calculi of the λ-calculus, listed below. The first three calculi arise from a careful analysis of the canonical example illustrating why α-conversion is unavoidable in the λ-calculus, the λ-term $(\lambda x.x\,x)\,(\lambda yz.y\,z)$. As illustrated in Figure 1, its reduction to normal form comprises first a *duplicating* step A (the subterm $\lambda yz.y\,z$ is duplicated), then a *creating* step B (creating the redex $(\lambda yz.y\,z)\,z$), and finally a *non-closed* step C (contracting an *open* redex $(\lambda yz.y\,z)\,z$, containing the free variable z bound outside). Somewhat surprisingly, forbidding *one* of these three types of steps suffices for α-avoidance.

1. *Developments* [14] are reductions in which no *created* redexes are contracted. Stated differently, in a development from M only *residuals* of redexes in M (no virtual redexes) are contracted. Intuitively, α can then be avoided since all residuals of a given redex are disjoint along a development.
   Developments [14, 6] feature prominently in the λ-calculus since its inception. They form the basis for the proof that β-reduction has the Church–Rosser property, more precisely, that parallel β-reduction has the diamond property and satisfies the cube law, using that all developments are finite (unlike arbitrary reductions).
2. The *linear* (*affine*) λ-calculus [21] forbids *duplication*. That is achieved formally by restricting term-formation, requiring the variable x to occur free exactly (at most) once in M in an abstraction term $\lambda x.M$. Intuitively, α can then be avoided since variables persist linearly along reductions.
   Though the linear λ-calculus [21, 23, 27, 38] had been studied before, it rose to prominence after the introduction of linear logic, via the connection between linear λ-terms and MLL-proofnets, with abstractions and applications corresponding to pars and tensors. Linearity affords nice properties, e.g. termination and simple typability.
3. The *weak* λ-calculus [39] forbids to contract *open* redexes. Intuitively, α is then avoided since when substituting by closed terms only, there's no risk of variable capture either.
   Weak reduction [37, 31, 1, 39, 8, 5] forms the basis of functional programming languages such as Haskell that evaluate to weak head normal form. Indeed, the fact that α-conversion can then be avoided was stated as an explicit motivation in [31], and makes that functional programs can be represented as orthogonal term rewrite systems and weak reduction can be optimally implemented via Wadsworth's graph reduction. (Weak reduction is often paraphrased as "no reduction under a λ", but that restriction is undesirable as it breaks the Church–Rosser property.)

These three examples are mainly methodological, since the fact *that* α can be avoided for them is well-known. We also present two important but less well-known examples.

4. The *modal μ-calculus* [25] has unfolding rules for least (μ) and greatest (ν) fixed-points in its formulas. Intuitively, α can then be avoided for the same reason it can for developments, namely that unfolding does not *create* new redexes [17, 7]. Here we show

that this can be obtained via α-paths, under a standard embedding of modal μ-formulas in the λ-calculus, representing unfolding using Turing's fixed-point combinator.

Though the literature on the modal μ-calculus is rich, only recently issues related to α-conversion seem to have been addressed [26]. The main point of this example is to suggest that the technology developed here for avoiding α in the λ-calculus, can be transferred to other calculi with binding, in mathematics, logic, programming languages, linguistics, music theory, etc..

5. In the *safe λ-calculus* [10, 9, 11] the occurrences of (free) variables are restricted according to the order of their type. Intuitively, this restriction on the order of the types of the variables can be transferred to the variables, guaranteeing that α can be avoided. (Note that as observed above, typing disciplines, say simple typing, in general do *not* suffice to be able to avoid α.)

Analysing the safe λ-calculus as presented in [9, 11] using our tools, we found that the claim that α could be avoided in it, was not entirely correct, provoking the further analysis, and a proposal for slight amendations, presented below.

Related Work

In the classical literature on the λ-calculus the focus was not on α-conversion. However, when the λ-calculus started being used as a tool, α-conversion had to be addressed. We briefly discuss two important strands of research. One approach is to abstract α away and to exclusively work with (representatives of) α-equivalence classes of λ-terms.[2] De Bruijn's lambda notation with nameless dummies [12] is widely adopted in implementations. This typically side-steps the issue but does not resolve it: the cost of α is now inextricably hidden in the cost of β, and α-conversion disappears in the notation with nameless dummies only to resurface in the form of reindexing. Moreover, any such representation runs the risk of creating a gap between the theory in the literature and the representation.[3] Another approach is to bring α-conversion about in another way. The *nominal* approach [19] is a prominent exponent of this, recasting the notion of a variable being bound via the dual notion of a variable being free for, allowing to recast α-conversion via the classical notion of *permutation*. We stress that α-conversion resurfaces in this setting, but unlike the modulo-approach now in an explicit form as in our case, making it interesting to study our question for it (and then compare both). We leave that to further research.

Finally, we mention that several other decision problems about α have been considered by Statman, which were reported in [35]. This work is based on [18].

Outline

This paper is structured as follows. In the next section, we recall fundamental concepts and notions. In Section 3, we motivate the definition of α-paths and provide a syntactic proof that developments can avoid α by using of a restricted form of α-paths. The latter are generalised in Section 4, where we establish the main contribution of this work, a sound characterisation of α-avoidance via α-paths. Section 5 applies this characterisation to affine, weak and the safe λ-calculus. Finally, we conclude in Section 6.

[2] Higher-Order Abstract Syntax goes one (big) step further by working with simply typed $\alpha\beta\eta$-equivalence classes of terms.

[3] The same holds for programming; everyone will have encountered inscrutable error-messages on De Bruijn-indices representing variables.

Table 1 Capture-avoiding and capture-permitting substitution.

M	$[\![x := N]\!]$ (capture-avoiding)	$[x := N]$ (capture-permitting)
x	N	N
y	y	y
$e_1\,e_2$	$e_1[\![x := N]\!]\,e_2[\![x := N]\!]$	$e_1[x := N]\,e_2[x := N]$
$\lambda x.e$	$\lambda x.e$	$\lambda x.e$
$\lambda y.e$	$\lambda y.e[\![x := N]\!]$ if $y \notin \mathcal{FV}(N)$ $\lambda z.e[\![y := z]\!][\![x := N]\!]$ else with z fresh for e and N.	$\lambda y.e[x := N]$

2 Preliminaries

We assume acquaintance with the standard definitions of the λ-calculus, cf. [6], but recall relevant concepts and notations. We use $=$ to denote syntactic equality of λ-terms, and $\equiv_\alpha$ for equality modulo α. We write $\mathcal{FV}(M)$ for the set of free variables in a λ-term M and $\mathcal{BV}(M)$ for the set of bound variables. We distinguish between a capture-*avoiding* and a capture-*permitting* substitution, cf. Table 1. The capture-avoiding substitution, denoted as $M[\![x := N]\!]$, deals with a potential variable capture, whereas the capture-permitting substitution, denoted as $M[x := N]$, naïvely substitutes. If $M[\![x := N]\!] \equiv_\alpha M[x := N]$ then we say that the substitution of N for x in M is *α-free*. The single-step β-reduction contracting a redex $(\lambda x.M)\,N$ in some arbitrary context, is said to be *α-free*, if the applied substitution is α-free.

▶ **Definition 1.** *A reduction sequence starting from a λ-term M is said to be* α-free*, if each β-reduction step is α-free. A λ-term M has α-free simulations, if there exists an α-equivalent λ-term N such that every reduction sequence starting from N is α-free. In such case we say that N* avoids α*. We say that we can avoid α in a calculus, if every term in this calculus has* α-free simulations*.*

The reduction sequence illustrated in Figure 1 is not α-free. The λ-term $(\lambda x.x\,x)\,(\lambda yz.y\,z)$ does not have α-free simulations, which shows that α cannot be avoided in the pure λ-calculus. The λ-term $(\lambda fx.f\,(f\,x))\,(\lambda fx.f\,(f\,(f\,x)))$, denoting the exponentiation 3^2 via Church numerals, has α-free simulations as the α-equivalent λ-term $(\lambda fy.f\,(f\,y))\,(\lambda fx.f\,(f\,(f\,x)))$ avoids α. (This can also be checked with our tool, see Listing 1 in Section 4 below).

The *position* in a λ-term is a finite sequence of 1s and 2s. The set of positions of a λ-term M is denoted as $\mathcal{P}os(M)$. We write $M|_p$ for the subterm at position p in M and $M(p)$ for the *symbol* at position p (the head-symbol of $M|_p$), where $M(p) \in \{x, @, \lambda x\}$ for some x. In the following we may write s^p when we depict a specific symbol s of a λ-term M at position p, $s = M(p)$, whenever both the position and the symbol are of interest.

$$M|_p := \begin{cases} M & \text{if } p = \epsilon \\ N|_{p'} & \text{if } M = \lambda x.N \text{ and } p = p'1 \\ N_i|_{p'} & \text{if } M = N_1\,N_2 \text{ and } p = p'i \end{cases}$$

A position p is a *prefix* of a position q, if $q = pq'$ for some position q'. We use the notation $p \preceq q$ to denote that p is a prefix of q and $p \prec q$ to denote that p is a *strict* prefix of q (q' is non-empty). Two positions p, q are said to be *parallel*, denoted by $p \parallel q$, if $p \npreceq q \land q \npreceq p$. A position p is said to be *left* of a position q, written as $p \parallel_l q$, if $p = s \cdot 1 \cdot p'$ and $q = s \cdot 2 \cdot q'$. We define the trace relation ▶ to be the relation between positions in the source and in the target of a β-step $s \to_\beta t$ contracting a redex at position o (cf. [36, Section 8.6.1]):

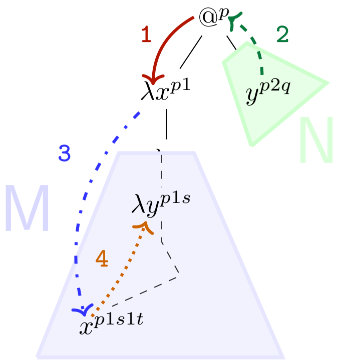


Figure 2 Substitution dynamics leading to variable capture.

- (context) $p \blacktriangleright p$ if o is not prefix of p,
- (body) $o11p \blacktriangleright op$ if $p \neq \epsilon$ and $p \neq q$,
- (arg) $o2p \blacktriangleright oqp$ for all positions q, such that $o11q$ is bound by $o1$.

A redex in a term t at position q is called a *residual* of a redex in some origin s ($s \to_\beta \ldots \to_\beta t$), if $p \blacktriangleright \ldots \blacktriangleright q$ and $s|_p$ is a redex (cf. [15, Chapter 4, Section 4]).

A path $\sigma = (p_1, p_2, \ldots, p_n)$ in a λ-term M is a sequence of positions in $\mathcal{P}os(M)$. The length $|\sigma|$ of a path σ is the number of positions minus 1. An *edge* is a path of length 1.

The reversal of a path σ is denoted by $(\sigma)^r$. Two paths $\sigma = (p_1, p_2, \ldots, p_n)$ and $\psi = (q_1, q_2, \ldots, q_n)$ are said to be *composable*, if $p_n = q_1$. We write $\sigma \cdot \psi$ to denote the composition of two (composable) paths σ, ψ resulting in $(p_1, p_2, \ldots, p_n, q_2, \ldots, q_n)$.

A path in M starting at position p and ending at position q is of type:

1) `@-v`, if $M(p) = @$ and $M(q) = x$ for some x.
2) `@-`λ, if $M(p) = @$ and $M(q) = \lambda x$ for some x.
3) `@-@`, if $M(p) = @$ and $M(q) = @$.
4) `v-v`, if $M(p) = x$ and $M(q) = y$ for some x, y.
5) `v-`λ, if $M(p) = x$ and $M(q) = \lambda y$ for some x, y.

To illustrate, let $M = (\lambda x.x\,x)\,(\lambda yz.y\,z)$. $\sigma = (\epsilon, 2, 2112)$ is a `@-v`-path in M with $|\sigma| = 2$. σ and $(\sigma)^r$ are composable and the path $(\epsilon, 2, 2112, 2, \epsilon)$ resulting from their composition $\sigma \cdot (\sigma)^r$ is of type `@-@`.

3 Developments Are α-Avoiding

Recall that reductions of residuals, also known as *developments*, are finite. This was proved already in 1936 by Church–Rosser for the λI-calculus [14] and then generalised to the full λ-calculus by Schroer [34] and independently by Hindley [20]. It is well known that in *finite developments* α-renaming can be avoided, cf. [24]. Intuitively, this is due to the fact that in developments the residuals of a redex-pattern remain disjoint [22]. Thus, if all binders are initially properly renamed apart, α can be avoided. To prepare the ground for our main contribution – α-paths – we sketch a purely syntactic proof of this result in this section.

We start by giving an intuition for how a capturing-potential can be characterised by paths. A naïve substitution leads to a variable capture whenever we

(i) naïvely contract a redex $(\lambda x.M)\ N$ where
(ii) some variable y occurring free in N
(iii) is moved into M, where some x is free in M
(iv) is in the scope of a λy.

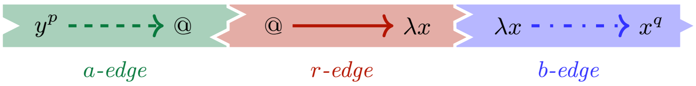
Each of these conditions can be represented via edges in the abstract syntax tree (AST), as formalised below and illustrated in Figure 2 for the redex $(\lambda x.M)\,N$. More precisely, we have an a-edge $(p2q, p)$, an r-edge $(p, p1)$, a b-edge $(p1, p1s1t)$ and a c-edge $(p1s1t, p1s)$.

Let M be a λ-term. We conceive the AST of M as a graph and define four additional types of *edges* for M:

1. (r-edge $\longrightarrow$) A *r*edex-edge $(p, p1)$ connects an @-node at position p to its left son at position $p1$, if $M(p1) = \lambda x$ for some x.
2. (a-edge $\dashrightarrow$) An *a*pplication-edge $(p2q, p)$ connects a variable x at position $p2q$ to an @-node at position p, if x is free in $M|_{p2}$.
3. (b-edge $\dashrightarrow$) A *b*inding-edge $(p, p1q)$ connects a λx at position p to a variable y at position $p1q$, if $x = y$ and y is free in $M|_{p1}$.
4. (c-edge $\cdots\rightarrow$) A *c*apturing-edge $(p1q, p)$ connects a variable y at position $p1q$ to a λx at position p to, if $x \neq y$ and y is free in $M|_{p1}$.

We add the a-, r-, b- and c-edges as actual edges to the graph of M in the standard way. We call such a graph the *α-graph* of a λ-term M, denoted as $G_\alpha(M)$. From the definition of an r-edge, we immediately obtain that for any r-edge in $G_\alpha(M)$ with the source at position p, $M|_p$ is a redex.

▶ **Definition 2.** *Let M be a λ-term, a an* a*-edge, r an* r*-edge and b a* b*-edge in $G_\alpha(M)$ with a, r and r, b composable. We call the* v*–*v*-path $\sigma_{arb} = a \cdot r \cdot b$ an* arb-path *of M.*

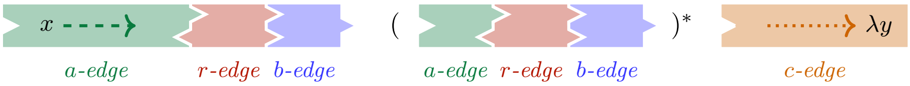


Let p be the position of the starting v-node y and q the position of the ending v-node of an *arb*-path φ. Then we have $q \parallel_l p$.

The example term from Figure 3 illustrates an example where an outermost reduction strategy needs α in the second reduction step. To characterise the need for α after the contraction of one or multiple redexes, *arbic*-paths are introduced next.

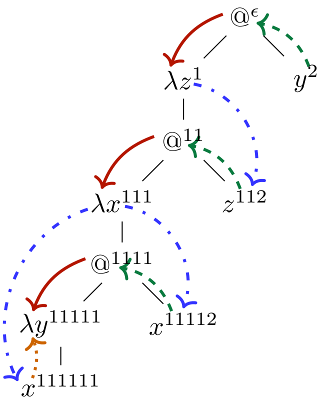
▶ **Definition 3.** *The set of* arbic*-paths of a λ-term M is inductively defined as follows.*

- *(base case) Let σ_{arb} be an* arb*-path of M and c a* c*-edge in $G_\alpha(M)$ with σ_{arb}, c composable. Then the* v*–λ–path $\sigma_{arb} \cdot c$ is an* arbic*-path of M.*
- *(*arb*-composition) Let σ_{arb} be an* arb*-path and ψ an* arbic*-path of M with σ_{arb}, ψ composable. Then the* v*–λ–path $\sigma_{arb} \cdot \psi$ is an* arbic*-path of M.*



From Definition 3 we see that *arbic*-paths are non-empty sequences of *arb*-paths followed by a c-edge ($\sigma_{arb}^{+} \cdot c$). As already remarked, an *arb*-path connects the occurrence of a variable to the occurrence of another variable at its left. By consequence arbic-paths are acyclic and the set of *arbic*-paths of a λ-term M is finite. The paths $\sigma_0 = 112 \to 11 \to 111 \to 111111 \to 11111$ and $\sigma_1 = 2 \to \epsilon \to 1 \to \sigma_0$ are *arbic*-paths for the λ-term illustrated in Figure 3. Specialising *arbic*-paths such that the names of the initial variable and of the final abstraction are equal, we obtain a characterisation of the need for α in some reduction sequence.

▶ **Definition 4** (*arbic α-path*)**.** *Let M be a λ-term and ψ an* arbic*-path of M. If ψ starts with a variable x and ends with a λ-node λy where $x = y$, then ψ is called an arbic α-path.*



leftmost–outermost reduction

$$\begin{array}{ll} & \underline{(\lambda z.(\lambda x.(\lambda y.x)\,x)\,z)\,y} \\ \to_\beta & \underline{(\lambda x.(\lambda y.x)\,x)\,y} \\ \to_\beta & \underline{(\lambda y'.y)\,x} \\ \to_\beta & y \end{array}$$

leftmost–innermost reduction

$$\begin{array}{ll} & (\lambda z.(\lambda x.\underline{(\lambda y.x)\,x})\,z)\,y \\ \to_\beta & (\lambda z.\underline{(\lambda x.x)\,z})\,y \\ \to_\beta & \underline{(\lambda z.z)\,y} \\ \to_\beta & y \end{array}$$

Figure 3 Leftmost–outermost needs α.

The path σ_1 as defined above is an arbic α-path for the λ-term illustrated in Figure 3. Now, essentially by construction, we can see that if there is no arbic α-path in $G_\alpha((\lambda x.M)\,N)$ starting at a free variable in N and ending in M, then $M[\![x := N]\!] \equiv_\alpha M[x := N]$. We emphasise, that we only claim α-equivalence and not syntactic equivalence ($=$) of $M[\![x := N]\!]$ and $M[x := N]$. To clarify, let $(\lambda x.M)\,N$ be a redex with $M = \lambda y.y$ and $N = y$. Then $M[\![x := N]\!] = \lambda z.z$ and $M[x := N] = \lambda y.y$. We have $M[\![x := N]\!] \equiv_\alpha M[x := N]$, but $M[\![x := N]\!] \neq M[x := N]$. Hence, α-equivalence is the strongest property that we can conclude.[4]

▶ **Lemma 5.** *Let $s \to_\beta t$. If $G_\alpha(s)$ contains no arbic α-path, then $G_\alpha(t)$, where the set of* r*-edges is restricted to those denoting residuals, also does not.*

Proof. We write $\langle G_\alpha(t)\rangle$ for the sub-graph of $G_\alpha(t)$, where the set of r-edges is restricted to those denoting residuals of s. Since there are no arbic α-paths in $G_\alpha(s)$, the β-step can be performed by means of capture-permitting substitution (no variable capture). We have $s = C[(\lambda x.M)\,N]$ and $t \equiv_\alpha C[M[x := N]]$ for some context C, body M and argument N, with $(\lambda x.M)\,N$ being the contracted redex at position o. We prove the lemma by relating the edges in $\langle G_\alpha(t)\rangle$ to edges and paths in $G_\alpha(s)$ and making a distinction according to the components as they appear in the source and the target. As done in [17], we use primed variables (p', q') to range over positions in the target term t, indicating the positions they trace back to in the source term s, by unpriming (p, q).

Consider an a- or a c-edge from p' to q' in $\langle G_\alpha(t)\rangle$ where p' denotes the position of a variable y and q' the position of an application (in the case of an a-edge) or an abstraction (in the case of a c-edge). We have $q' \prec p'$ and the variable y at $t(p')$ occurs free in $t|_{q'}$. We distinguish the following cases:

- p', q' in the same component: we have the same edge from p to q in $G_\alpha(s)$.
- q' in the context and p' in the body: then $x \neq y$ (otherwise the y would have been replaced by N) and we have the same edge in $G_\alpha(s)$ with 11 inserted at o.
- q' in the context and p' in the argument: there is no variable capture so $s(p)$ must occur free in $s|_q$. Therefore, we have the same edge from p to q in $G_\alpha(s)$.
- q' in the body and p' in the argument: the origin of the a-edge/c-edge is an arb-path from p to qq_1, for some q_1, followed by an a-edge/c-edge from qq_1 to q in $G_\alpha(s)$.

[4] We stick to the standard definition of substitution $M[\![x := N]\!]$, which renames even if the variable x to be replaced does not occur in the body M [6]. We note that, if we were to adapt the substitution so that it is not applied when the argument is erased ($x \notin \mathcal{FV}(M)$), then we could claim syntactic equivalence.

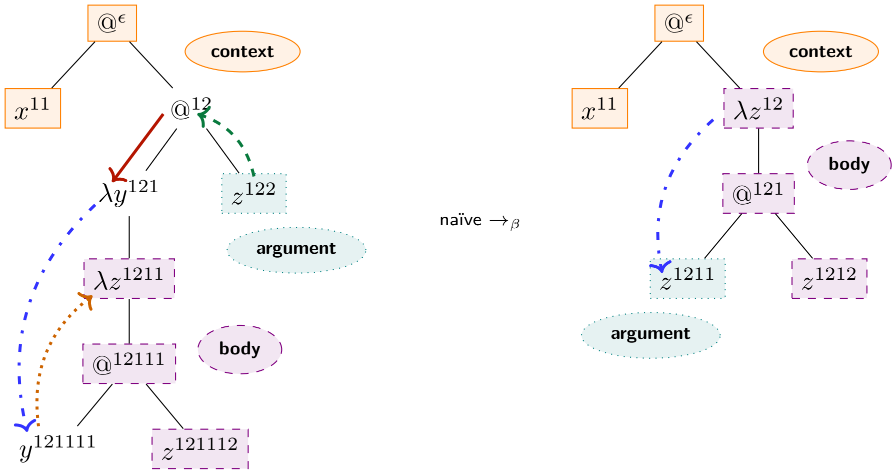


Figure 4 A b-edge that traces back to an arbic α-path.

Given a b-edge from q' to p' in $\langle G_\alpha(t)\rangle$. p' denotes the position of the bound variable y, q' the position of the binder λy. We have $q' \prec p'$ and distinguish following cases:

- p', q' in the same component: then we have a b-edge from q to p in $G_\alpha(s)$.
- q' in the context and p' in the body: we have $x \neq y$ and a b-edge in $G_\alpha(s)$ with 11 inserted at o.
- q' in the context and p' in the argument: there is no variable capture so $s(p)$ must occur free in $s|_q$. Therefore, we have a b-edge from q to p in $G_\alpha(s)$.
- q' in the body and p' in the argument: such a b-edge would map back to an arbic α-path from p to q in $G_\alpha(s)$, which is excluded by the assumption (Figure 4 illustrates an example).

For the r-edges $(p', p'1)$ in $\langle G_\alpha(t)\rangle$ we make the following case distinction:

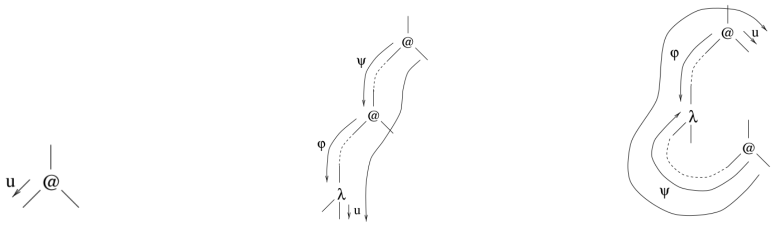
- p' and q' are in the same component: then we have an r-edge from p to q in $G_\alpha(s)$.
- in all other cases: such an r-edge would denote a created redex in t. We have no such r-edge in $\langle G_\alpha(t)\rangle$.

We have seen that a r-edges and b-edges in $\langle G_\alpha(t)\rangle$ map back to an edge of the same type in $G_\alpha(s)$. a-edges and c-edges map back to a path of shape $\sigma^*_{arb} \cdot e$, where e denotes an edge of the same type and σ_{arb} an arb-path in $G_\alpha(s)$. An arbic α-path in $G_\alpha(t)$ has the following shape $(a'_1, r'_1, b'_1, \ldots, a'_n, r'_n, b'_n, c')$, where x_i denotes an x-edge (p_i, q_i). If we replace the edges in this path by the edges and paths they map back to, we get a path of the shape $(\sigma^*_{arb_1} \cdot a_1, r_1, b_1, \ldots, \sigma^*_{arb_n} \cdot a_n, r_n, b_n, \sigma^*_{arb} \cdot c)$, which would be an arbic α-path in $G_\alpha(s)$. ◀

Based on the lemma, we obtain the characterisation of α-freeness via arbic α-paths. Let M be a λ-term. If M contains no arbic α-path, then every development from M is α-free. Arbic α-paths can also witness to the capture-potential for the term shown in Figure 3, where α is needed in the second reduction step. Note that with these arbic α-paths we do not yet characterise variable captures that result from the contraction of *created* redexes. This we will take up in Section 4 below, where we make use of *legal* paths, cf. [3].

In sum, α-paths allow us to reprove the well-known result that in finite developments α-conversions potentially only need to be performed on the initial term (and are thus cheap).

▶ **Theorem 6.** *In finite developments α can be avoided.*



(a) Base case. **(b)** @-composition. **(c)** λ-composition.

Figure 5 Well-balanced paths [3].

Proof. Let M be a λ-term. By the above, if M contains no arbic α-path, then every development from M is α-free. Thus, it remains to observe that for every λ-term M there exists a λ-term N where $M \equiv_\alpha N$, such that N does not contain any arbic α-paths. The latter follows as all binders in M can trivially be renamed apart. ◀

This result is not new, as noted above, but illustrates how α-paths give a new perspective on this problem and therefore offer a different way to reason about α.

4 α-Paths – A Sound Characterisation For α

In this section, we generalise arbic α-paths so that the thus obtained α-paths reflect the conditions that necessitate the application of α. For that we also have to characterise the need for α that may arise for created redexes. A (sub)term, which is not a redex yet, but might become one along reduction, is called a *virtual* redex, which in turn is characterised by *legal paths*, cf. [3].

Legal Paths

In the following, to keep this paper self-contained, we briefly recall the formal definition of legal paths as established in [3]. For motivation and underlying intuitions, we kindly refer the reader to [3] and to [4], where the legal paths have been introduced. *Legal* paths start at an @-node and connect via a path the @-node with all the subterms with which it can interact in some reduction sequence. Legal paths ending at a λ-node therefore characterise a virtual redex. Legal paths are defined via the *well-balanced* paths.

The set of *well-balanced* paths (abbreviated as *wbp*) of a term M is inductively defined on $G_\alpha(M)$ as described in the following and illustrated in Figure 5.

- (base case) The path $(p, p1)$ with $M(p) = @$ is a wbp.
- (*@-composition*) let ψ, φ be two composable wbps of type `@-@` and `@-`λ, respectively. Then $\psi \cdot \varphi \cdot u$ is a wbp, where $u = (p, p1)$ with p the position of the final abstraction of φ.
- (λ-composition) Let $\varphi = (p, \ldots, p_n)$ a wbp of type `@-`λ and $\psi = \sigma_a \cdot (\sigma_b)^r$ with σ_a a wbp of type `@-v` ending at position q and $\sigma_b = (p_n, q)$ a b-edge in $G_\alpha(M)$. Then $\psi \cdot (\varphi)^r \cdot u$, where $u = (p, p2)$, is a wbp.

Legal paths impose a legality constraint on the well-balanced paths, restricting the call and return paths of *cycles*. Next, we recall the definition of a cycle. Let φ be a wbp. A subpath ψ of φ is an *elementary* @-cycle of ψ (over an @-node) when (i) it starts and ends with the argument edge of the @-node and (ii) is internal to the argument N of the application corresponding to the @-node (i.e., does not traverse any variable that occurs free in N). The set of *@-cycles* of φ (over an @-node) and of the *v-cycles* of φ (over the occurrence v of a variable) is defined inductively, as follows: (i) every elementary @-cycle of ζ is an @-cycle; (ii) *$\boldsymbol{v}$-cycle*: every cyclic subpath of ζ of the form $(v)^r \cdot (\phi)^r \cdot \psi \cdot \phi \cdot v$, where $\phi = (p_2, \ldots, q_n)$ is a wbp, ψ is an @-cycle and $v = (p_1, p_2)$ a b-edge, is an $\mathtt{v}$-cycle; (iii) *@-cycle*: every subpath ψ of ζ that starts and ends with the argument edge of a given @-node, and that is composed of subpaths internal to the argument N of @- and $\mathtt{v}$-cycles over free variables of N is an @-cycle (over the @-node). As stated by the following proposition @-cycles are always surrounded by two wbps of type $\texttt{@}$-λ, cf. [3].

▶ **Proposition 7** ([3, Corollary 6.2.26]). *Let ψ be an @-cycle of ϕ over an @-node. The wbp ϕ can be uniquely decomposed as: $\phi = \zeta_1\ \lambda\ (\zeta_2)^r\ @\ \psi\ @\ \zeta_3\ \lambda\ \zeta_4$,[5] where ζ_2 (call-path) and ζ_3 (return-path) are wbps of type $\texttt{@}$-λ.*

Considering the statement of the proposition, the last label of ζ_1 and the first label of ζ_4 are called *discriminants*. Finally, the *legality* constraint ensures that the *call*- and the *return*-path of such cycles coincide.

▶ **Definition 8** ([3, Definition 6.2.27]). *A wbp is a* legal *path if the call and return paths of any @-cycle are one the reverse of the other and their discriminants are equal.*

▶ **Proposition 9** ([3, Section 6.2.5]). *For all (virtual) redexes of a λ-term M there is a legal path of type $\texttt{@}$-λ in M.*

It follows that for any (created) redex along a reduction sequence starting from a λ-term M, we have a legal path in M characterising the redex. This path also encodes the reduction sequence that leads to its creation, if it is not already a redex in M.

Characterisation of α-avoidance via α-paths

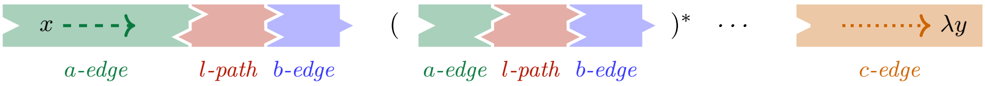
In Section 3, we have seen how arbic α-paths characterise the need for α for developments with no redex creation. The α-paths presented in this section are an extension of them and allow to characterise the need for α in λ-calculi with redex creation. α-paths are defined on the so-called albic-paths that rely on *legal* paths.[6] First, we define *alb*-paths.

▶ **Definition 10.** *Let M be λ-term, a an* a-edge*, l a legal path and b a* b-edge *in $G_\alpha(M)$ with a, l and l, b composable and b, a not composable. We call the $\mathtt{v}$–$\mathtt{v}$-path $\sigma_{alb} = a \cdot l \cdot b$ an* alb-path *of M.*

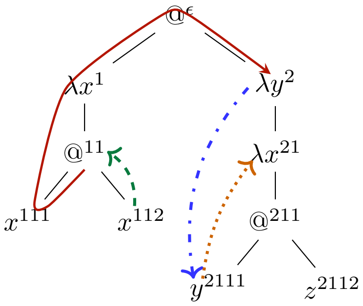
Second, essentially iterating *alb*-paths, we obtain the definition of *albic*-paths. Note that each *arbic*-path is also an *albic*-path, as each r-edge constitutes a legal path.

▶ **Definition 11.** *The set of* albic-paths *of M is inductively defined:*

- *(*base case*) let σ_{alb} be an* alb*-path and c a* c*-edge with σ_{alb}, c composable; Then the $\mathtt{v}$-λ–path $\sigma_{alb} \cdot c$ is an* albic-path.
- *(*alb-composition*) let σ_{alb} be an* alb*-path and ψ an* albic-path *with σ_{alb}, ψ composable. Then the $\mathtt{v}$-λ–path $\sigma_{alb} \cdot \psi$ is an* albic-path.



[5] We use the λ- and @-symbol to point out the start- and end-nodes of the different wbps.
[6] We call them *albic*, or $(alb)^i c$, because they consist of i (with $i \geq 1$) sequences of *alb*-paths and a final *c*-edge.



$$\begin{aligned} & \underline{(\lambda x.x\,x)\,(\lambda yx.y\,z)} \\ \to_\beta\; & \underline{(\lambda yx.y\,z)\,(\lambda yx.y\,z)} \\ \to_\beta\; & \lambda x.\underline{(\lambda yx.y\,z)\,z} \\ \to_\beta\; & \lambda x.(\lambda x.z\,z) \end{aligned}$$

$(\lambda x.x\,x)\,(\lambda yx.y\,z)$ is α-free

Figure 6 α-paths overapproximate the need for α.

Finally, based on Definitions 10 and 11 we can define *α-paths*.

▶ **Definition 12** (α-path)**.** *Let ψ be an albic-path of λ-term M. If ψ starts at a variable x and ends at a λ-node λy, where $x = y$, then the v–λ–path ψ is called an α-*path*.*

Inductively, we can conclude that the absence of α-paths implies α-avoidance.

▶ **Lemma 13** (α-free)**.** *Suppose that there is no α-path in $G_\alpha((\lambda x.M)\,N)$ starting at a free variable in N and ending in M. Then $M[\![x := N]\!] \equiv_\alpha M[x := N]$.*

Proof. If there is no α-path, then by Definition 12 there is no albic α-path hence also no arbic α-path, as observed above. From this we conclude $M[\![x := N]\!] \equiv_\alpha M[x := N]$ by using the observation below Definition 4. ◀

Further, α-path freeness is preserved by β-reduction.

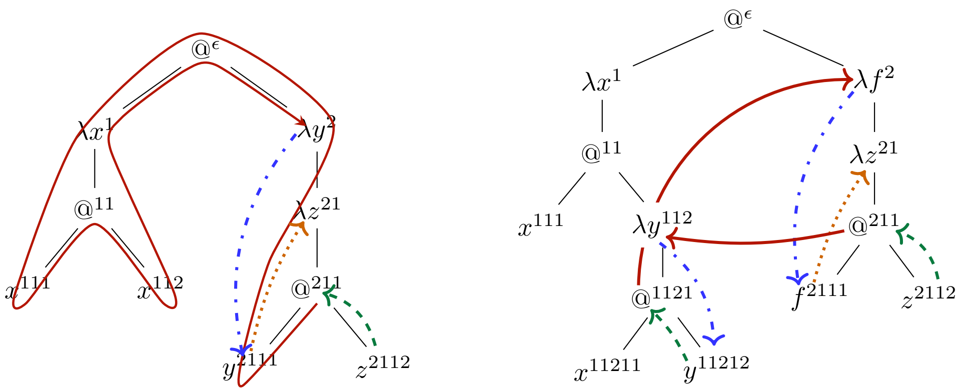
▶ **Lemma 14** (β-invariance)**.** *$\to_\beta$ preserves α-path-freeness.*

Proof. The proof proceeds the same way as the proof of Lemma 5. We restrict ourselves to the most interesting parts here. Again, we use primed variables (p', q') to range over positions in the target term t, indicating the positions they trace back to in the source term s, by unpriming (p, q). Let $s \to_\beta t$. r-edges and b-edges in $\langle G_\alpha(t)\rangle$ map back to an edge of the same type in $G_\alpha(s)$. a-edges and c-edges map back to a path of shape $\sigma^*_{alb} \cdot e$, where e denotes an edge of the same type and σ_{alb} an alb-path in $G_\alpha(s)$. An α-path in t has the following shape $(a'_1, l'_1, b'_1, \ldots, a'_n, l'_n, b'_n, c')$, where x_i denotes an x-edge/legal path from p_i to q_i. If we replace a-edges and c-edges by the path the map back to we get $(\sigma^*_{alb_1} \cdot a_1, l_1, b_1, \ldots, \sigma^*_{alb_n} \cdot a_n, l_n, b_n, \sigma^*_{alb} \cdot c)$, where $\sigma^*_{alb_i} \cdot x_i$ in s connects the same positions as the corresponding x-edge in t. It follows that if we have an α-path in t, then we have an α-path in s. ◀

▶ **Theorem 15.** *Let M be a λ-term. If M contains no α-path, then M avoids α.*

Proof. Assume M contains no α-path. Due to Lemma 14, α-path freeness is preserved by β-reduction. Then it follows by Lemma 13 that capture-permitting substitutions can be employed in place of capture-avoiding ones. Thus M avoids α. ◀

Not every α-path is problematic in the sense that it characterises a variable capture. An α-path may predict name collisions that will never occur if the starting variable gets substituted before the characterised redex will be contracted. This is the case for the term depicted in Figure 6. The α-path $112 \to 11 \to 111 \to 1 \to \epsilon \to 2 \to 2111 \to 21$ is harmless, as



(a) $G_\alpha((\lambda x.x\,x)\,(\lambda yz.y\,z))$.

(b) $G_\alpha((\lambda x.x\,(\lambda y.x\,y))\,(\lambda fz.f\,z))$.

Figure 7 Unremovable α-paths.

the variable x^{112} gets substituted by the argument $\lambda yx.y\ z$ before the redex characterised by the legal path $11 \to 111 \to 1 \to \epsilon \to 2$ gets contracted. Thus, α-paths overapproximate the need of α. This overapproximation is sufficiently accurate to still allow interesting statements about different calculi, since α-avoidance is mainly about *unremovable* α-paths.

An α-path is called *unremovable*, if it starts at a variable occurrence at position $p1q$ and ends at its binder at position p ($p \prec p1q$). In Theorem 6 we employed that we can get rid of arbic α-paths by naming all binders appropriately. This is possible because the starting and the ending position of these paths are always parallel. For unremovable α-paths this is not always the case, as illustrated by the λ-terms in Figures 7a and 7b. Note that Figure 7b illustrates that an unremovable α-path does not necessarily have to contain legal paths from a position p to a position q with $q \prec p$.

▶ **Lemma 16.** *For every λ-term M containing no unremovable α-paths, there exists a λ-term N where $M \equiv_\alpha N$, such that N does not contain any α-paths.*

Undecidability

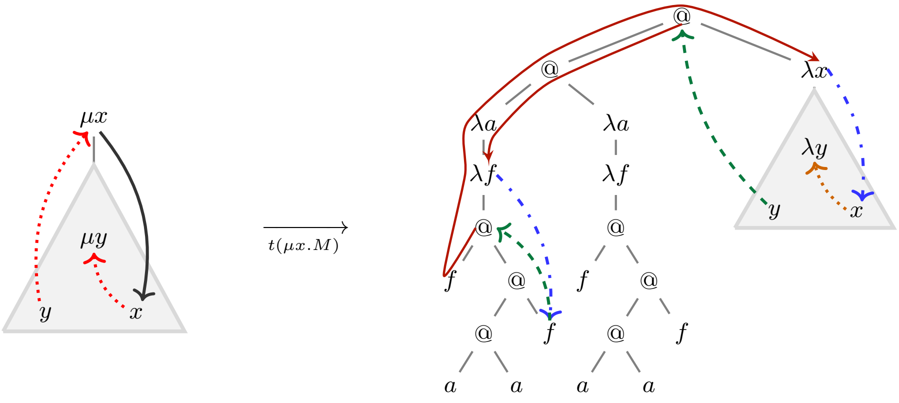
Arbitrary λ-terms may have an unbounded set of legal paths, all of them characterising a different virtual redex. For such terms, making a prediction about the need for α via α-paths is not feasible. This problem is even undecidable for leftmost–outermost reductions, as established by our next result.

▶ **Theorem 17.** *α-avoidance is undecidable for the leftmost–outermost reduction strategy.*

Proof. In proof, we employ a reduction from Post's correspondence problem (PCP short), whose undecidability is well-known [32]. Recall that PCP asks whether for an arbitrary finite set of string pairs $\langle s_1, s_1'\rangle, \langle s_2, s_2'\rangle, \dots, \langle s_n, s_n'\rangle$ over the alphabet $\{\mathsf{a}, \mathsf{b}\}$, there exists indices $i_j \in \{1, 2, \dots, n\}$ such that

$$s_{i_1} s_{i_2} \dots s_{i_k} = s'_{i_1} s'_{i_2} \dots s'_{i_k} \qquad k \geqslant 1\ .$$

It is not difficult to define λ-terms for (i) strings aa, bb, namely $AA := \lambda abx.a\,(a\,x)$ and $BB := \lambda abx.b\,(b\,x)$, respectively; (ii) *conditionals* (denoted as *ITE*); (iii) *pairs* (*PAIRS*) and (iv) in particular PCP (*PCP*), such that the λ-term *PCP* takes an (encoding) of list of pairs



(a) A self-capturing chain in $\mu.M$.

(b) An α-path in $t(\mu.M)$.

Figure 8 A self-capturing chain in μ is an α-path in Λ_μ.

as input and recursively combines them, until a solution is produced (if it exists at all). As the leftmost–outermost strategy is normalising for the λ-calculus, this solution can be found by this strategy. Now, consider the following program

$$(ITE\,(PCP\,PAIRS)\,AA\,BB)\,(\lambda xyz.(x\,z)\,y)\ ,$$

ITE, PCP, $PAIRS$, AA and BB are defined as above. As $ITE\,(PCP\,PAIRS)\,AA\,BB$ is typable, α can be avoided in its reduction, cf. Section 5.3 or [31, Section 11.3.2]. If the problem has a solution, it will reduce to the λ-term $AA\,(\lambda xyz.(x\,z)\,y)$, where α is unavoidable. Otherwise, it will reduce to the λ-term $BB\,(\lambda xyz.(x\,z)\,y)$, from which we get with one β-step to $\lambda bx.b\,(b\,x)$. Moreover, as mentioned the reduction sequence to these terms is α-free. Thus, if we further reduce these terms to normal form, then we need α iff the PCP problem has a solution. Thus, we conclude the theorem. ◀

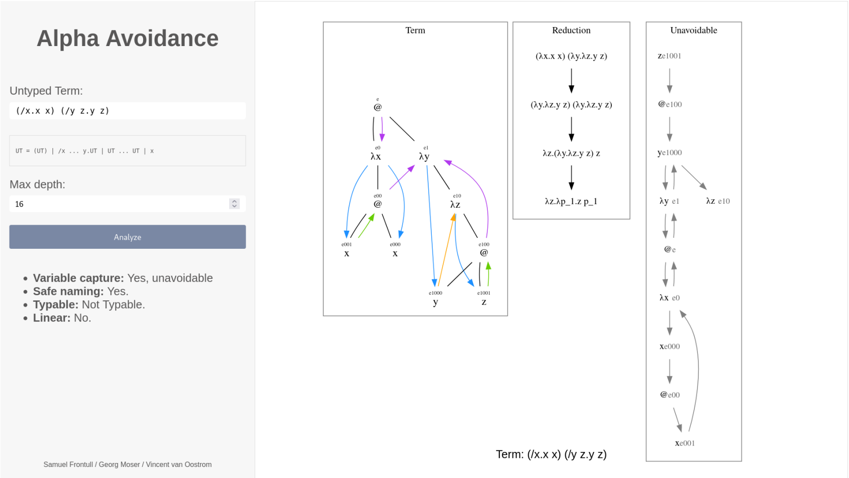
As already mentioned, α-paths characterise α-avoidance for seemingly unrelated calculi like (i) developments, (ii) affine λ-calculus, (iii) weak λ-calculus and (iv) safe λ-calculus. In Section 3 we have already seen this for developments and in the next section we illustrate this characterisation of the affine and the weak λ-calculus as well as the safe λ-calculus [11, 9].

In the sequel, we clarify the ancestry of α-paths wrt. the concept of *chains* in the μ-calculus, cf. [17]. Further, we briefly detail our tool Alpha that can be used to compute and illustrate α-paths.

Interpretation of μ in the λ-calculus

We show that α-paths are a strict generalisation of the chains considered for the μ-calculus in [17]. We do this by considering the sub-calculus Λ_μ of the λ-calculus obtained by the t-image of μ-terms defined as $t(x) = x$, $t(M\,N) = t(M)\,t(N)$, $t(\mu x.M) := A\,A\,(\lambda x.t(M))$ for $A = \lambda af.f\,(a\,a\,f)$. As suggested in [17], this translation allows simulating μ-terms in the λ-calculus, provided that we adopt the leftmost–outermost reduction strategy.

$$\begin{aligned} t(\mu x.M) := A\,A\,(\lambda x.t(M)) &\to_\beta (\lambda f.f\,(A\,A\,f))\,(\lambda x.t(M)) \to_\beta (\lambda x.t(M))\,(A\,A\,(\lambda x.t(M))) \\ &\to_\beta t(M)[\![x := A\,A\,(\lambda x.t(M))]\!] = t(M)[\![x := t(\mu x.M)]\!] \\ &= t(M[\![x := \mu x.M]\!])\ . \end{aligned}$$



Alpha Avoidance

Untyped Term:

(/x.x x) (/y z.y z)

UT = (UT) | /x ... y.UT | UT ... UT | x

Max depth:

16

Analyze

- **Variable capture:** Yes, unavoidable
- **Safe naming:** Yes.
- **Typable:** Not Typable.
- **Linear:** No.

Samuel Frontull / Georg Moser / Vincent van Oostrom

Term: (/x.x x) (/y z.y z)

Figure 9 α-avoidance tool web-interface.

In Λ_μ we can use α-paths to characterise α. We sketch the argument that an α-path in $t(M)$ for some μ-term M correspond a self-capturing chain in M. Note that, as reducing Turing's fixed point combinator $A\,A$ itself does not cause any capturing problems, we do not introduce "new α-problems". Thus, we only need to characterise the paths that correspond to reductions at the root of a reduct of $A\,A\,(\lambda x.M)$ to characterise the need for α in Λ_μ. For that, we observe that a pair of connected binding and capturing links in μ correspond to an *alb*-path in Λ_μ and a self-capturing chain to an α-path. Figure 8 illustrates this correspondence.

Implementation

Based on the notion of α-paths, we have implemented a tool, dubbed Alpha, to (partially) decide whether or not α-conversion can be avoided. The tool is publicly available and can either be accessed via the command-line or its web interface. The web interface also visualises the computed α-paths.

Depending on whether α-paths can be found or not (up to a variable depth), the tool gives one of the following results:

1) `alpha free`, if no α-paths were found and the calculation is terminated;

2) `alpha can be avoided`, if α-paths were found (but no unremovable α-paths); in this case, the tools prints an α-equivalent term for which the calculation is α-free;

3) `alpha is unavoidable`, if unremovable α-paths have been found;

or returns `maybe`, if no α-paths have been found, but the computation has not been terminated (the maximum depth has been reached). Recall that the problem is undecidable, cf. Section 4.[7] Listing 1 shows an exemplary output of the command line tool.

Listing 1 Church encoding of 3^2.

```
$ dune exec bin/main.exe "(/f␣x.f␣(f␣x))␣(/f␣x.f␣(f␣(f␣x)))"
alpha can be avoided:
(/f x.f (f x)) (/f p_12.f (f (f p_12)))
```

The web interface displays the α-graph and the computed α-paths. Figure 9 shows a screenshot of the tool illustrating this for $(\lambda x.x\,x)\,(\lambda yz.y\,z)$.

[7] The command line tool and the link to the web interface can be found at `https://tcs-informatik.uibk.ac.at/tools/alpha/`.

5 α-Avoidance In Affine, Safe And Weak λ-Calculi

In this section, we show how α-paths can be applied to analyse the need for α in restricted λ-calculi.

5.1 The affine λ-calculus

The affine λ-calculus [21, 23, 27, 38], forbids duplication by restricting term-formation, requiring the variable x to occur free at most once in M in an abstraction term $\lambda x.M$. This calculus is strongly normalising; we recall the central definition.

▶ **Definition 18.** *The set Λ_{AFF} of* affine λ*-terms is a subset of Λ and inductively defined as follows:*

- *(var)* $x \in \Lambda_{AFF}$*, for all variables x;*
- *(app)* $M, N \in \Lambda_{AFF} \implies M\,N \in \Lambda_{AFF}$*, if $\mathcal{FV}(M) \cap \mathcal{FV}(N) = \emptyset$;*
- *(abs)* $M \in \Lambda_{AFF} \implies \lambda x.M \in \Lambda_{AFF}$*.*

Since the size of terms steadily decreases with each reduction step and variables persist linearly along reductions, it follows that this calculus is strongly normalising. This allows a precise analysis for the need of α.

▶ **Lemma 19.** *Let $M \in \Lambda_{AFF}$, $M \to_\beta N$ and $q \prec p$ for some positions p, q in M. If $p \blacktriangleright p'$ and $q \blacktriangleright q'$, then $q' \prec p'$.*

Proof. Since we have no duplication, each symbol has at most one copy in N. We distinguish the following cases where we have $p \prec q$, with $p \blacktriangleright p'$ and $q \blacktriangleright q'$:

1. p, q both in the context: Then as $p' = p$ and $q' = q$ so by assumption we have $p' \prec q'$.
2. $p = o11s_1$ and $q = o11s_2$ both in the body: Then from $p \prec q$ we know that $s_1 \prec s_2$ and we have $os_1 = p' \prec q' = os_2$.
3. $p = o2s_1$ and $q = o2s_2$ both in the argument: Then from $p \prec q$ we know that $s_1 \prec s_2$ and we have $ots_1 = p' \prec q' = ots_2$.
4. p is in the context and $q = o11s$ in the body. Then $p' = p$ and $q' = os$ and since $p \prec q$ we also have $p' \prec q'$.
5. p is in the context and $q = o2s$ in the argument. Then $p' = p$ and $q' = oqs$. Since we know that $p \prec o$ (because it is in the context), we also have $p' \prec q'$.

The other cases can be omitted because they violate the assumption that $p \prec q$. ◀

Since each β-step preserves the property proven in Lemma 19, we cannot have a reduct of M where for the copy of p (the position of a variable), p', and the copy q (the position of an abstraction), q', we have $p' \parallel q'$, if for the origins we have $q \prec p$. This would temporarily be needed to form a redex whose contraction causes a variable capture. Moreover, as argued above we could map back such a setting to an (unremovable) α-path in M. We conclude that no such path can exist in M.

▶ **Lemma 20.** *Let M be an arbitrary term in Λ_{AFF}. There are no unremovable α-paths in M.*

In sum, we obtain the following, well-known result.

▶ **Theorem 21.** *In the affine λ-calculus α can be avoided.*

Proof. Due to Lemma 20 it only remains to prove that for every affine λ-term M there exists an affine λ-term N such that $M \equiv_\alpha N$ and N avoids α. This, however, follows from Lemma 19 in conjunction with Lemma 16. ◀

$$(var)\ \frac{}{x : A \vdash_s x : A} \qquad (const)\ \frac{}{\vdash_s f : A}\ f : A \in \Xi \qquad (wk)\ \frac{\Gamma \vdash_s M : A}{\Delta \vdash_s M : A}\ \Gamma \subset \Delta \qquad (\delta)\ \frac{\Gamma \vdash_s M : A}{\Gamma \vdash_{asa} M : A}$$

$$(app_{asa})\ \frac{\Gamma \vdash_{asa} M : A \to B \quad \Gamma \vdash_s N : A}{\Gamma \vdash_{asa} M\ N : B} \qquad (app)\ \frac{\Gamma \vdash_{asa} M : A \to B \quad \Gamma \vdash_s N : A}{\Gamma \vdash_s M\ N : B}\ \boldsymbol{ord\ B \leq ord\ \Gamma}$$

$$(abs)\ \frac{\Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash_{asa} M : B}{\Gamma \vdash_s \lambda x_1^{A_1} \ldots x_n^{A_n}.M : (A_1, \ldots, A_n, B)}\ \boldsymbol{ord\ (A_1, \ldots, A_n, B) \leq ord\ \Gamma}$$

Figure 10 The safe λ-calculus [9].

5.2 The safe λ-calculus

In the safe λ-calculus, a variable capture can never occur by definition, thus α is not needed. This calculus was first introduced in [28] and then further developed and formalised in [11]. The fundamental concept allowing α-free computations is known as the *safety* restriction. In the standard form this syntactic restriction restricts the free occurrences of variables according to their type-theoretic order. It was initially introduced for higher-order grammars, cf. [16]. The safe λ-calculus is the result of the transposal of the safety condition for higher-order grammars to the simply-typed calculus à la Church.

In this section, we show that α cannot be avoided in the safe λ-calculus as presented in [11] and [9] by giving a counterexample and clarify why we need to stick to a more restricted version of the safe λ-calculus (as presented in [10]) if we aim for α-free reductions. More precisely, we show how α-paths can be used to reason that α is not needed in the safe λ-calculus and that the absence of α-paths implies the safe variable typing convention.

Simple types over the atomic type o are defined as usual, cf. [6], $A_1 \to \cdots \to A_n \to o$ is abbreviated as $(A_1, \ldots, A_n, o)$ and (o) as o. The order of a type is given by (i) $ord\,o := 0$ and (ii) $ord\,(A \to B) := max(1 + ord\,A, ord\,B)$. The order of a typed term or symbol is defined to be the order of its type. The lowest order in a set of type assignments Γ is denoted by $ord\ \Gamma$ (0 if Γ empty). A set of type assignments Γ is *order-consistent* if all the types assigned to a given variable are of the same order.

▶ **Example 22.** Let $\Gamma = \{x : o, y : (o, o)\}$, then Γ is order-consistent and $ord\ \Gamma = 0$. Conversely, the set $\{x : ((o, o), o), x : (o, o)\}$ is not order-consistent and $ord\,\Gamma = 1$.

▶ **Definition 23.** *A term M of type A is said to be* safe*, if $\mathcal{FV}(M) \vdash_s M : A$ is a valid statement in the inference system of the safe λ-calculus depicted in Figure 10.*

We can abstract multiple variables at once, $\lambda x_1 \ldots x_n.M$, provided that they are pairwise distinct (*abs*-rule). In particular, $\lambda.x$ and $\lambda x^o.\lambda x^o.x$ are valid λ-terms of the safe λ-calculus, $\lambda x^o x^o.x$ is not. The conditions on the types in the *app*- and *abs*-rule ensure that the variables occurring free in some term M have order at least the order M (*safety* condition). The subscript *asa* stands for *almost safe* (application). Almost safe applications can be turned into a safe term via further applications or further abstractions. For example, $(\lambda x^o y^o.x)\,z$ (with z of type o) is an almost safe application but not safe. However, in $(\lambda x^o y^o.x)\,z\,f$ (with f, z of type o) it is a subterm of a safe application.

In the safe λ-calculus, consecutive redexes are contracted simultaneously, as the standard β-reduction does not preserve safety [9, Section 3.1.2]. This requires a notion of simultaneous substitution. The definitions of simultaneous capture-permitting and simultaneous capture-avoiding substitution are given in Table 2.

Table 2 Simultaneous capture-avoiding and simultaneous capture-permitting substitution.

M	$M[\![\overline{N}/\overline{x}]\!]$ (sim. capture-avoiding)	$M[\overline{N}/\overline{x}]$ (sim. capture-permitting)
x_i	N_i	N_i
y	y	y
$e_1\, e_2$	$e_1[\![\overline{N}/\overline{x}]\!]\, e_2[\![\overline{N}/\overline{x}]\!]$	$e_1[\overline{N}/\overline{x}]\, e_2[\overline{N}/\overline{x}]$
$(\lambda\overline{y}.e)[\![\overline{N}/\overline{x}]\!]$	$\lambda\overline{y}.e[\![\overline{N'}/\overline{x'}]\!]$ where $\overline{x'} = \overline{x} - \overline{y}$ if $\overline{y} \cap FV(t) = \emptyset$ for all $t \in \overline{N'}$, else $\lambda\overline{z}.e[\![\overline{z}/\overline{y}]\!][\![\overline{N}/\overline{x}]\!]$ where z_i fresh for $e, \overline{N}$	$\lambda\overline{y}.e[\overline{N'}/\overline{x'}]$ where $\overline{x'} = \overline{x} - \overline{y}$

▶ **Definition 24** (*safe redex* [9, Definition 3.21])**.** *An untyped safe redex is an untyped almost safe application (a succession of several standard redexes) of the form $(\lambda x_1 \dots x_n.M)\, N_1 \dots N_l$ for some $l, n \geq 1$ where $\lambda x_1 \dots x_n.M$ is safe and each N_i, for $1 \leq i \leq n$, is safe.*

▶ **Definition 25** (*safe redex contraction*)**.** *The relation β_s is defined on the set of safe redexes as follows:*

$$\beta_s = \{(\lambda x_1 \dots x_n.M)\, N_1\, \dots\, N_l \mapsto (\lambda x_{l+1} \dots x_n.M)[N_1 \dots N_l/x_1 \dots x_l] \mid n > l\}$$
$$\cup\, \{(\lambda x_1 \dots x_n.M)\, N_1\, \dots\, N_l \mapsto M[N_1 \dots N_n/x_1 \dots x_n]\, N_{n+1}\, \dots\, N_l \mid n \leq l\}$$

where $\lambda.M = M$ and $M[\overline{N}/\overline{x}]$ denotes the simultaneous capture-permitting substitution.

Note that simultaneous capture-permitting substitution cannot be applied serially because it may require α. The statement $M[x_1 := N_1][x_2 := N_2] = M[x := y, y := z]$ is not true in general, as x_2 may be free in N_1, e.g. $x[x := y][y := z] = z$ and $x[x := y, y := z] = y$.

▶ **Definition 26.** *The* safe *β-reduction, written as $\rightarrow_{\beta_s}$, is the compatible closure of the relation β_s with respect to the formation rules of the safe λ-calculus.*

In addition to the inference rules, the safe variable typing convention is adopted, which restricts the naming of variables according to their type.

▶ **Definition 27** (*safe variable typing convention* [9])**.** *A type-annotated term M is order-consistent just if the set of type-assignments induced by the type annotations in M is. In any definition, theorem or proof involving countably many terms, it is assumed that the set of terms involved is order-consistent.*

According the safe variable typing convention, variables of distinct order must have distinct names. This is crucial for α-avoidance in the safe λ-calculus.[8] However, if we consider the term $M = \lambda y^o.(\lambda x^o y^o.x)\, y$ we see that although this term is safe ($\vdash_s \lambda y^o.(\lambda x^o y^o.x)\, y : (o, o)$) and $(\lambda x^o y^o.x)\, y$ is a safe redex, it cannot be contracted by means of capture-permitting substitution, because this would lead to a variable capture. This invalidates a central property of this calculus, according to which a variable capture can never happen, and leads to the fact that we may compute different normal forms for α-equivalent terms.[9]

[8] $\{y : (((o, o), o), o), z : ((o, o), o)\} \vdash_s \underline{(\lambda x^{((o,o),o)} y^{(o,o)} z^o.x)\ (\lambda q^{(o,o)}.y\ z)} : ((o, o), o, ((o, o), o))$ is a counter-example to [9, Lemma 3.17].

[9] Compare to the errata published at `https://github.com/blumu/dphil.thesis/blob/erratum/Current/thesis-erratum/dphilerratum.pdf`.

$$(var)\ \frac{}{\{x : A\} \vdash_{s\alpha} x : A} \qquad (const)\ \frac{}{\vdash_{s\alpha} f : A}\ f : A \in \Xi \qquad (wk)\ \frac{\Gamma' \vdash_{s\alpha} M : A}{\Gamma \vdash_{s\alpha} M : A}\ \Gamma' \subset \Gamma$$

$$(app)\ \frac{\Gamma \vdash_{s\alpha} M : (A_1, \dots, A_n, B) \quad \Gamma_{\geq m} \vdash_{s\alpha} N_1 : A_1 \quad \dots \quad \Gamma_{\geq m} \vdash_{s\alpha} N_j : B_j}{\Gamma \vdash_{s\alpha} M\, N_1 \dots N_j : B} \quad m = ord\, B$$

$$(abs)\ \frac{\Gamma_{\geq m} \cup \{x_1 : A_1, \dots, x_n : A_n\} \vdash_{s\alpha} M : B}{\Gamma \vdash_{s\alpha} \lambda x_1 \dots x_n . M : (A_1, \dots, A_n, B)} \quad \boldsymbol{m = ord\,(A_1, \dots, A_n, B)}$$

Figure 11 An α-avoiding safe λ-calculus.

A more restrictive set of rules is needed to resolve this issue. These rules are depicted in Figure 11.[10] In this system we dropped the "almost safety" and allow to type only applications that provide enough arguments to abstractions. More precisely, if an argument of order k is provided, the arguments of all abstracted variables of order k and higher must be provided. In this way, we avoid free variables ending up in the scope of abstractions of the same order during reduction. This avoids potential variable capture, since it can be assumed that free variables are always of a higher order than the abstractions they enter the scope of. Therefore, according to the safe variable typing convention, they are named differently.[11]

▶ **Example 28.** The simply-typed term $(\lambda f^{(o,o,o)} y^o . f\, y)\, (\lambda x^o y^o . x)$ is derivable in the safe λ-calculus from Figure 10, but not in the system from Figure 11 because of the unsafe application $f\, y$. Indeed, this term reduces in one step to $\lambda y^o . (\lambda x^o y^o . x)\, y$ where α is required to further reduce it.

In the following Lemma 29 we show that the safe λ-calculus of Figure 11 avoids α by reasoning with α-paths. This can be done by interpreting safe λ-terms as ordinary terms.

▶ **Theorem 29.** *In the safe λ-calculus no variable capture can occur, provided that the safe variable typing convention is adopted.*

Proof. Suppose we have an α-path in a safe λ-term M with $\Gamma \vdash_{s\alpha} M : A$. Then this path would start at a variable y occurring free in the argument N of some application, which is connected via a legal path to an abstraction λx binding a variable x in the scope of a λy, as illustrated below. In such case, by definition of safe terms, we know that $\lambda x . M$ and N are both safe. Moreover, we know that $ord\, y \geq N$ and $ord\, N = ord\, x$. We can therefore have the following two cases: (i) $ord\, y > ord\, x$ or (ii) $ord\, y = ord\, x$. In any case, as the subterm $\lambda y . M'$ would be unsafe in isolation, we conclude that the λy and the λx must be jointly abstracted. By definition of safe β-reduction, we know that compound abstractions of same order are contracted simultaneously. Therefore, we cannot have a variable capture. ◀

5.3 The weak λ-calculus

The *weak* λ-calculus [39] forbids to contract *open* redexes, i.e. redexes that involve free variables that are bound outside. Thus, if the name of the free variables and the bound variables are chosen to be distinct, a variable capture can by definition never occur. We recall the notion of *weak β-reduction*.

[10] Simultaneous substitutions coincide with the singleton substitutions from Table 1 in the case $|\overline{x}| = 1$.

[11] We note that these rules correspond to the rules of the safe λ-calculus published in [10] and to the typing rules for *long-safe* terms (without constants) listed in [9, Table 3.2].

▶ **Definition 30** (*weak λ-reduction [39, Definition 3.1]*)**.** *A particular occurrence of a redex R in a λ-term M will be called weak in M iff no variable-occurrence free in R is bound in M. A weak β-contraction in M is the contraction of a β-redex-occurrence that is weak in M.*

The characterisation of the virtual redexes by legal paths is not suitable for the weak λ-calculus, since they include redexes that are not reduced at all. However, we can infer from the structure of the unremovable α-paths that α can also be avoided in this calculus. To this end, we rely on the fact that bound variables are never released, i.e. they do not change or loose their binder.

▶ **Lemma 31.** *For every λ-term M there exists a λ-term N such that $M \equiv_\alpha N$ and any $\to_{\beta w}$-reduction from N is α-free.*

Proof. We prove it by showing that the name-collision characterised by an unremovable α-paths will not arise. Suppose we have an unremovable α-path in a λ-term M. Such path has the shape $\sigma^+_{alb} \cdot c$. Assume, that at some point along the reduction sequence of M we reach a λ-term N, containing a redex R whose contraction leads to the predicted name-collision. Let q be the position of the variable y occurring free in the argument of R in N. Since the position q originates from position p in M ($p \blacktriangleright p' \blacktriangleright \ldots \blacktriangleright q$) and the variable occurrence at position p in M was bound, we know that also the variable y at position q in N is bound (as bound variables are never released). So R would be an open redex and thus not contracted. Any other α-path can be removed by naming each binder distinctly and distinct from the free variables, as proven in Lemma 16. ◀

In sum, α-avoidance is immediate from the definitions.

▶ **Theorem 32.** *In the weak λ-calculus α can be avoided.*

6 Conclusion

We have presented a sound characterisation of α-avoidance, via α-paths, generalising self-capturing chains [17], studied in the context of the μ-calculus; α-paths exploit the predictive power of legal paths, characterising virtual redexes of a λ-term M, that is, all redexes occurring in some reduction sequence starting from M. By reasoning on the structure of the initial term, we estimated whether α is needed, when contracting these virtual redexes. Further, we have shown undecidability of α avoidance for (leftmost-outermost reductions in) the untyped λ-calculus. Moreover, α-paths were instantiated to different restrictive λ-calculi, where they can be used to show that α can be avoided, namely developments, the affine λ-calculus, the weak λ-calculus and the safe λ-calculus. In short, forbidding redex creation, duplication, or the contraction of redexes involving variables bound outside is enough to allow α-avoidance. For all calculi where we can avoid α, we can infer potential α-conversions needed to allow α-free computations from the α-paths. This allows to move a dynamic problem to a static one.

We have shown that α-avoidance is undecidable for the leftmost–outermost strategy in the untyped λ-calculus. These leaves the question open, whether undecidability holds in general. We further note that α-paths only overapproximate the need for α. It remains an open question whether we could tighten the definition of α-paths such that the established (sound) characterisation becomes precise, that is, complete. These questions are left to future work.

References

1 Samson Abramsky and C.-H. Luke Ong. Full abstraction in the lazy lambda calculus. *Information and Computation*, 105(2):159–267, 1993. doi:10.1006/inco.1993.1044.

2 Andrea Asperti, Vincent Danos, Cosimo Laneve, and Laurent Regnier. Paths in the lambda-calculus. In *Proceedings of the Ninth Annual Symposium on Logic in Computer Science (LICS '94), Paris, France, July 4-7, 1994*, pages 426–436. IEEE Computer Society, 1994. doi:10.1109/LICS.1994.316048.

3 Andrea Asperti and Stefano Guerrini. *The optimal implementation of functional programming languages*, volume 45 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, 1998.

4 Andrea Asperti and Cosimo Laneve. Paths, computations and labels in the λ-calculus. *Theoretical Computer Science*, 142(2):277–297, 1995. doi:10.1016/0304-3975(94)00279-7.

5 Thibaut Balabonski. Weak optimality, and the meaning of sharing. In Greg Morrisett and Tarmo Uustalu, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, pages 263–274. ACM, 2013. doi:10.1145/2500365.2500606.

6 Henk P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 2nd revised edition, 1984. doi:10.2307/2274112.

7 Henk P. Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Perspectives in logic. Cambridge University Press, 2013.

8 Tomasz Blanc, Jean-Jacques Lévy, and Luc Maranget. Sharing in the Weak Lambda-Calculus. In Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel C. de Vrijer, editors, *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*, volume 3838 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2005. doi:10.1007/11601548_7.

9 William Blum. *The Safe Lambda Calculus*. PhD thesis, Oxford University, UK, 2009.

10 William Blum and C.-H. Luke Ong. The Safe Lambda Calculus. In *TLCA*, pages 39–53, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

11 William Blum and C.-H. Luke Ong. The safe lambda calculus. *Logical Methods in Computer Science*, Volume 5, Issue 1, February 2009. doi:10.48550/arXiv.0901.2399.

12 Nicolaas Govert de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church–Rosser theorem. *Indagationes Mathematicae*, 75(5):381–392, 1972. Proc. 19th International Conference on Automated Deduction. doi:10.1016/1385-7258(72)90034-0.

13 Felice Cardone and J. Roger Hindley. Lambda-Calculus and Combinators in the 20th Century. In *Logic from Russell to Church*, 2009. doi:10.1016/S1874-5857(09)70018-4.

14 Alonzo Church and John Barkley Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39:472–482, 1936. doi:10.1090/S0002-9947-1936-1501858-0.

15 Haskell B. Curry. *Combinatory Logic*. Amsterdam: North-Holland Pub. Co., 1958.

16 Werner Damm. The IO- and OI-hierarchies. *Theoretical Computer Science*, 20(2):95–207, 1982. doi:10.1016/0304-3975(82)90009-3.

17 Jörg Endrullis, Clemens Grabmayer, Jan Willem Klop, and Vincent van Oostrom. On equal μ-terms. *Theoretical Computer Science*, 412(28):3175–3202, 2011. doi:10.1016/j.tcs.2011.04.011.

18 Samuel Frontull. Alpha Avoidance. Master's thesis, University of Innsbruck, 2021.

19 Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *Proc. 14th LICS*, pages 214–224, 1999. doi:10.1109/LICS.1999.782617.

20 J. Roger Hindley. Reductions of Residuals are Finite. *Transactions of the American Mathematical Society*, 240:345–361, 1978. doi:10.2307/1998825.

21 J. Roger Hindley. BCK-Combinators and Linear lambda-Terms have Types. *Theoretical Computer Science*, 64(1):97–105, 1989. doi:10.1016/0304-3975(89)90100-X.

22 Martin Hyland. A simple proof of the Church–Rosser theorem. Oxford University, UK, 1973.

23 Bart Jacobs. Semantics of lambda-I and of other substructure lambda calculi. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications*, pages 195–208, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg. doi:10.1007/BFb0037107.

24 Jan W. Klop. *Combinatory reduction systems*. PhD thesis, Rijksuniversiteit Utrecht, 1980.

25 Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983. doi:10.1016/0304-3975(82)90125-6.

26 Clemens Kupke, Johannes Marti, and Yde Venema. Size measures and alphabetic equivalence in the μ-calculus. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 18:1–18:13. ACM, 2022. doi:10.1145/3531130.3533339.

27 Harry G. Mairson. Linear lambda calculus and PTIME-completeness. *Journal of Functional Programming*, 14(6):623–633, 2004. doi:10.1017/S0956796804005131.

28 Jolie G. de Miranda. *Structures generated by higher-order grammars and the safety constraint*. PhD thesis, University of Oxford, UK, 2006.

29 Maxwell H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of mathematics*, 43(2):223–243, 1942. doi:10.2307/1968867.

30 Vincent van Oostrom and Roel de Vrijer. Four equivalent equivalences of reductions. *Electronic Notes in Theoretical Computer Science*, 70(6):21–61, 2002. WRS 2002, 2nd International Workshop on Reduction Strategies in Rewriting and Programming - Final Proceedings (FLoC Satellite Event). doi:10.1016/S1571-0661(04)80599-1.

31 Simon L. Peyton Jones. *The Implementation of Functional Programming Languages*. Prentice Hall, January 1987.

32 Emil L. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946. doi:10.2307/2267252.

33 John Barkley Rosser. Review: H. B. Curry, A New Proof of the Church–Rosser Theorem. *Journal of Symbolic Logic*, 21(4):377–378, 1956.

34 David E. Schroer. *The Church–Rosser Theorem*. PhD thesis, Cornell University, 1965.

35 Rick Statman. On the complexity of alpha conversion. *The Journal of Symbolic Logic*, 72(4):1197–1203, 2007. doi:10.2178/jsl/1203350781.

36 Terese. *Term rewriting systems*. Cambridge University Press, 2003.

37 Christopher P. Wadsworth. *Semantics and Pragmatics of the Lambda-Calculus*. PhD thesis, University of Oxford, 1971.

38 Noam Zeilberger. Linear lambda terms as invariants of rooted trivalent maps. *Journal of Functional Programming*, 26:e21, 2016. doi:10.1017/S095679681600023X.

39 Naim Çağman and J. Roger Hindley. Combinatory weak reduction in lambda calculus. *Theoretical Computer Science*, 198(1-2):239–247, 1998. doi:10.1016/S0304-3975(97)00250-8.

Partial Model Checking and Partial Model Synthesis in LTL Using a Tableau-Based Approach

Serenella Cerrito
Université Paris Saclay, Univ EVRY, France

Valentin Goranko
Stockholm University, Sweden
and University of the Witwatersrand, Johannesburg, South Africa (visiting professorship)

Sophie Paillocher
Université Paris Saclay Univ EVRY, France

Abstract

In the process of designing a computer system S and checking whether an abstract model $\mathscr{M}$ of S verifies a given specification property η, one might have only a partial knowledge of the model, either because $\mathscr{M}$ has not yet been completely defined (constructed) by the designer, or because it is not completely observable by the verifier. This leads to new verification problems, subsuming satisfiability and model checking as special cases. We state and discuss these problems in the case of LTL specifications, and develop a uniform tableau-based approach for their solutions.

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

Keywords and phrases Linear temporal logic LTL, partial transition systems, partial model checking, partial model synthesis, tableaux

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.23

Acknowledgements We thank the anonymous reviewers for their detailed and helpful comments.

1 Introduction

In the process of designing a computer system S and checking whether an abstract transition system $\mathscr{M}$ modelling S satisfies a given specification property η, one might have only a partial knowledge of the model, either because $\mathscr{M}$ has not yet been completely defined (constructed) by the designer, or because it is not completely observable by the verifier. Typically, $\mathscr{M}$ is a finite transition system with states labeled by sets of atomic propositions (Boolean variables) with their truth values, and the partial construction or partial observability may be manifested in one or more of the following components: a) the truth values of some propositions at some states; b) whether two given states are connected by a transition; c) whether the observed states are all the existing or intended states, or there are more, not observed or not yet constructed. In a multi-agent context there are additional components, e.g., some agents and/or their actions might also be partly observable or unspecified yet.

Now two natural questions arise:

i) *whether there is at least some way to extend (eventually, complete) the partially constructed transition system in such a way that the property η holds.*

ii) *whether what is already known suffices to verify η, no matter how the partial information might be extended (eventually, completed).*

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 23; pp. 23:1–23:21
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

(Alternatively, one may think that in both cases the model is partially constructed, but in the former case it is up to the verifier to complete the construction, whereas in the latter case it will be completed by another, possibly adversarial, agent.)

These questions have been stated respectively as *partial model synthesis* and *partial model checking* and discussed in general in [14], on which the present work follows up.

In this paper we take the property η to be expressed by an LTL formula and the model $\mathscr{M}$ to be a transition system (Kripke structure), which is incomplete in either of the senses a, b and c described above. Then the two questions above unfold into four decision problems and two associated synthesis problems, precisely described in Section 3. The aim of this work is not only to provide a unified approach for solving the decision problems, but also to provide *constructive* methods for their solutions, thus also solving the associated synthesis problems, by constructive methods that are neither non-deterministic nor brute-force. Thus, when considering, for instance, whether there exists an extension of the partially given model $\mathscr{M}$ assuring the existence of a path verifying η, we are not only interested in providing a correct YES/NO answer, but, in the case where the answer is positive, we want *to build* an extension $\mathscr{M}'$ of the input $\mathscr{M}$, and a path in it that is a linear model of η.

Furthermore, when extending the model with truth values of Boolean variables at states, we aim to do it in a "minimal" (or, "most general") way, so as to leave to the designer as much freedom as possible for further extensions. That is, if a given Boolean variable p has an unspecified (or, unknown) truth value at a given state s on a path that is to satisfy η, we only force that value to be defined (True or False) when this is really necessary to produce such a path. For these reasons, *inter alia*, our methods and algorithms are inspired by the *tableau methods* for constructively solving the satisfiability problem in LTL, first developed in [22] and further optimised and presented in detail in [11, Chapter 13], that are usually proposed as the most constructive methods for solving the satisfiability problem. Here we will follow the style and technical details of the tableaux construction procedure in the latter.

The decision problems described in Section 3 informally ask respectively whether some (resp., every) admissible extension of the partial model is such that some (resp., every) path in it starting from the given initial state satisfies η. These problems are denoted respectively EE, AA, EA, AE. In this work we first present in full details the solution to EE (and to its dual problem AA) in the case where only the state labels (given by the truth values of the Boolean variables) may be not yet determined or unknown, but the states themselves and the transitions between them are completely specified and observed. Then we show how the proposed algorithm can be extended to deal also with the cases where transitions and/or states may be not yet determined or unknown, partially or completely (the limit case being when nothing is known at all). After that, we show how the proposed methods for solving EE and AA also enable handling of the problems EA and AE. Thus, the algorithm that we initially propose for solving EE is the *core* method of our approach, to which all the other variants of model extension or completion problems are reduced.

Some of the problems that we address here are related to previously published work, esp. on partial model checking of LTL and other logics. For instance, there is a clear connection between our work and [9, 13]. We postpone the discussion of related works to Section 7.2.

The paper is organized as follows. In section 2 we briefly recall some standard notions on LTL and we set some preliminary definitions and notation. In Section 3 we state the main problems. In Section 4 we describe our core algorithm and in Section 5 we present basic results about it. In Section 6 we show how it can be used to solve the other problems we consider. Section 7 discusses related works, points at some perspectives, and concludes.

2 Preliminaries

2.1 Partial transition systems, extensions, and completions

In what follows, Prop is a nonempty finite set of atomic propositions (Boolean variables) and $B = \{\top, ?, \bot\}$ is the set of truth-values, where ? is a third Boolean value intuitively meaning "undetermined" or "unknown". The definition below extends the well-known notion of transition system (aka Kripke structure) to the case of partial construction or incomplete knowledge. It combines some ideas that come, *inter alia*, from [16] (for transitions) and [13] (for three-valued state labels).

▶ **Definition 1** (Partial and complete transition systems). *A **Partial transition system** (also called further a **partial model**) is a tuple $\mathscr{M} = (S, R^{must}, R^{may}, L)$, where: S is a finite set of states, $R^{must} \subseteq S \times S$ and $R^{may} \subseteq S \times S$ are two transition relations such that $R^{must} \subseteq R^{may}$, and $L : S \times \mathrm{Prop} \to B$ is an **interpretation function** associating with each state in S and each atomic proposition in* Prop *a truth value in B. The relation R^{must} corresponds to the already determined, or known, transitions, whereas R^{may} also includes the possible but not yet determined, or not yet known, transitions.*

We assume that R^{may} is serial (aka, total); hence, every path in (S, R^{may}, L) is infinite.

*If $R^{may} = R^{must} = R$, we denote the partial transition system simply as (S, R, L) and say that it is **transition-complete**. A transition-complete structure (S, R, L) is **complete**, or just a **transition system**, if all values that L assigns are in $\{\top, \bot\}$. Thus, every (complete) transition system is also a partial transition system.*

▶ **Definition 2** (Extension and completion of an interpretation function). *Given a partial transition system $\mathscr{M} = (S, R^{must}, R^{may}, L)$, an interpretation function $L' : S \times \mathrm{Prop} \to B$ is said to be an **extension** of L, denoted by $L \preceq L'$, if for each $s \in S$ and $p \in$ Prop, if $L(s,p) \neq ?$ then $L'(s,p) = L(s,p)$. When $L' : S \times \mathrm{Prop} \to \{\top, \bot\}$ (i.e., all values that L' assigns are in $\{\top, \bot\}$), then L' is said to be a **completion**, denoted $L \preceq^c L'$.*

▶ **Definition 3** (Extension and completion of a partial transition system). *A partial transition system $\mathscr{M}' = (S', R'^{must}, R'^{may}, L')$ is an **extension of a partial transition system** $\mathscr{M} = (S, R^{must}, R^{may}, L)$, denoted by $\mathscr{M} \preceq \mathscr{M}'$ if: $S \subseteq S'$, $R^{must} \subseteq R'^{must}$, $R'^{may} \subseteq R^{may}$, and $L \preceq L'$. If R'^{must} is serial, then $\mathscr{M}'$ is a **total extension** of $\mathscr{M}$. A complete transition system $\mathscr{M}' = (S', R', L')$ which is a total extension of $\mathscr{M}'$ is a **completion** of $\mathscr{M}$, denoted by $\mathscr{M} \preceq^c \mathscr{M}'$. Thus, $\mathscr{M} \preceq^c \mathscr{M}'$ iff: $S \subseteq S'$, $R^{must} \subseteq R' \subseteq R^{may}$, R' is serial, and $L \preceq^c L'$.*

2.2 The logic LTL

Here we only fix basic notation and terminology on the Linear Temporal Logic LTL used further in the paper. For further details the reader is referred e.g., to [11].

The formulae of LTL are given by the grammar

$$\varphi, \psi := \top \mid \bot \mid p \mid \neg\varphi \mid (\varphi \wedge \psi) \mid \mathsf{X}\,\varphi \mid \mathsf{G}\,\varphi \mid (\psi\,\mathsf{U}\,\varphi)$$

where $p \in \mathrm{Prop}$. The operators $\vee$, $\to$, $\leftrightarrow$, and F are defined in the standard way.

A **linear model** for LTL is an infinite sequence $\sigma : \mathbb{N} \to \mathscr{P}(\mathrm{Prop})$. **Truth (satisfaction) of an** LTL **formula** φ **at a position** $i \in \mathbb{N}$ **of a linear model** σ, denoted $\sigma, i \models \varphi$, is defined as usual. An LTL formula φ is said to be **true in a linear model** σ, denoted $\sigma \models \varphi$, if $\sigma, 0 \models \varphi$; in this case σ is said to be a **linear model of** φ.

LTL formulae are also interpreted at states in transition systems, as follows. Given a transition system $\mathscr{M} = (S, R, L)$ the sequence $\pi = s_0, s_1, s_2, ...$ is a **path in** $\mathscr{M}$ if $s_0 R s_1 R s_2, ...$. Every finite sequence $s_0, s_1, ..., s_n$ is called an **initial sub-path of** π. The **computation in** $\mathscr{M}$ **corresponding to** π is the sequence of labels (sets of true propositions) along π: $T(s_0), T(s_1), T(s_2) \ldots, ...$, where $T(i) = \{p \in \text{Prop} \mid L(s_i, p) = \top\}$. Given a state $s \in S$, we denote by $comput_{\mathscr{M}}(s)$ the set of computations in $\mathscr{M}$ with initial state s.

Every computation in $\mathscr{M}$ can be regarded as a linear LTL model, hence the notions $\sigma, i \models \varphi$ and $\sigma \models \varphi$ are readily defined for any computation σ. We say that an LTL formula φ is **universally true** in a transition system $\mathscr{M}$ at a state s if $\sigma \models \varphi$ for all elements σ of $comput_{\mathscr{M}}(s)$. In that case, we also simply say that φ **is true in** $\mathscr{M}$ **at** s, denoted $\mathscr{M}, s \models_{\forall} \varphi$ or just $\mathscr{M}, s \models \varphi$. Likewise, we say that φ is **existentially true in** $\mathscr{M}$ **at** s if for some computation $\sigma \in comput_{\mathscr{M}}(s)$ we have $\sigma \models \varphi$. In that case we write $\mathscr{M}, s \models_{\exists} \varphi$.

Given a finite set of formulas $\Gamma = \{\varphi_1, \ldots, \varphi_n\}$, where $n \geq 0$, we denote
$\bigvee \Gamma := \varphi_1 \vee \cdots \vee \varphi_n$, and $\bigwedge \Gamma := \varphi_1 \wedge \cdots \wedge \varphi_n$. When $\Gamma = \emptyset$, $\bigvee \Gamma := \bot$ and $\bigwedge \Gamma := \top$.
We will write $\mathscr{M}, s \models \Gamma$ as a shorthand for $\mathscr{M}, s \models \bigwedge \Gamma$.

2.3 Tableau-related preliminaries

The definitions here are borrowed from the presentation of tableaux for LTL in [11, Ch.13.2].

We distinguish four types of formulas in LTL: *literals (atoms or negated atoms), conjunctive, disjunctive and successor formulas*, and *eventualities*, i.e. formulas of the type $\mathsf{F}\,\varphi$ or $\psi \,\mathsf{U}\, \varphi$. In both cases, their truth at a time point implies that φ *must be realised at some point in the future*, but its realisation can be procrastinated. By the well known fixed-point LTL equivalences $\mathsf{F}\,\varphi \equiv \varphi \vee \mathsf{X}\,\mathsf{F}\,\varphi$ and $\psi \,\mathsf{U}\, \varphi \equiv \varphi \vee (\psi \wedge \mathsf{X}\,(\psi \,\mathsf{U}\, \varphi))$, an eventuality can be seen as a disjunction, as it is shown Figure 1.

▶ **Definition 4** (Components of a formula). *The* ***components of a formula*** φ *are defined depending on its type. The components of a conjunctive (resp. disjunctive) formula* φ *are formulas, defined further, such that* φ *is equivalent to their conjunction (resp. disjunction). A successor formula* ψ *has only one component, written* $scomp(\psi)$*. Literals do not have components. All types and components of LTL-formulas are summarised in table 1.*

Table 1 Types and components of formulas in LTL.

conjunctive		disjunctive		successor	
formula	components	formula	components	formula	components
$\neg\neg\varphi$	$\varphi,\ \varphi$	$\varphi \vee \psi$	$\varphi,\ \ \psi$	$\mathsf{X}\,\varphi$	φ
$\varphi \wedge \psi$	$\varphi,\ \psi$	$\varphi \to \psi$	$\neg\varphi,\ \psi$	$\neg\mathsf{X}\,\varphi$	$\neg\varphi$
$\neg(\varphi \vee \psi)$	$\neg\varphi,\ \ \neg\psi$	$\neg(\varphi \wedge \psi)$	$\neg\varphi,\ \neg\psi$		
$\neg(\varphi \to \psi)$	$\varphi,\ \neg\psi$	$\mathsf{F}\,\varphi$	$\varphi,\ \mathsf{X}\,\mathsf{F}\,\varphi$		
$\mathsf{G}\,\varphi$	$\varphi,\ \mathsf{X}\,\mathsf{G}\,\varphi$	$\psi \,\mathsf{U}\, \varphi$	$\varphi,\ \psi \wedge \mathsf{X}\,(\psi \,\mathsf{U}\, \varphi)$		
$\neg\mathsf{F}\,\varphi$	$\neg\varphi,\ \mathsf{X}\,\neg\mathsf{F}\,\varphi$	$\neg\mathsf{G}\,\varphi$	$\neg\varphi,\ \mathsf{X}\,\neg\mathsf{G}\,\varphi$		
$\neg(\psi \,\mathsf{U}\, \varphi)$	$\neg\psi \vee \mathsf{X}\,\neg(\psi \,\mathsf{U}\, \varphi),\ \neg\varphi$				

When Γ is a set of formulae, the notation $Scomp(\Gamma)$ will denote $\{scomp(\psi) | \psi \in \Gamma\}$.

▶ **Definition 5** (Extended closure of a formula). *The extended closure* $\mathsf{ecl}(\varphi)$ *of the formula* φ *is the least set of formulas such that :*

1. $\varphi \in \mathsf{ecl}(\varphi)$*,*
2. $\mathsf{ecl}(\varphi)$ *is closed under taking all conjunctive, disjunctive, successor components of the respective formulas in* $\mathsf{ecl}(\varphi)$*.*

▶ **Definition 6** (Extended closure of a set of formulas). *For any set of formulas Γ we define*

$$\mathsf{ecl}(\Gamma) := \bigcup\{\mathsf{ecl}(\varphi) | \varphi \in \Gamma\}.$$

A set of formulas Γ is ***closed*** *if $\Gamma = \mathsf{ecl}(\Gamma)$.*

▶ **Definition 7** (Patently inconsistent). *A set of formulas is* ***patently inconsistent*** *if it contains $\bot$, or $\neg\top$, or a contradictory pair of formula φ and $\neg\varphi$.*

▶ **Definition 8** (Fully expanded). *A set of formulas Γ is* ***fully expanded*** *if and only if*

1. *it is not patently inconsistent,*
2. *for every conjunctive formula in Γ, all of its conjunctive components are in Γ,*
3. *for every disjunctive formula in Γ, at least one of its disjunctive components is in Γ.*

Note that the empty set of formulas is vacuously fully expanded.

▶ **Definition 9** (Full expansion of a set of formulas). *A fully expanded set of formulas Δ is a* ***full expansion*** *of a set of formulas Γ if Δ can be obtained from Γ by repeated applications of the following rules, where initially no formula is marked as "used":*

- ***(C-comp)*** *for every conjunctive formula φ in the current set Γ that has not been marked as "used", add all of its conjunctive components to Γ and mark φ as "used".*
- ***(D-comp)*** *for every disjunctive formula φ in the current set Γ that has not been marked as "used", add one of its disjunctive components to Γ and mark φ as "used".*

Note that the rule (D-comp) is non-deterministic, so a set Γ may have several (or no) full expansions. The notation $FE(\Delta)$ means the set of the full expansions of a set of formulas Δ.

Below we also recall some definitions and a theorem about Hintikka traces, that can be found in [11], as we will make use of them in our approach.

▶ **Definition 10.** *Given a closed set of formulas Γ, a* ***Hintikka trace*** *(HT) for Γ is a mapping $H : \mathbb{N} \to \mathscr{P}(\Gamma)$ satisfying the following conditions for every $n \in \mathbb{N}$:*

1. *$H(n)$ is fully expanded.*
2. *If $\varphi \in H(n)$ is a successor formula, then scomp$(\varphi) \in H(n+1)$.*
3. *If $\varphi \,\mathsf{U}\, \psi \in H(n)$, then there exists $i \geq n$ such that $\psi \in H(n+i)$ and $\varphi \in H(n+j)$ for every j such that $0 \leq j < i$.*

An LTL *formula φ is* ***satisfiable in a Hintikka trace*** *H if $\varphi \in H(n)$ for some $n \in \mathbb{N}$.*

▶ **Theorem 11.** *An LTL formula η is satisfiable iff it is satisfiable in some Hintikka trace.*

3 The problems we study

Let $\mathscr{M}$ be a partial transition system, let s_0 be a state in $\mathscr{M}$ and let η be an LTL formula. The following four decision problems naturally arise (cf. Definitions 2 and 3):

- **Existential extension for path existence**
  Are there a total extension $\mathscr{M}'$ of $\mathscr{M}$ and a path $s_0, s_1, s_2, \ldots$ in $\mathscr{M}'$ such that its corresponding computation σ is a linear model of η? We denote this decision problem by EE.
- **Existential extension for all paths**
  Is there a total extension $\mathscr{M}'$ of $\mathscr{M}$, so that for all paths $s_0, s_1, s_2, \ldots$ in $\mathscr{M}'$ the corresponding computations are linear models of η? We denote this decision problem by EA.

- **Universal extension for path existence**
  Does every total extension $\mathscr{M}'$ of $\mathscr{M}$ has a path $s_0, s_1, s_2, \ldots$ such that its corresponding computation is a linear model of η? We denote this decision problem by AE.
- **Universal extension for all paths**
  Is every total extension $\mathscr{M}'$ of $\mathscr{M}$ such that for every path $s_0, s_1, s_2, \ldots$ in it the corresponding computation is a linear model of η? We denote this decision problem by AA.

Clearly, the problems EE and AA are dual: a positive answer to EE with input $(\mathscr{M}, s_0, \eta)$ is a negative answer for AA on input $(\mathscr{M}, s_0, \neg\eta)$, while a negative answer for EE on input $(\mathscr{M}, s_0, \eta)$ is a positive answer for AA on input $(\mathscr{M}, s_0, \neg\eta)$. Likewise, EA and AE are dual.

It is worthwhile observing that when $\mathscr{M}$ is completely unknown then EE coincides with EA and amounts to the LTL satisfiability problem, while each of the problems AA and AE is nothing but the LTL validity problem. On the other hand, when $\mathscr{M}$ is completely known, both EE and AE turn out to be the existential model checking problem, while EA and AA coincide with the universal model checking problem.

As mentioned in the introduction, we consider three types of possible extensions and completions of partial transition systems. Each case is a special case of an extension in terms of Definition 3, as follows: (i) **label extension**, where the states and transitions are fixed, but the labels of the states are extended by an extension of the interpretation function (cf. Definition 2); (ii) **transition extension**, where the states (and their labels) are fixed, but new transitions are added; (iii) **state extension**, where new states, with partial or complete labels, as well as transitions to and from them can be added. Clearly, (i) and (ii) can be combined, whereas (iii) subsumes (ii) and can naturally subsume (i), too.

Each of these decision problems is easily seen to be PSPACE-complete. In the cases of label extensions and transition extensions, let us non-deterministically choose a completion $\mathscr{M}'$ and existentially - respectively universally - model-check η on $\mathscr{M}'$. Since the decision problems for existential and universal model checking of LTL formulae are PSPACE complete ([17, 11]), EE and EA are NPSPACE-easy. On the other hand, they are also NPSPACE-hard, because existential - respectively universal - model checking problems are trivially polynomially reducible to them (as they are the special cases where $\mathscr{M}$ is already complete). Hence they are NPSPACE-complete, so, by Savitch theorem, they are PSPACE-complete. Consequently, their dual problems AE and AA, are also PSPACE-complete. In the case of state extensions, as shown in Section 6.3, the EE problem is easily reducible to the satisfiability of a formula produced from η and the label of the initial state, AA is again its dual, and the problems AE and EA are reducible to repeatedly solving EE, hence they are PSPACE-complete, too. Still, let us note that the decision methods developed here are practically much more efficient than brute-force, generic PSPACE-complete algorithms.

There are variants of two of the previous problems that are not decision problems but rather *model synthesis problems*. The variant of EE where, on the same input, one does not ask for a YES/NO answer, but rather for an output consisting of a total extension $\mathscr{M}'$ and a path $\pi = s_0, s_1, s_2, \ldots$ in $\mathscr{M}'$ such that its corresponding trace σ is a linear model of η, when these exists (and for a failure message otherwise) is the synthesis problem that we will denote by EE^{S}. Similarly, the variant of EA where one asks for an extension $\mathscr{M}^c$ where η is universally true is the synthesis problem denoted by EA^{S}.

4 Tableau-based algorithms for EE and EE^{S} for label extensions

Usually, tableau methods are used to determine the satisfiability of an input formula η by trying systematically to build from scratch possible models of it. Here, we adapt the method to take into account also a partial Kripke structure $\mathscr{M}$ as an additional input, the goal being to possibly construct a model of η extending $\mathscr{M}$.

Here we consider the case of partial transition systems $\mathscr{M} = (S, R^{must}, R^{may}, L)$, where $R^{must} = R^{may} = R$ and S is fixed, and focus on the case of label extensions.

The core algorithm that we propose solves both problems EE and EE^{S} (modulo some technical variations) as follows. Given as an input a partial transition system $\mathscr{M}$, a distinguished state $s_0 \in \mathscr{M}$ and an LTL formula η, the algorithm enables the computation of *all* the total extensions[1] $\mathscr{M}'$ where η is true on some path starting from s_0, in a sense explained further.

4.1 The Core Tableau-based Algorithm

We describe the algorithm step-by-step, with the help of a very simple running example.

▶ **Example 12** (Running Example). We have a partial model of an automatic subway and we want to know if it can be extended/completed according to the specification "*the doors cannot be open while the train is running, and the train will eventually run*". For the sake of simplicity, we assume that the partial model consists of only two states and we suppose that the doors are open at the initial state s_0. This partial model is represented in Figure 1. Formally the question is "*Is there an extension $\mathscr{M}'$ of the partial model $\mathscr{M}$ and a computation σ in $comput_{\mathscr{M}}(s_0)$ such that $\sigma, s_0 \models \eta_0$?*", where $\eta_0 := \mathsf{F}\, r \wedge \mathsf{G}\,(r \to \neg d)$, with d interpreted as "*the doors are open*" and r as "*the train is running*".

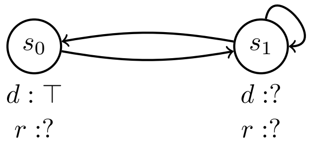


Figure 1 A partial model for Example 12.

The approach that we use to solve this problem is based on the tableau methods for LTL satisfiability, also modified for LTL model-checking, outlined in [11, Section 13.2]. Here is an outline of our procedure. First, we construct a tableau, adapted so as to take into account the partially constructed or partially observable input model. Then we use it to:

- conclude that the answer to the decision problem EE is YES, when the tableau is "open" (in a sense that will be specified further), else it is NO;
- in the case of YES, to extract an extension $\mathscr{M}'$ of $\mathscr{M}$ and a path π in it, starting from s_0, such that the corresponding computation in $\mathscr{M}^c$ satisfies η.

4.1.1 Pretableau Construction Phase

Here, as in Section 2, Prop is a set of atomic propositions. Given a partial model $\mathscr{M} = (S, R, L)$ (i.e., $R = R^{must} = R^{may}$) a state $s_0 \in S$ and an LTL formula η, we construct a graph called **pretableau of** $(\mathscr{M}, s_0, \eta)$ and denoted $\mathscr{P}^{\mathscr{M}, s_0, \eta}$. The pretableau, and then the tableaus, are directed graphs consisting of **nodes** and arrows between them. The nodes are triples

[1] To be precise, the algorithm itself does not produce the set of all such extensions – including completions of the input model – as it may leave undetermined the truth values of some atoms at states of paths that are not relevant for the existence of a path satisfying η. But, all the extensions of the input model can be trivially computed from the set of the extensions directly built via the tableau based algorithm.

(s, Γ, A_Γ) consisting of: (the name of) a state s in $\mathscr{M}$, a set of formulas Γ that must be true at that state, which we call its **description**, and an **annotation function** $A_\Gamma : S \times \mathrm{Prop} \to B$, explained further[2].

There are two types of nodes in the pretableau:

- **tableau states**[3], where the set of formulas is fully expanded;
- **prestates**, that only play an auxiliary and temporary role.

As explained in [11, Chapter 13], the sets of formulae labelling a tableau prestate are not yet fully processed. For example, a prestate component Γ may declare that a formula $p \vee q$ is true in some state s in $\mathscr{M}$ without indicating which of p and q must be true. Thus, in order to get a precise description of s, Γ needs to be expanded (via the rule later called Exp) into two alternative sets, by explicitly adding respectively p and q to them, and possibly leading to two "offspring states".

The initialisation of the algorithm produces an initial tableau prestate. Then, tableau states and prestates are built alternatively, in an iterative way described further.

▶ **Notation 13.** *Let Γ be a set of* LTL *formulas, $\mathscr{M} = (S, R, L)$ be a partial model, and let $s \in S$. We denote $\mathsf{Litt}(s, \Gamma, L)$ the set of literals l over atomic propositions appearing in Γ such that the truth value of l at s according to the input interpretation function L is $\top$. Formally, for $p \in \mathrm{Prop}$:* $\mathsf{Litt}(s, \Gamma, L) := \{p | p \in \Gamma, L(s, p) = \top\} \cup \{\neg p | p \in \Gamma, L(s, p) = \bot\}$.

4.1.2 Initialisation of the algorithm

The pretableau construction begins with the creation of the prestate $(s_0, \Gamma_0, A_{\Gamma_0})$ where $\Gamma_0 := \{\eta\} \cup \mathsf{Litt}(s_0, ecl(\eta))$ and the initial annotation function A_{Γ_0} coincides with L. Note that, in general, any propositional letter $q \in PROP \cap \mathsf{ecl}(\eta)$ such that r does not occur in η is such that neither of q and $\neg q$ is in $ecl(\eta)$, so r's truth value does not impact the satisfaction of η_0.

▶ **Example 14** (Running Example continued)**.** In Example 12 with $\eta_0 = \mathsf{F}\, r \wedge \mathsf{G}\,(r \to \neg d)$, we obtain $\mathsf{ecl}(\eta_0) = \{\eta_0, \mathsf{F}\, r, r, \mathsf{X}\,\mathsf{F}\, r, \mathsf{G}\,(r \to \neg d), r \to \neg d, \mathsf{X}\,\mathsf{G}\,(r \to \neg d), \neg r, \neg d\}$. Further, the input partial model is such that $L(s_0, d) = \top$ and $L(s_0, r) =?$, so $\mathsf{Litt}(s_0, ecl(\eta_0), L) = \emptyset$ and $\Gamma_0 := \{\eta_0\}$. Note that although $\neg r \in \mathsf{ecl}(\eta_0)$ neither r nor $\neg r$ appears in Γ_0, as its truth value is undetermined/unknown. The construction of the tableau begins with the creation of the prestate node $(s_0, \Gamma_0, A_{\Gamma_0})$ where $A_{\Gamma_0} = L$

4.1.3 The body of the algorithm

Two rules will be alternatively applied:

- Exp: producing so called **offspring states** of a given prestate,
- Next: producing **successor prestates** of a given state.

Each rule may be applied at most once to each node, to ensure termination.

The rule Exp, defined below, statically analyzes the formulas in a tableau prestate, following their semantics, and generates tableau states.

[2] An annotation function should not be confused with the interpretation function L given in the input model; it is used in a tableau node to describe and extend, incrementally, the interpretation function L given in the input model.

[3] Not to be confused with states in a partial transition system.

▶ **Definition 15** (Rule Exp). *Given a prestate (s, Γ, A_Γ), do the following:*

1. *Compute the family $FE(\Gamma)$ of all full expansions of Γ in the sense of Definition 9.*
2. *For each $\Delta \in FE(\Gamma)$ and $p \in \mathsf{prop}$ define the new annotation function A_Δ as*

$$A_\Delta(s,p) = \begin{cases} L_\Gamma(s,p) & \textit{if } L_\Gamma(s,p) \neq ? \\ \top & \textit{if } L_\Gamma(s,p) = ? \textit{ and } p \in \Delta \\ \bot & \textit{if } L_\Gamma(s,p) = ? \textit{ and } \neg p \in \Delta \\ ? & \textit{if } L_\Gamma(s,p) = ? \textit{ and } p, \neg p \notin \Delta \end{cases}$$

   and add in the current pretableau a new offspring state (s, Δ, A_Δ).
3. *For each newly introduced state (s, Δ, A_Δ), create an edge $(s, \Gamma, A_\Gamma) \dashrightarrow (s, \Delta, A_\Delta)$.*
4. *If the pretableau already contains a state (s, Δ, A_Δ), then do not create a new state but create only an edge $(s, \Gamma, A_\Gamma) \dashrightarrow (s, \Delta, A_\Delta)$ to that state.*

▶ **Notation 16.** *The set $\{(s, \Delta, A_\Delta) | (s, \Gamma, A_\Gamma) \dashrightarrow (s, \Delta, A_\Delta)\}$ of offspring states of (s, Γ, A_Γ) is denoted by* $\mathsf{states}(s, \Gamma, A_\Gamma)$*;*

▶ **Example 17** (Running example continued). We apply the rule Exp to the prestate $(s_0, \Gamma_0, A_{\Gamma_0})$ obtained in Example 14. The set of the full expansions of Γ_0 contains only one state, Δ_0, because a choice of r when expanding Γ_0 would lead to a patently inconsistent set of formulae. We have:

$$\Delta_0 := \{d, \neg r, \eta_0, \mathsf{G}\,(r \to \neg d), \mathsf{F}\,r, r \to \neg d, \mathsf{X}\,\mathsf{F}\,r, \mathsf{X}\,\mathsf{G}\,(r \to \neg d)\}.$$

Note that $\neg r \in \Delta_0$ while $L(s_0, r) = ?$.
The new annotation function A_{Δ_0} is defined by: $A_{\Delta_0}(s_0, d) = \top$ and $A_{\Delta_0}(s_0, r) = \bot$.
This is the beginning of the update of the input interpretation function in $\mathscr{M}$.

The currently constructed pretableau is given in Figure 2.

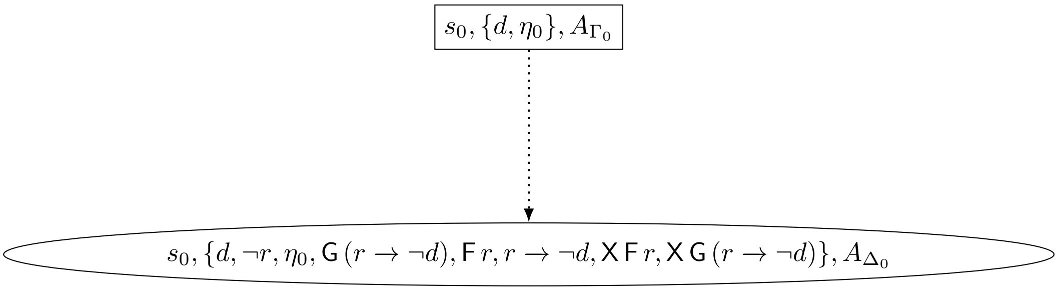

$s_0, \{d, \eta_0\}, A_{\Gamma_0}$
$s_0, \{d, \neg r, \eta_0, \mathsf{G}\,(r \to \neg d), \mathsf{F}\,r, r \to \neg d, \mathsf{X}\,\mathsf{F}\,r, \mathsf{X}\,\mathsf{G}\,(r \to \neg d)\}, A_{\Delta_0}$

Figure 2 The pretableau construction stage after applying the rule Exp to the initial prestate, for Example 17.

The rule Next creates in the tableau graph successors of any *tableau state* (s, Δ, A_Δ) corresponding to a *state* s in input model $\mathscr{M}$. (NB the distinction between states in a model, which are semantic objects, and tableau states, which are syntactic objects.)

▶ **Definition 18** (Rule Next). *Given a state (s, Δ, A_Δ), do the following:*

- *If there are no R-successors of s in the partial model, remove the state (s, Δ, A_Δ) from the pretableau. Else:*
  - *For every R-successor state t of s in the current partial model, add in the current pretableau a successor prestate $(t, Scomp(\Delta) \cup \mathsf{Litt}(t, ecl(\eta), A_\Delta))$ of (s, Δ, A_Δ)*
  - *If the pretableau-graph already contains a node $V = (t, \Gamma, A_\Gamma)$ that is then do not create a new state but create only an edge $(s, \Delta, A_\Delta) \to (t, \Gamma, A_\Gamma)$ to V.*

▶ **Example 19** (Running example continued)**.** We apply the rule Next to the offspring state $(s_0, \Delta_0, A_{\Delta_0})$ obtained in the example 17. There is only one R-successor of s_0 that is s_1 in our partial model. So, the rule Next generate the only prestate $(s_1, \Gamma_1, A_{\Gamma_1})$, where $\Gamma_1 := \{\mathsf{F}\, r, \mathsf{G}\,(r \to \neg d)\}$. We have that $A_{\Gamma_1} = A_{\Delta_0}$(only the rule Exp can modify an annotation function). The current pretableau is given in Figure 3.

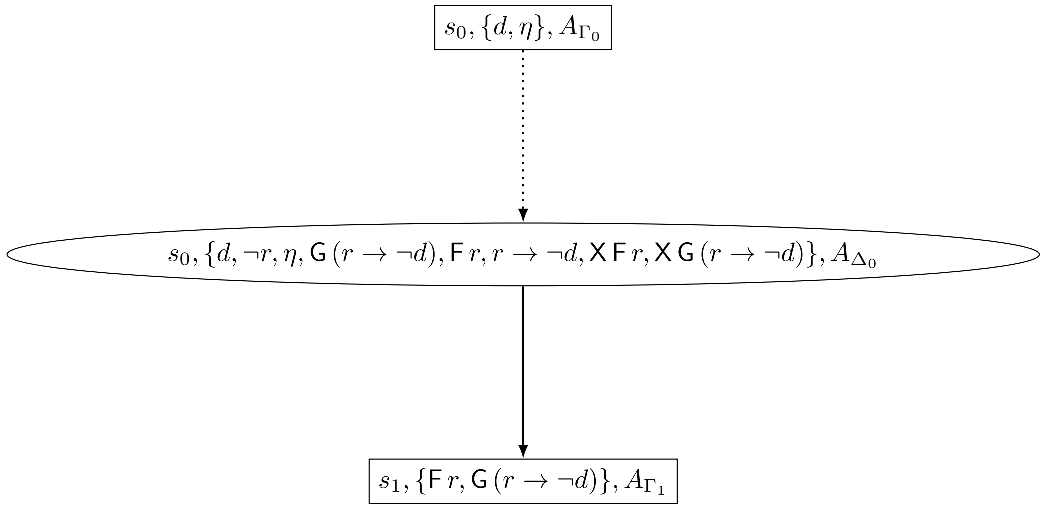

$s_0, \{d, \eta\}, A_{\Gamma_0}$
$s_0, \{d, \neg r, \eta, \mathsf{G}\,(r \to \neg d), \mathsf{F}\, r, r \to \neg d, \mathsf{X}\,\mathsf{F}\, r, \mathsf{X}\,\mathsf{G}\,(r \to \neg d)\}, A_{\Delta_0}$
$s_1, \{\mathsf{F}\, r, \mathsf{G}\,(r \to \neg d)\}, A_{\Gamma_1}$

Figure 3 The pretableau construction stage for Example 19.

▶ **Lemma 20.** *The pretableau construction phase terminates.*

Proof. For any pretableau node $\langle s, \Sigma, A\rangle$ there are only finitely many annotation functions A (since both S and P are finite) and $\Sigma \subseteq \mathsf{ecl}(\eta)$, where $\mathsf{ecl}(\eta)$ is finite. Now, it suffices to note that both the rules Exp and Next disallow duplication of already existing nodes. ◀

▶ **Example 21** (Running example continued)**.** At the end of the construction phase we obtain the pretableau shown in the in the Figure 4, where prestates are indicated with rectangular boxes and states with oval boxes. Nodes of the pretableau are enumerated to help reading the picture, but we leave it to the interested reader to flesh out the content of each node.

4.1.4 Prestate and State Elimination Phase

First, we remove all prestates from the pretableau with their incoming and outgoing edges, by applying the following **prestate elimination rule**:

▶ **Definition 22** (Rule PrestateElim)**.** *For every prestate (s, Γ, A_Γ) in the pretableau, do:*

1. *Remove (s, Γ, A_Γ) from the pretableau;*
2. *If there is a state (t, Δ, A_Δ) in the pretableau with $(t, \Delta, A_\Delta) \to (s, \Gamma, A_\Gamma)$, then for every state $(s, \Delta', A'_\Delta) \in \mathsf{states}(s, \Gamma, A_\Gamma)$ create an edge $(t, \Delta, A_\Delta) \to (s, \Delta', A'_\Delta)$.*

The resulting graph is called the ***initial tableau*** *for $(\mathscr{M}, s_0, \eta)$, denoted by $\mathscr{T}_0^{\mathscr{M}, s_0, \eta}$.*

Once all prestates have been eliminated, we must remove all "wrong" states from the tableau (dead ends and roots of paths that *never realize an eventuality*, which is explained further) by applying the rules StateElim1 and StateElim2, formulated shortly.

In what follow, we denote $\mathscr{T}_n^{\mathscr{M}, s_0, \eta}$ the tableau obtained from the initial tableau after n applications of an elimination rule StateElim1 or StateElim2.

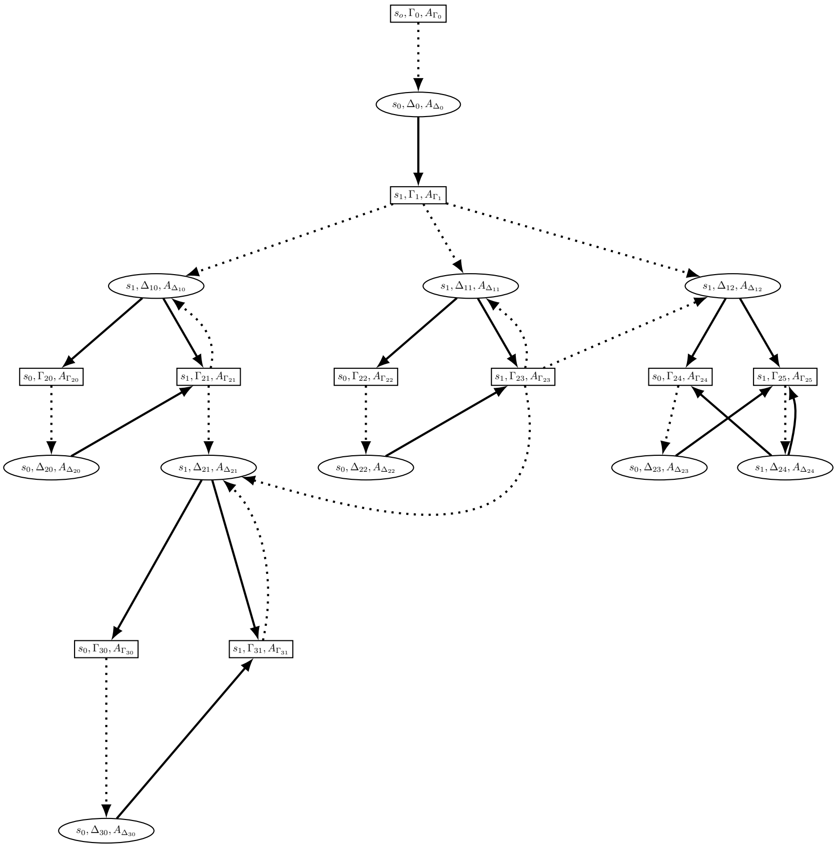

$s_o, \Gamma_0, A_{\Gamma_0}$
$s_0, \Delta_0, A_{\Delta_0}$
$s_1, \Gamma_1, A_{\Gamma_1}$
$s_1, \Delta_{10}, A_{\Delta_{10}}$
$s_1, \Delta_{11}, A_{\Delta_{11}}$
$s_1, \Delta_{12}, A_{\Delta_{12}}$
$s_0, \Gamma_{20}, A_{\Gamma_{20}}$
$s_1, \Gamma_{21}, A_{\Gamma_{21}}$
$s_0, \Gamma_{22}, A_{\Gamma_{22}}$
$s_1, \Gamma_{23}, A_{\Gamma_{23}}$
$s_0, \Gamma_{24}, A_{\Gamma_{24}}$
$s_1, \Gamma_{25}, A_{\Gamma_{25}}$
$s_0, \Delta_{20}, A_{\Delta_{20}}$
$s_1, \Delta_{21}, A_{\Delta_{21}}$
$s_0, \Delta_{22}, A_{\Delta_{22}}$
$s_0, \Delta_{23}, A_{\Delta_{23}}$
$s_1, \Delta_{24}, A_{\Delta_{24}}$
$s_0, \Gamma_{30}, A_{\Gamma_{30}}$
$s_1, \Gamma_{31}, A_{\Gamma_{31}}$
$s_0, \Delta_{30}, A_{\Delta_{30}}$

Figure 4 The complete pretableau for the running example 21.

▶ **Definition 23** (Rule StateElim1). *If a state (r, Δ, A_Δ) has no successor states in the current tableau, then remove (r, Δ, A_Δ) from the tableau.*

In order to formulate StateElim2 we need the following definition.

▶ **Definition 24** (Realization of an eventuality at a tableau state). *An eventuality $\varphi \mathsf{U} \psi$ (resp. $\mathsf{F}\psi$) is realized at the state (s, Δ, A_Δ) in $\mathcal{T}_n^{\mathcal{M}, s_0, \eta}$ if there exists a finite path in $\mathcal{T}_n^{\mathcal{M}, s_0, \eta}$*

$$\Pi = (s, \Delta, A_\Delta), (s_{i_1}, \Delta_{j_1}, A_{\Delta_{j_1}}), \ldots, (s_{i_m}, \Delta_{j_m}, A_{\Delta_{j_m}})$$

where $m \geq 0$, such that

- *$\varphi \mathsf{U} \psi, \psi \in \Delta_{j_m}$ (resp. $\mathsf{F}\psi, \psi \in \Delta_{j_m}$);*
- *$\varphi \mathsf{U} \psi, \varphi \in \Delta_{j_i}$ (resp. $\mathsf{F}\psi \in \Delta_{j_i}$) for every $i = 0, \ldots, m-1$.*

We say that $\varphi \mathsf{U} \psi$ (resp. $\mathsf{F}\psi$) is realized on the path Π, and we call any such path a witness of the realization of the eventuality $\varphi \mathsf{U} \psi$ (resp. $\mathsf{F}\psi$) in Δ. If $\psi \in \Delta$, then we say that $\varphi \mathsf{U} \psi$ (resp. $\mathsf{F}\psi$) is immediately realized at the state (s, Δ, A_Δ) (on the path constituted by the singleton (s, Δ, A_Δ)).

▶ **Definition 25** (Rule StateElim2). *If an eventuality $\alpha \in \Delta$ is not realised on any path starting from (s, Δ, A_Δ) in the current tableau, then remove (s, Δ, A_Δ) from the tableau.*

The state elimination phase is carried out in a sequence of stages, starting at stage 0 with the initial tableau $\mathcal{T}_0^{\mathcal{M}, s_0, \eta}$, and eliminating at every stage n at most one state for the current tableau $\mathcal{T}_n^{\mathcal{M}, s_0, \eta}$ – by applying one of the state elimination rules – , to produce the new current tableau $\mathscr{T}_{n+1}^{\mathscr{M},s_0,\eta}$. When the state elimination phase reaches a stabilisation point, *i.e.* no more nodes are removed, the current tableau at that stage is called **final tableau** for $(\mathscr{M}, s_0, \eta)$ and denoted by $\mathscr{T}^{\mathscr{M},s_0,\eta}$.

▶ **Example 26** (Running example continued)**.** Figure 5 shows the initial tableau, where only prestates have been eliminated. All the states here have at least one successor state so the rule StateElim1 can not be applied. However there is a state with description component $\{d, \neg r, \mathsf{F}\, r, \mathsf{X}\,\mathsf{F}\, r, \mathsf{G}\,(r \to \neg d), r \to \neg d, \mathsf{X}\,\mathsf{G}\,(r \to \neg d\})$ shown in Figure 6.

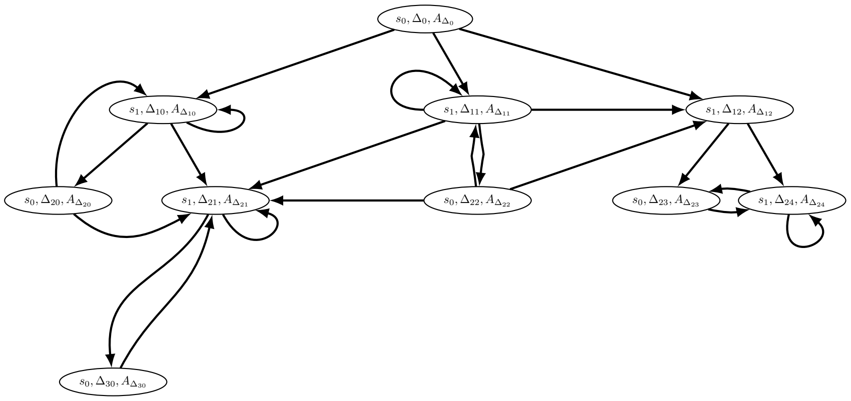

$s_0, \Delta_0, A_{\Delta_0}$
$s_1, \Delta_{10}, A_{\Delta_{10}}$
$s_1, \Delta_{11}, A_{\Delta_{11}}$
$s_1, \Delta_{12}, A_{\Delta_{12}}$
$s_0, \Delta_{20}, A_{\Delta_{20}}$
$s_1, \Delta_{21}, A_{\Delta_{21}}$
$s_0, \Delta_{22}, A_{\Delta_{22}}$
$s_0, \Delta_{23}, A_{\Delta_{23}}$
$s_1, \Delta_{24}, A_{\Delta_{24}}$
$s_0, \Delta_{30}, A_{\Delta_{30}}$

Figure 5 The initial tableau for the running example.

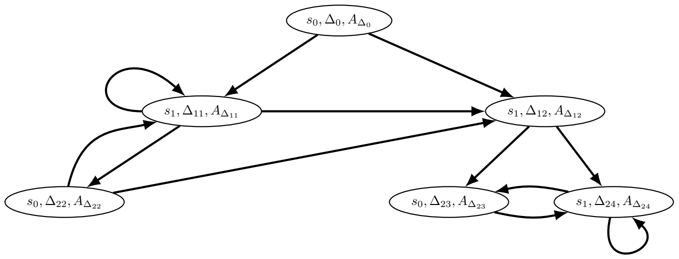

$s_0, \Delta_0, A_{\Delta_0}$
$s_1, \Delta_{11}, A_{\Delta_{11}}$
$s_1, \Delta_{12}, A_{\Delta_{12}}$
$s_0, \Delta_{22}, A_{\Delta_{22}}$
$s_0, \Delta_{23}, A_{\Delta_{23}}$
$s_1, \Delta_{24}, A_{\Delta_{24}}$

Figure 6 The final tableau for the running example 26.

▶ **Definition 27.** *The final tableau $\mathscr{T}^{\mathscr{M},s_0,\eta}$ is* ***open*** *if it contains at least one state $(s_0, \Delta_0, A_{\Delta_0})$ such that $\eta \in \Delta_0$. Otherwise it is* ***closed****.*

4.2 Specialized algorithms for EE and EE$^{\mathsf{S}}$

The tableau-based core algorithm that we have described in Section 4.1 can be refined and modified so as to obtain two algorithms, solving respectively the problem EE and the problem EE$^{\mathsf{S}}$. In both cases the input is given by a partial model $\mathscr{M} = (S, R, L)$, a distinguished state s_0 and an LTL formula η. However the outputs are different. The initial procedure is always the same: apply the core algorithm to construct a tableau $(T)^{\mathscr{M},s_0,\eta}$.

4.2.1 Algorithm for EE

The output for the decision problem EE is either *YES* or *NO*. Once applied the core algorithm to construct a tableau $(T)^{\mathscr{M},s_0,\eta}$, the output is *YES* if and only if $(T)^{\mathscr{M},s_0,\eta}$ is open.

4.2.2 Algorithm for EE^{S}

We give two versions of the algorithm. The first version is *non-deterministic*, and outputs just an extension of the input model $\mathscr{M}$ and a path in it (whenever some exist) that satisfies the specification formula η. The second version (useful in Section 6) is *deterministic* and produces *all possible ways to extend* the input model so as to existentially satisfy η, and thus enables the generation of all such extensions, which we will use in Section 6.

Non-deterministic version ND-EE$^{\mathsf{S}}$. Here the output is either $\emptyset$ (when $\mathscr{T}^{\mathscr{M},s_0,\eta}$ is closed) or else a singleton containing a pair $(\mathscr{M}' = (S, R, L'), \pi)$ where $\mathscr{M}'$ is an extension of the input model and π a path in it starting at s_0 and satisfying η.

1. Construct the tableau $\mathscr{T}^{\mathscr{M},s_0,\eta}$.
2. If the final tableau is closed, then return the empty set. Else:
   a. Non-deterministically choose a path $(s_0, \Delta_0, A_{\Delta_0}), (s_1, \Delta_1, A_{\Delta_1}), \dots$ in $\mathscr{T}^{\mathscr{M},s_0,\eta}$ that starts from a root of the tableau (note that it may have several roots) and such that the trace $\Delta_0, \Delta_1, \dots$ is a Hintikka trace (as in Definition 10).
   (When the final tableau is open such a Hintikka trace exists, as we prove in Section 5.)
   b. Define L' as follows:
      - if $A_{\Delta_n}(s,p) = \top$ for some node $(s_n, \Delta_n, A_{\Delta_n})$ on the chosen path, then $L'(s,p) := \top$;
      - else:
        - if $A_{\Delta_n}(s,p) = \bot$ for some node $(s_n, \Delta_n, A_{\Delta_n})$ on the chosen path then $L'(s,p) := \bot$;
        - else set $L'(s,p) := ?$.
   c. Return $\mathscr{M}' = (S, R, L')$ and the path $s_0, s_1, s_2, \dots$.

Deterministic version D-EE$^{\mathsf{S}}$. Here the output is either the empty set (when $\mathscr{T}^{\mathscr{M},s_0,\eta}$ is closed) or else a set of pairs $(\mathscr{M}', \sigma)$, such that $\mathscr{M}'$ is a total extension of $\mathscr{M}$ and σ is a path in it starting at s_0 and satisfying η. Intuitively, the algorithm can eventually produce all possible ways to extend the input model so as to existentially satisfy η.

1. Construct the tableau $\mathscr{T}^{\mathscr{M},s_0,\eta}$.
2. If the tableau is closed, then return the empty set.
3. Else, let $paths(\mathscr{T}^{\mathscr{M},s_0,\eta})$ be the set of all paths in $\mathscr{T}^{\mathscr{M},s_0,\eta}$ starting from a root node such that the sequence of their description components is a Hintikka trace (Def. 10):

   $$paths(\mathscr{T}^{\mathscr{M},s_0,\eta}) := \{(s_0, \Delta_0, A_{\Delta_0}), (s_1, \Delta_1, A_{\Delta_1}), \dots \mid \Delta_0, \Delta_1, \dots \text{ is a Hintikka trace}\}$$

   For every tableau path π in $paths(\mathscr{T}^{\mathscr{M},s_0,\eta})$, define $\sigma_\pi = s_0, s_1, s_2, \dots$ and the set $LC(\sigma_\pi)$ as the set of *all* the interpretation functions $L^c{}_{\sigma_\pi}$ such that:
   - if $A_{\Delta_n}(s,p) = \top$ for some node $(s_n, \Delta_n, A_{\Delta_n})$ of π, then $L^c{}_{\sigma_\pi}(s,p) = \top$;
   - if $A_{\Delta_n}(s,p) = \bot$ for some node $(s_n, \Delta_n, A_{\Delta_n})$ of π, then $L^c{}_{\sigma_\pi}(s,p) = \bot$.
   - If $A_{\Delta_n}(s,p) = ?$ for every node $(s_n, \Delta_n, A_{\Delta_n})$ of π, then we have two possibilities: $L^c{}_{\sigma_\pi} = \top$ and $L^c{}_{\sigma_\pi} = \bot$, hence here the construction of $L^c{}_{\sigma_\pi}$ branches and two different interpretation functions are produced.

   It is now easy to define a sub-procedure that generates, path by path, the elements of the set of all couples formed by a total extension $\mathscr{M}^c$ of the input partial model that is induced by an element of $paths(\mathscr{T}^{\mathscr{M},s_0,\eta})$ and a path in it, namely the set:

   $$\{((S, R, L'), \sigma_\pi) \mid \pi \in paths(\mathscr{T}^{\mathscr{M},s_0,\eta}), L' \in LC(\sigma_\pi)\}.$$

   Applying such a sub-procedure will eventually generate the full set described above.

▶ **Example 28** (End of the running example)**.** The final tableau for the running example of this section is the one in Figure 6. This tableau is open, so the partial model $\mathscr{M}$ given in Example 12 can be completed so as to show that a path starting from s_0 and satisfying η exists. For instance, one can take the path $s_0, s_1, s_1, s_1, \ldots$ in the complete transition system (S, R, L') where $L' = A_{\Delta_{1.1}}$ is such that:

- $L'(s_0, d) = \top$
- $L'(s_0, r) = \bot$
- $L'(s_1, d) = \bot$
- $L'(s_1, r) = \top$

Indeed the tableau path $(s_0, \Delta_0, A_{\Delta_0}), (s_1, \Delta_{12}, A_{\Delta_{12}}), (s_1, \Delta_{24}, A_{\Delta_{24}}), (s_1, \Delta_{24}, A_{\Delta_{24}}), \ldots$ corresponds to the Hintikka trace $\Delta_0, \Delta_{12}, \Delta_{24}, \Delta_{24}, \ldots$ where
$\Delta_0 := \{d, \neg r, \eta, \mathsf{G}\,(r \to \neg d), \mathsf{F}\,r, r \to \neg d, \mathsf{X}\,\mathsf{F}\,r, \mathsf{X}\,\mathsf{G}\,(r \to \neg d)\}$,
$\Delta_{12} := \{\neg d, r, \mathsf{F}\,r, \mathsf{G}\,(r \to \neg d), r \to \neg d, \mathsf{X}\,\mathsf{G}\,(r \to \neg d)\}$,
$\Delta_{24} := \{\neg d, r, \mathsf{G}\,(r \to \neg d), r \to \neg d, \neg d, \mathsf{X}\,\mathsf{G}\,(r \to \neg d))\}$.

The following result is an immediate consequence of Lemma 20:

▶ **Theorem 29.** *The algorithms for* EE *and for* EE$^{\mathsf{S}}$ *(in both versions) terminate.*

5 Soundness and Completeness Results

▶ **Theorem 30.** *Let $\mathscr{M} = (S, R, L)$ be a partial model, let $s_0 \in S$ be an initial state and let η be an* LTL *formula, such that $\mathscr{T}^{\mathscr{M}, s_0, \eta}$ is an open tableau. Then the following hold:*

1. Soundness with respect the existence of a solution for EE.
   The problem EE *has an answer YES.*
2. Appropriate tableau paths describe solutions for EE.
   Let $\pi = (s_0, \Delta_0, A_0), (s_1, \Delta_1, A_1), (s_2, \Delta_2, A_2), \ldots$ be any path in $\mathscr{T}^{\mathscr{M}, s_0, \eta}$ such that $\Delta_0, \Delta_1, \Delta_2, \ldots$ is a Hintikka trace. Then for any extension $\mathscr{M}' = (S, R, L')$ of $\mathscr{M}$ such that $A_n \preceq L'$ for each n, the computation σ in $\mathscr{M}' = (S, R, L')$ corresponding to $s_0, s_1, s_2, \ldots$ satisfies η.

Proof.

1. Here, intuitively we show that *any* open tableau $\mathscr{T}^{\mathscr{M}, s_0, \eta}$ necessarily contains at least *some path* π that "describes" *some* solution to EE.
   First, we inductively construct an infinite chain $\mathscr{S}$ of initial sub-paths in the tableau:
   $\pi_0 = (s_0, \Delta_0, A_0)$,
   $\pi_1 = (s_0, \Delta_0, A_0), (s_1 \Delta_1, A_1)$,
   $\pi_2 = (s_0, \Delta_0, A_0), (s_1, \Delta_1, A_1), (s_2, \Delta_2, A_2)$...
   where each π_k is a finite initial sub-path of the next one, as follows:
   - π_0 consists of the initial state node (s_0, Δ_0, A_0), where $\eta \in \Delta_0$ and $A_0 = L$.
   - Let π_k be the last constructed sub-path and let $V = (s_k, \Delta_k, A_k)$ be its last node. Let E be the set of eventualities in Δ_k.
     - If E is non-empty, let $e = \varphi \,\mathsf{U}\, \psi$ be one of them. Since rule StateElim2 has not removed V, there is a finite sub-path π_{sub} in the tableau which starts at V and ends at a node $(s_{k+m}, \Delta_{k+m}, A_{k+m})$, for some $m \geq 0$, such that $\varphi \,\mathsf{U}\, \psi, \psi \in \Delta_{k+m}$. Let us observe that if $e' = \varphi' \,\mathsf{U}\, \psi'$ is another eventuality in E, and ψ' does not appear in (the description component of) any node of π_{sub}, then each node in such a sub-path will be such that its Δ_l component has $\varphi' \,\mathsf{U}\, \psi'$ as an element (by tableau construction). We then build the new sub-path π_{k+1} by appending π_{sub} to π_k.

- If E is empty, then let V' be any successor of V in the tableau $\mathscr{T}^{\mathscr{M},s_0,\eta}$ (by rule StateElim1 each tableau path is infinite). We extend π_k by appending V' to it, thereby obtaining π_{k+1}. Let us observe that this case necessarily eventually occurs, because eventuality formulae in $\mathscr{T}^{\mathscr{M},s_0,\eta}$ are elements of $\mathsf{ecl}(\eta)$, that is finite.

Let π be the limit tableau path of the sequence $\mathscr{S}$. The sequence of the description components of the vertices in π is a Hintikka trace (see Definition 10). By construction, each node $(s_i, \Delta_i, A_i) \in \pi$ is such that $L \preceq A_i$, and for some $n \geq 0$ the annotation function becomes stable, that is, for any $m \geq n$, $A_m = A_n$.

Now let $\mathscr{M}'$ be the extension of $\mathscr{M}$ obtained by replacing L by A_n. It is easy to show that every state s_i in $\mathscr{M}'$ satisfies all the formulae in Δ_i. In particular s_0 satisfies all formulae in Δ_0, in particular η. Therefore the computation corresponding to the path $s_0, s_1, s_2, ...$ generated by π is a linear model of η, and the answer to EE is YES.

2. Now we must show that *any* tableau path $\pi = (s_0, \Delta_0, A_0), (s_1, \Delta_1, A_1), (s_2, \Delta_2, A_2), \ldots$ where the corresponding description components constitute a Hintikka trace describes a linear model σ of η for some extension $\mathscr{M}' = (S, R, L')$ of $\mathscr{M}$ (actually for a class of extensions). A computation σ in $\mathscr{M}' = (S, R, L')$ corresponding to $s_0, s_1, s_2, ...$ is defined so that, for any $i \geq 0$, $L'(s_i) = \mathrm{Prop} \cap \Delta_i$. For any i, we have $\sigma, i \models \Delta_i$, thus, in particular, we have $\sigma, 0 \models \Delta_0$, and $\eta \in \Delta_0$, which implies that σ is a linear model of η. ◀

The completeness result below shows not only that when the answer to EE is YES due to some extension $\mathscr{M}'$ then the corresponding tableau is open, but a stronger result, which intuitively says that each computation in $\mathscr{M}'$ satisfying η is described by a path of such a tableau. We make use of this stronger result in Section 6.

▶ **Theorem 31** (Completeness, stronger version).

1. All solutions to EE are described by tableau paths.
   Let $\mathscr{M} = (S, R, L)$ be a partial transition system that is transition-complete, let η be an LTL *formula and let $s_0 \in S$. Let $\mathscr{M}' = \{S, R, L\}$ be any total extension of $\mathscr{M}$ and let $s_0, s_1, s_2, \ldots$ be a path in $\mathscr{M}'$ such that its corresponding computation σ satisfies η.*
   Then any final tableau $\mathscr{T}^{\mathscr{M},s_0,\eta}$ contains a path $\pi = (s_0, \Delta_0, A_0), (s_1, \Delta_1, A_1), (s_2, \Delta_2, A_2), \ldots$ such that for each n we have $A_n \preceq L'$.
2. Completeness with respect to the existence of a solution for EE.
   If a solution to EE *exists than $\mathscr{T}^{\mathscr{M},s_0,\eta}$ is open.*

Proof.

1. To prove this item we build π inductively, so that each step of the construction satisfies the following invariants:
   a. For any i, $A_i \preceq L'$.
   b. For any tableau state (s_i, Δ_i, A_i) in π, we have $\sigma, i \models \Delta_i$.

   Here is the construction. The initial node is the state $(s_0, \Delta_0, A_{\Delta_0})$ defined as follows. The state description Δ_0 is obtained by applying Exp to the original prestate $(s_0, \Gamma_0, A_{\Gamma_0})$ where $\Gamma_0 := \{\eta\} \cup \mathsf{Litt}(s_0, ecl(\eta))$ and the initial annotation function A_{Γ_0} coincides with L. By the assumption that σ satisfies η and by definition of $\mathsf{Litt}(s_0, ecl(\eta))$, we have that $\sigma, 0 \models \Gamma_0$. Then we choose Δ_0 to be a full expansion of Γ_0 such that $\sigma, 0 \models \Delta_0$. The annotation function A_{Δ_0} is determined by Δ_0, and by construction obviously $A_{\Delta_0} \preceq L'$.

   Now, suppose the sub-path π_j of $\pi = (s_0, \Delta_0, A_0), \cdots, (s_j, \Delta_j, A_j)$ has been already defined. If Δ_j contains a still pending eventuality $e = \varphi \,\mathsf{U}\, \psi$ by the hypothesis on σ we have $\sigma, j \models e$. Therefore there is finite initial segment of σ, say $s_0, \cdots s_j, s_{j+1}, \cdots s_k$, for some

$k > j$, such that $\sigma, k \models \psi$ and for any r in $[j, ..., k-1]$ we have $\sigma, r \models \varphi$. Then the tableau sub-path $\pi_k = (s_0, \Delta_0, A_0), \cdots, (s_j, \Delta_j, A_j), (s_{j+1}, \Delta_{j+1}, A_{j+1}) \cdots, (s_k, \Delta_k, A_k)$ can be obtained by choosing in the suitable way (via applications of Next) the corresponding prestates $(s_{j+1}, \Gamma_{j+1}, A_{j+1}), ..., (s_k, \Gamma_k, A_k)$ in the pretableau, and choosing the suitable full expansions of their Γ-components. It is worthwhile observing that each given Γ_{j+i} component determines its corresponding annotation function A_i (while Next does not produce updates of the annotation function). If Δ_j contains no pending eventuality then any tableau sub-path having π_j has initial sub-path will do.

This completes the inductive definition of π. Let us observe that the invariants 1a and 1b hold for π and a third property holds: any eventuality e in a description component Δ_i is realized on a finite sub-path of π. These three properties of π imply that no state in π is removed by any application of the rules StateElim1 or StateElim2. In fact, a routine induction on the number of rounds in the elimination phase suffices to prove this claim. Hence the defined π survives to the elimination procedure and, as required, for each n we have $A_n \preceq L'$.

2. This claim of the theorem follows from the previous one. ◀

▶ **Corollary 32** (Soundness and completeness of the tableau). *Let $\mathscr{M} = (S, R, L)$ be a partial model, $s_0 \in S$ a state in it and η an* LTL *formula. Then the tableau $\mathscr{T}^{\mathscr{M}, s_0, \eta}$ is open if and only if the answer to the problem* EE *is YES.*

6 Harvest: solving all problems

So far we have only considered the problems EE and EE^{S} in the case of label extensions and have developed methods for their solution. Here we will adapt our methods to solve all problems defined in Section 3 for all cases of extensions that we study. For lack of space, here we leave out the routine details and only outline the adapted procedures.

6.1 The case of label extensions

Since our approach solves EE for label extensions, it clearly also solves the corresponding dual problem AA with input $(\mathscr{M}, s_0, \eta)$, by calling the algorithm for EE on the input $(\mathscr{M}, s_0, \neg\eta)$.

Now, for EA and its dual problem AE, we proceed as follows.

1. First, we apply the algorithm for AA with the input $(\mathscr{M}, s_0, \eta)$. If the answer to AA is YES, then so are also the answers to EA and AE and we are done.
2. In the case the answer to AA is NO, we call the algorithm for EE on $(\mathscr{M}, s_0, \neg\eta)$.
   a. If the returned answer is NO, return YES for both EE and AE.
   b. If the answer is YES, an application of algorithm D-EE^{S} (Section 4.2.2) on $(\mathscr{M}, s_0, \neg\eta)$ generates *all* total extensions where $\neg\eta$ is existentially true at s_0 (Thm. 31, claim 1).
      - If there is at least one (amongst the finitely many) total extension $\mathscr{M}'$ of $\mathscr{M}$ that is not returned by D-EE^{S}, this means that $\neg\eta$ is not existentially true in $\mathscr{M}'$ at s_0. Hence, η is universally true in $\mathscr{M}^c$ at s_0, therefore the answer to EA is YES.
      - Else, if every possible total extension of $\mathscr{M}$ is returned, then the answer of AE on $(\mathscr{M}, s_0, \neg\eta)$ is YES and EA returns NO.

6.2 The case of transition extensions

Now, we will adapt our methods to the case of transition extensions. (We can just as well consider the more general case of combined transition and label extensions, but the case of transition extensions is representative enough for that more general case.) First, the tableau based method presented in Section 4.1 is modified to solve EE and EE^{S} in this case, as follows.

- The pretableau construction phase starts with a partial model $\mathscr{M} = (S, R^{must}, R^{may}, L)$, a state $s_0 \in S$ and an LTL formula η and aims to construct a pretableau $\mathscr{P}^{\mathscr{M},s_0,\eta}$.
- The main difference is that, whenever the rule Next is applied to a given pretableau state (s, Δ, A_Δ), the first (state) components of the successor prestates are selected amongst all R^{may}-successors of s in $\mathscr{M}$. In the case of the EE problem, one such prestate is guessed, or selected non-deterministically, and added to the pretableau. In the case of the EE^{S} problem, all such prestates are identified and added to the pretableau.
  (In order to keep the method more economical in terms of the number of added transitions of the produced extension, in case of several possible successor prestates those corresponding to R^{must}-successors of s may be selected with priority.)
- Thereafter, the prestate elimination and state elimination phases, as well as the analysis of the outcome, being the final tableau, proceed essentially the same way as before.

Now, the problem EE^{S} is solved just like in the case of label extensions, with an algorithm generating all (finitely many) total transition extensions satisfying η. Then, the problems AA, EA, and AE are solved essentially in the same way as for label extensions.

6.3 The case of state extensions

Finally, we consider the most general case, of state extensions. Note that it subsumes the case of transition extensions, and it can also naturally incorporate the case of label extensions, where states with partial labels may be added. The EE problem now takes as an input $\mathscr{M} = (S, R^{must}, R^{may}, L)$, a state $s_0 \in S$ and an LTL formula η and asks for a total extension of $\mathscr{M}$ with new states and transitions to them that has a path starting at s_0 and satisfying η.

Note that this problem is easily reducible to the LTL satisfiability problem as follows.

Let $Var(\eta)$ be the set of all propositional variables occurring in η, and let $l_{\mathscr{M}}(s_0, \eta) := \bigwedge \mathsf{Litt}(s_0, Var(\eta), L) = \bigwedge\{p | p \in Var(\eta), L(s_0, p) = \top\} \wedge \bigwedge\{\neg p | p \in Var(\eta), L(s_0, p) = \bot\}$.

Then $l_{\mathscr{M}}(s_0, \eta)$ is true at s_0 in $\mathscr{M}$ and in any extension of $\mathscr{M}$ of the most general type. Therefore, a path starting at s_0 in any such extension satisfies η iff it satisfies $\eta \wedge l_{\mathscr{M}}(s_0, \eta)$. On the other hand, any linear LTL model σ that satisfies $\eta \wedge l_{\mathscr{M}}(s_0, \eta)$ can be "grafted" to $\mathscr{M}$ as a path starting at s_0, i.e. all states of σ after s_0 can be added as new states to $\mathscr{M}$, and the resulting state extension of $\mathscr{M}$ will be a solution of the problem EE. Thus, EE has a solution iff $\eta \wedge l_{\mathscr{M}}(s_0, \eta)$ is satisfiable. Furthermore, it suffices to consider only state extensions of $\mathscr{M}$ obtained by grafting at s_0 "small satisfiability witnesses" of the satisfiability of $\eta \wedge l_{\mathscr{M}}(s_0, \eta)$, *i.e.*, ultimately periodic linear models of size exponentially bounded by the length of $\eta \wedge l_{\mathscr{M}}(s_0, \eta)$ (cf. [11, Section 6.3]). So, the problem is clearly decidable, and can be solved in PSPACE, just like the satisfiability problem for LTL.

In a sense, this observation all but relatively trivialises the problem EE in the case of state extensions. If, however, a "minimal" state extension solving the problem EE is sought, in the sense of adding the least possible number of additional states, then the problem becomes non-trivial again, in terms of its practical complexity. We leave the search for practically optimal algorithms to future work.

Now, the problem AA is again solved as dual to EE. To solve the AE and EA problems, we would need a procedure generating and checking all (infinitely many!) state + transition extensions of $\mathscr{M}$ for existential, respectively universal truth of η. That task, however, can be reduced to a finitary one, as follows. Consider the EA problem. To solve it, it suffices to look for a total state extension of η solving the problem EA which is *minimal* in the sense of only grafting at each existing dead-end state an ultimately periodic path of sufficiently "small size" (lengths of the prefix and of the period, to be guessed, according to the theory of LTL satisfiability, cf. e.g., [11, Chapter 6]) initially with labels assigning ? to all atomic propositions, and then solving the EA problem for a *label extension* on the resulting state extension. If a solution exists for some suitable guesses of the sizes of the appended ultimately periodic paths, then the initial EA problem for (total) state extensions has a solution, too. Otherwise, that problem has no solution. Indeed, if there is any such solution, then there would be one of the type described above for suitable choices of the sizes of the appended ultimately periodic paths, because of the "small satisfiability witness" property of LTL (cf. [11, Section 6.3]). The solution of the EA problem also solves the AE problem by duality.

7 Concluding remarks: summary, related work, and future work

7.1 Summary

In this paper we have formulated four decision problems concerning the possibility of extending and completing in three natural possible ways a partially defined or partially known transition system $\mathscr{M}$ which is supposed to satisfy a specification expressed by an LTL formula. We have also considered two variations of the first two problems, that are model synthesis problems. We have proposed tableau based algorithms that provably solve the core problems EE and EE^{S} and we have shown how these algorithms can also be used to solve the remaining problems.

7.2 Related Work

Here we mention and briefly discuss some previous publications that are related, at least in terms of the framework and problems they consider, to our work.

One essentially related work is [20], although the formal tools used there (partially labeled transition systems (PTS) and the logic FLTL) are different. Indeed, in that work PTS are labelled transition systems where states are not labelled by propositions (expressing static atomic properties), but, rather, transitions (edges) are labelled by actions; some actions (resp. transitions), however, are forbidden. As in our work, the authors are concerned with explicitly modelling those aspects of system behaviour that are still unknown, because knowing such gaps can help to improve incremental system modelling. Compared to the notion of PTS in [20], our notion of partial transition system allows for the expression of two distinct kind of incomplete knowledge: of static properties of individual states as well of transitions between states. Another specificity of our approach is that it allows for the precise formulation of six different completion problems and establishes a unified framework where the classical decisions problems of satisfiability, validity, existential model-checking and universal model-checking for LTL formulae are just specific cases.

In [13] two distinct, though related, problems are studied, both named *Generalized Model checking* (LTL GMC). The first problem had already been introduced in [9] and amounts to deciding, given a transition system $\mathscr{M}$ where the state labels are 3-valued, and an LTL formula η, whether there exists a 2-valued partial transition system $\mathscr{M}'$ that is "more complete" than $\mathscr{M}$ and that universally satisfies η, *i.e.* all the paths issued from its initial state satisfy η. The notion "more complete" is based on a completeness pre-order $\preceq$ on states of partial transition systems, that, in its turn, induces a pre-order on partial transition systems. This is closely related to the notion of bisimilarity between transition systems. The complexity of this problem is shown to be 2EXPTIME in the length of the formula and polynomial in the size of the model, by reduction to LTL synthesis. The LTL GMC problem in this version is obviously related to our EA problem, limited to the case where only state labels can be unknown. Yet, the two problems do not coincide. Our problem EA asks whether $\mathscr{M}$ *itself* can be extended (that is: its labels can), and possibly completed so as to get a bi-valued $\mathscr{M}^c$ that universally satisfy η, whereas the above described LTL GMC problem asks for the existence of *some* model $\mathscr{M}'$ satisfying η, where $\mathscr{M}$ and $\mathscr{M}'$ are related via the mentioned pre-order on structures; observe that $\mathscr{M}'$ might have less states than $\mathscr{M}$. The second version of LTL GMC studied in [13]) is based on a simpler pre-order on partial transition systems (previously suggested in [9]). It is less directly related to our work, so we will not discuss it.

Other, conceptually or technically related earlier works include:

- [18], taking an algebraic approach to the completion of partial first-order models, again representing partial information about the actual world;
- [2] (see also, [3], [10]) where a technique also called "partial model checking" was introduced for verifying concurrent systems by gradually removing concurrent components and then reducing the specification, until completely checked, in order to avoid the state explosion problem. This idea is only implicitly related to the problems discussed here.
- [15], where synthesis of reactive programs and systems under incomplete information is studied, in the case of an open system must be guaranteed to satisfy a given safety specification but can only partly read the input signals generated by its environment. An important practical case of synthesis from partial models is the problem of *controller synthesis*.

Lastly, another, more technically related to the present work, is the line of research in modal logic involving modal operators that change the models dynamically in the course of formula evaluation. It originates, *inter alia*, with van Benthem's *sabotage logics* [21], [8] and Gabbay's *reactive Kripke semantics* [12]. These were followed by various further proposals for *model update logics*, including enrichments of modal logics with *global and local graph modifiers* [7], and *relation-changing modal operators and logics* [4]. Typically, these approaches involve operations on models that not only extend, but also shrink or modify them in various ways and, moreover, add syntactic instructions for such operations in the language. These features can often lead to undecidability of both model checking and satisfiability of such logics, see [5], [6].

7.3 Future Work

We intend to organise implementation of our algorithms and to experimentally test their practical feasibility on some case studies. For example, our approach might be adapted to the formal modelling of ecology systems via transition graphs, proposed in [19]. Clearly, some expert knowledge can be missing in representations based on empirical observations of the environment, and to fill such gaps might turn out useful. In that case, as in other application cases, certainly not every conceivable completion of a partial transition system might be suitable for the intended applications, because of possibly implicit assumptions on which are the "realistic possible completions". It would then be interesting to study also criteria for classifying completions, so as to direct the search in a more realistic way, according to the domain experts.

We also intend to extend the study to the branching time logics CTL and CTL*, and to a multi-agent context, where some agents and/or their actions might also be unspecified or unknown, thus possibly taking branching time and multi-agent logics, such as the alternating time temporal logic ATL [1], to express specification properties.

References

1 R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

2 Henrik Reif Andersen. Partial model checking (extended abstract). In *Proc. of LICS 1995*, pages 398–407. IEEE Computer Society, 1995.

3 Henrik Reif Andersen and Jørn Lind-Nielsen. Partial model checking of modal equations: A survey. *Intern. J. on Software Tools for Technology Transfer*, 2(3):242–259, 1999.

4 Carlos Areces, Raul Fervari, and Guillaume Hoffmann. Relation-changing modal operators. *Logic Journal of the IGPL*, 23(4):601–627, 2015.

5 Carlos Areces, Raul Fervari, Guillaume Hoffmann, and Mauricio Martel. Undecidability of relation-changing modal logics. In *Proc. of DALI 2017*, volume 10669 of *LNCS*, pages 1–16. Springer, 2017.

6 Carlos Areces, Raul Fervari, Guillaume Hoffmann, and Mauricio Martel. Satisfiability for relation-changing logics. *J. Log. Comput.*, 28(7):1443–1470, 2018.

7 Guillaume Aucher, Philippe Balbiani, Luis Fariñas del Cerro, and Andreas Herzig. Global and local graph modifiers. *Electr. Notes Theor. Comput. Sci.*, 231:293–307, 2009.

8 Guillaume Aucher, Johan van Benthem, and Davide Grossi. Modal logics of sabotage revisited. *J. Log. Comput.*, 28(2):269–303, 2018.

9 Glenn Bruns and Patrice Godefroid. Generalized model checking: Reasoning about partial state spaces. In Catuscia Palamidessi, editor, *CONCUR 2000 - Concurrency Theory, 11th International Conference, University Park, PA, USA, August 22-25, 2000, Proceedings*, volume 1877 of *Lecture Notes in Computer Science*, pages 168–182. Springer, 2000. doi:10.1007/3-540-44618-4_14.

10 Gabriele Costa, David A. Basin, Chiara Bodei, Pierpaolo Degano, and Letterio Galletta. From natural projection to partial model checking and back. In *Proc. of TACAS 2018*, volume 10805 of *LNCS*, pages 344–361. Springer, 2018.

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

12 Dov M. Gabbay. *Reactive Kripke Semantics*. Cognitive Technologies. Springer, 2013.

13 Patrice Godefroid and Nir Piterman. LTL generalized model checking revisited. In Neil D. Jones and Markus Müller-Olm, editors, *Verification, Model Checking, and Abstract Interpretation, 10th International Conference, VMCAI 2009, Savannah, GA, USA, January 18-20, 2009. Proceedings*, volume 5403 of *Lecture Notes in Computer Science*, pages 89–104. Springer, 2009. doi:10.1007/978-3-540-93900-9_11.

14 Valentin Goranko. Model checking and model synthesis from partial models: a logic-based perspective. https://arxiv.org/abs/2012.12398, 2020.

15 Orna Kupferman and Moshe Y. Vardi. Synthesis with incomplete informatio. In Howard Barringer, Michael Fisher, Dov Gabbay, and Graham Gough, editors, *2nd Intern. Conf. on Advances in Temporal Logic*, pages 109–127, Dordrecht, 2000. Springer Netherlands.

16 David A Schmidt. Binary relations for abstraction and refinement. In *Workshop on Refinement and Abstraction, Amagasaki, Japan*. Citeseer, 1999.

17 A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, July 1985. doi:10.1145/3828.3837.

18 Bozena Staruch and Bogdan Staruch. First order theories for partial models. *Studia Logica*, 80(1):105–120, 2005.

19 Colin Thomas, Maximilien Cosme, Cédric Gaucherel, and Franck Pommereau. Model-checking ecological state-transition graphs. *PLoS Comput. Biol.*, 18(6), 2022. doi:10.1371/journal.pcbi.1009657.

20 Sebastian Uchitel, Jeff Kramer, and Jeff Magee. Behaviour model elaboration using partial labelled transition systems. *ACM SIGSOFT Software Engineering Notes*, 28(5):19–27, 2003.

21 Johan van Benthem. An essay on sabotage and obstruction. In *Mechanizing Mathematical Reasoning, Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*, volume 2605 of *LNCS*, pages 268–276. Springer, 2005.

22 Pierre Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, 28:119–136, 1985.

Combinatory Logic and Lambda Calculus Are Equal, Algebraically

Thorsten Altenkirch
School of Computer Science, University of Nottingham, UK

Ambrus Kaposi
Eötvös Loránd University, Budapest, Hungary

Artjoms Šinkarovs
Heriot-Watt University, Edinburgh, UK

Tamás Végh
Eötvös Loránd University, Budapest, Hungary

Abstract

It is well-known that extensional lambda calculus is equivalent to extensional combinatory logic. In this paper we describe a formalisation of this fact in Cubical Agda. The distinguishing features of our formalisation are the following: (i) Both languages are defined as generalised algebraic theories, the syntaxes are intrinsically typed and quotiented by conversion; we never mention preterms or break the quotients in our construction. (ii) Typing is a parameter, thus the un(i)typed and simply typed variants are special cases of the same proof. (iii) We define syntaxes as quotient inductive-inductive types (QIITs) in Cubical Agda; we prove the equivalence and (via univalence) the equality of these QIITs; we do not rely on any axioms, the conversion functions all compute and can be experimented with.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Combinatory logic, lambda calculus, quotient inductive types, Cubical Agda

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.24

Supplementary Material *Software (Formalisation)*: `https://bitbucket.org/akaposi/combinator`
archived at `swh:1:dir:5275bf1e26eb4aa1e2391d0ace5953c3f4898ef4`

Funding The first two authors were supported by the "Application Domain Specific Highly Reliable IT Solutions" project which has been implemented with support from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme.
Artjoms Šinkarovs: Supported by the Engineering and Physical Sciences Research Council through the grant EP/N028201/1.

1 Introduction

Proofs of the equivalence of extensional lambda calculus and extensional combinatory logic (e.g. [10, 5, 11, 6]) generally use the traditional untyped definition of presyntax. In particular, abstraction for combinators (the lambda operator, also called the bracket abstraction algorithm) is defined on combinator preterms. There are hints in the literature [21, 12] that the correspondence can be proven in an algebraic setting. This would be contrasted with the textbook proofs which work on particular representations of the syntax.

It is clear what combinatory logic is as an algebraic theory: there is a single sort of terms, two nullary operations S and K, one binary operation $- \cdot -$ and two equations $\mathsf{S} \cdot t \cdot u \cdot v = t \cdot v \cdot (u \cdot v)$ and $\mathsf{K} \cdot u \cdot v = u$. Models of this theory are called combinatory

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 24; pp. 24:1–24:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

algebras. What about the lambda calculus? Castellan, Clairambault and Dybjer [9] suggest that the syntax of lambda calculus should be defined as the initial category with families (CwF) with extra structure. This representation is generalised algebraic [8], and by indexing terms by their typing contexts, it avoids the problems related to the ξ rule [21]. In short, untyped lambda calculus is a uni-typed CwF with an isomorphism

$$\mathsf{lam} : \mathsf{Tm}\,(m+1) \cong \mathsf{Tm}\,m$$

natural in m, where Tm is the sort of terms which is indexed by the possible number of free variables. The left to right direction is abstraction, the right to left direction is application, the fact that the two roundtrips are identities are the β and η laws. In simply typed CwFs, terms are also indexed by types, the simply typed lambda calculus has an arrow type former $- \Rightarrow - : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty}$ and the above isomorphism becomes

$$\mathsf{lam} : \mathsf{Tm}\,(\Gamma \triangleright A)\,B \cong \mathsf{Tm}\,\Gamma\,(A \Rightarrow B).$$

Here terms are indexed both by contexts and types, and $\Gamma \triangleright A$ is the context Γ extended with one variable of type A. In fact, the untyped case is a special case of the typed one where we assume all elements of Ty to be equal.

Quotient inductive-inductive types (QIITs) [14, 15] are inductive types where later sorts can be indexed over previous ones, and where equality constructors are also allowed: a QIIT is freely generated by its constructors and is quotiented at the same time by its equality constructors. The sorts, constructors and equality constructors of a QIIT can be also seen as the sorts, operations and equations of a generalised algebraic theory. Given a generalised algebraic theory, the corresponding QIIT is its initial algebra. The elimination principle of the QIIT correponds exactly to the universal property (initiality) of the initial algebra. When a language is defined as an algebraic theory, the corresponding QIIT is its intrinsic (well-formed, well-typed) syntax quotiented by conversion. That is, convertible terms are equal in such a syntax. This approach to the syntax is very natural for dependently typed languages [2] where conversion cannot be defined separately from typing, but in this paper we apply it in a simply typed setting. Cubical Agda [24] is currently the only implementation of type theory with native support for QIITs. It features the more general higher inductive-inductive types (HIITs [15]). A QIIT is a HIIT with constructors truncating each sort to be a set, in the sense of homotopy type theory [18]. This means that any two equalities between equalities of elements of a QIIT are equal. Cubical Agda also features the univalence axiom which turns an isomorphism (bijection) into an equality.

In this paper we prove that combinatory terms of a given type are isomorphic to lambda terms of the same type, thus by univalence we obtain an equality $\mathsf{Tm}_{\mathsf{C}}\,A = \mathsf{Tm}_{\mathsf{L}}\,\diamond\,A$. The subscripts denote combinatory and lambda terms, respectively, and $\diamond$ is the empty context. Here Tm_{C} also features four equations expressing extensionality in addition to the computation rules of S and K mentioned above. In the proof, we make use of an auxiliary theory Cwk which is a variant of C featuring contexts, an operation q (the last variable in a context, a.k.a. the zero De Bruijn index) and a weakening operation (which is also the successor operation for De Bruijn indices). We define lambda abstraction by induction on terms in Cwk. An illustration of Cwk is that extensionality can be expressed by the implication $\mathsf{wk}\,t \cdot \mathsf{q} = \mathsf{wk}\,t' \cdot \mathsf{q} \to t = t'$.

Equality of combinatory and lambda terms ensures that lambda terms can be replaced by combinatory terms in any construction and vice versa. This is indeed the case in Cubical Agda as equality of the two different sets of terms is proof-relevant, and we can transport over it. Our proof is parameterised by a set of types Ty closed under arrow $- \Rightarrow -$, thus it applies to both the un(i)-typed and simply typed cases.

The main contribution of this paper is an algebraic proof of the equivalence of combinatory logic and lambda calculus which does not mention representations. Further contributions are the typing generality and the formalisation in Cubical Agda.

1.1 Structure of the paper

After summarising related work and describing our notations, in Section 2 we define the three theories C, Cwk and L. In Section 3 we prove the isomorphism $\mathsf{Tm}_{\mathsf{C}}\,A \cong \mathsf{Tm}_{\mathsf{Cwk}}\diamond A$. We define the lambda calculus operations using the syntax of Cwk in Section 4. Using these, in Section 5 we prove the isomorphism $\mathsf{Tm}_{\mathsf{Cwk}}\,\Gamma\,A \cong \mathsf{Tm}_{\mathsf{L}}\,\Gamma\ A$. To represent open lambda terms as combinator terms, we introduce an arrow type with a context domain. $\Gamma \Rightarrow^* A$ is defined as $\diamond \Rightarrow^* A :\equiv A$ and $(\Gamma \triangleright B) \Rightarrow^* A :\equiv \Gamma \Rightarrow^* B \Rightarrow A$. In Section 6 we prove $\mathsf{Tm}_{\mathsf{L}}\diamond(\Gamma \Rightarrow^* A) \cong \mathsf{Tm}_{\mathsf{L}}\,\Gamma\,A$. Putting together everything we obtain our main theorem $\mathsf{Tm}_{\mathsf{C}}\,(\Gamma \Rightarrow^* A) \cong \mathsf{Tm}_{\mathsf{L}}\,\Gamma\,A$. We conclude in Section 7.

1.2 Related work

It is usual to describe the semantics of languages as (generalised or essentially) algebraic theories, see e.g. [16]. The syntax is however usually given by abstract syntax trees. There are few textbooks which use well-typed unquotiented syntax trees, e.g. [25]. Several important constructions on the syntax of typed lambda calculi can be performed on intrinsic quotiented terms, e.g. normalisation [1], parametricity [2] and typechecking [13]. In this paper we show that the bracket abstraction algorithm can be also defined in the typed and quotiented setting.

Selinger [21] remarks that extensional models of lambda calculus do not form an algebraic variety because the subalgebra of closed terms is not extensional. This does not apply in our setting using CwFs because closed terms do not form a subalgebra. We use the algebraic description of lambda calculus by Castellan, Clairambault and Dybjer (uni-typed CwF) [9]. Hyland [12] describes lambda calculus in a way equivalent to ours using notions from categorical universal algebra, but omits the connection to combinatory logic for reasons of space.

Swierstra [22] defines a correct-by-construction conversion of combinators into lambda terms. He uses intrinsically typed unquotiented terms indexed by their semantics using a trick by McBride [17].

The relationship of combinatory logic and lambda calculus is still an active research area, for example a rewriting relation for combinator terms equivalent to β reduction was investigated in [19, 20], using preterms and typing relations. Combinators are used in realisability semantics in the form of partial combinatory algebras, for example in [4].

1.3 Metatheory and formalisation

We work with notations close to Agda's. The universe of types is written Set, we don't write universe indices, however we work in a predicative setting. Dependent functions are written $(x : A) \to B$ where B can use x, application is juxtaposition. We write implicit arguments as $\{x : A\} \to B$ or we simply omit them and just write B. When f is an implicit function, we can supply arguments in curly brackets as $f\,\{a\}$. Implicit arguments are used in many places for readability, but this is just a concise notation, formally all arguments are always specified. Σ types are written using infix $\times$, the unit type $\top$ has one definitionally unique element tt. We write definitional equality as $\equiv$, definitions using $:\equiv$, propositional equality as $=$. We assume definitional function extensionality, that is, a propositional equality of two functions is proven as a pointwise equality. We use equational reasoning notation when proving equalities. We define isomorphism as the following iterated Σ type (named record type). We overload the name of the isomorphism and the function in the forward direction.

$$(\mathsf{f} : A \cong B) :\equiv (\mathsf{f} : A \to B) \times (\mathsf{f}^{-1} : B \to A) \times (f^{\beta} : \mathsf{f}^{-1}\,(\mathsf{f}\,a) = a) \times (\mathsf{f}^{\eta} : \mathsf{f}\,(\mathsf{f}^{-1}\,b) = b)$$

We use QIITs (set-truncated HIITs) and eliminate from them by pattern matching. All the pattern matching definitions can be defined using the elimination principles of the QIITs. Because most of the equalities that we prove or use are propositions, we do not prove equalities of equalities in the paper. This is the main difference between the paper and the Cubical Agda formalisation. In the formalisation most of the line count comes from boilerplate proofs proving equalities of equalites using the isSet constructors of QIITs. We define special cases of the elimination principles for some of the QIITs and use them to reduce this boilerplate. For example, when proving a proposition by induction on a QIIT, we do not need to provide methods for the equality constructors. The only non-propositional equalities we prove are coming from isomorphisms via univalence. The formalisation is available as supplementary material.

This paper can be understood without knowing Cubical Agda or even homotopy type theory.

2 Three theories

We parameterise Sections 2–6 by a $\mathsf{Ty} : \mathsf{Set}$ and $- \Rightarrow - : \mathsf{Ty} \to \mathsf{Ty} \to \mathsf{Ty}$. We define contexts inductively by the following constructors.

$$\begin{aligned} &\mathsf{Con} &&: \mathsf{Set} \\ &\diamond &&: \mathsf{Con} \\ &- \triangleright - &&: \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Con} \end{aligned}$$

$\diamond$ denotes the empty context, $\Gamma \triangleright A$ is the context Γ extended by the type A. A context of length three containing types A, B, C is written $\diamond \triangleright A \triangleright B \triangleright C$.

In Figures 1, 2, 3 we define the theories C, Cwk and L, respectively. The syntaxes (initial models/algebras) for these theories are implemented in Cubical Agda using inductive types with equality constructors. The syntaxes of C and Cwk are indexed quotient inductive types (QIT), the syntax of L is given by two mutually defined indexed QITs. The operators become constructors, the equations become equality constructors, and each type has an extra isSet constructor ensuring that the higher equality structure is trivial.

In the rest of this section we explain the operations and equations of the three theories, and define some derivable operations and equations in each.

2.1 Combinatory logic with extensionality

Theory C is defined by the indexed sort, three operations and six equations in Figure 1. Some of the operators have implicit parameters. For example, application $- \cdot -$ takes the types A and B implicitly, its fully explicit type is $\{A : \mathsf{Ty}\}\{B : \mathsf{Ty}\} \to \mathsf{Tm}\,(A \Rightarrow B) \to \mathsf{Tm}\,A \to \mathsf{Tm}\,B$. K has two, S has three implicit type parameters. $\mathsf{K}\beta$ has two implicit type parameters and two implicit term parameters, and we understand $\mathsf{K}\beta$ with the most general implicit parameters,

$$
\begin{aligned}
&\mathsf{Tm} &&: \mathsf{Ty} \to \mathsf{Set}\\
&- \cdot - &&: \mathsf{Tm}\,(A \Rightarrow B) \to \mathsf{Tm}\,A \to \mathsf{Tm}\,B\\
&\mathsf{K} &&: \mathsf{Tm}\,(A \Rightarrow B \Rightarrow A)\\
&\mathsf{S} &&: \mathsf{Tm}\,\big((A \Rightarrow B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow A \Rightarrow C\big)\\
&\mathsf{K}\beta &&: \mathsf{K} \cdot u \cdot v = u\\
&\mathsf{S}\beta &&: \mathsf{S} \cdot t \cdot u \cdot v = t \cdot v \cdot (u \cdot v)\\
&\mathsf{lamK}\beta &&: \mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{K})\big) = \mathsf{K}\\
&\mathsf{lamS}\beta &&: \mathsf{S} \cdot \Big(\mathsf{K} \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S})\big)\Big) \cdot \Big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S})\big)\Big) =\\
& &&\quad \mathsf{S} \cdot \Big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{K}) \cdot (\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot (\mathsf{S} \cdot (\mathsf{K} \cdot (\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}))) \cdot \mathsf{S}))\big)\Big) \cdot (\mathsf{K} \cdot \mathsf{S})\\
&\mathsf{lamwk}\cdot &&: \mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{K}) = \mathsf{S} \cdot \Big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{K}) \cdot (\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \mathsf{K})\big)\Big) \cdot (\mathsf{K} \cdot \mathsf{K})\\
&\eta &&: \mathsf{S} \cdot \mathsf{K} \cdot \mathsf{K} = \mathsf{S} \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \mathsf{K}\big) \cdot \big(\mathsf{K} \cdot (\mathsf{S} \cdot \mathsf{K} \cdot \mathsf{K})\big)
\end{aligned}
$$

Figure 1 Theory C: combinatory logic with extensionality equations. Note that Ty, – ⇒ – are parameters (beginning of Section 2).

$$
\begin{aligned}
&\mathsf{Tm} &&: \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Set}\\
&- \cdot - &&: \mathsf{Tm}\,\Gamma\,(A \Rightarrow B) \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,B\\
&\mathsf{K} &&: \mathsf{Tm}\,\Gamma\,(A \Rightarrow B \Rightarrow A)\\
&\mathsf{S} &&: \mathsf{Tm}\,\Gamma\,\big((A \Rightarrow B \Rightarrow C) \Rightarrow (A \Rightarrow B) \Rightarrow A \Rightarrow C\big)\\
&\mathsf{K}\beta &&: \mathsf{K} \cdot u \cdot v = u\\
&\mathsf{S}\beta &&: \mathsf{S} \cdot t \cdot u \cdot v = t \cdot v \cdot (u \cdot v)\\
&\mathsf{q} &&: \mathsf{Tm}\,(\Gamma \triangleright A)\,A\\
&\mathsf{wk} &&: \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,(\Gamma \triangleright B)\,A\\
&\mathsf{wk}\cdot &&: \mathsf{wk}\,(t \cdot u) = \mathsf{wk}\,t \cdot \mathsf{wk}\,u\\
&\mathsf{wkK} &&: \mathsf{wk}\,\mathsf{K} = \mathsf{K}\\
&\mathsf{wkS} &&: \mathsf{wk}\,\mathsf{S} = \mathsf{S}\\
&\mathsf{lamK}\beta &&: \mathsf{S}\,\{\diamond\} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{K})\big) = \mathsf{K}\\
&\mathsf{lamS}\beta &&: \mathsf{S}\,\{\diamond\} \cdot \Big(\mathsf{K} \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S})\big)\Big) \cdot \Big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S})\big)\Big) =\\
& &&\quad \mathsf{S}\,\{\diamond\} \cdot \Big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{K}) \cdot (\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot (\mathsf{S} \cdot (\mathsf{K} \cdot (\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}))) \cdot \mathsf{S}))\big)\Big) \cdot (\mathsf{K} \cdot \mathsf{S})\\
&\mathsf{lamwk}\cdot &&: \mathsf{S}\,\{\diamond\} \cdot (\mathsf{K} \cdot \mathsf{K}) = \mathsf{S} \cdot \Big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{K}) \cdot (\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \mathsf{K})\big)\Big) \cdot (\mathsf{K} \cdot \mathsf{K})\\
&\eta &&: \mathsf{S}\,\{\diamond\} \cdot \mathsf{K} \cdot \mathsf{K} = \mathsf{S} \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \mathsf{K}\big) \cdot \big(\mathsf{K} \cdot (\mathsf{S} \cdot \mathsf{K} \cdot \mathsf{K})\big)
\end{aligned}
$$

Figure 2 Theory Cwk: combinatory logic with variables, weakenings and extensionality equations. Note that the extensionality equations only hold in the empty context. This is enforced by specifying the implicit context argument of the first Ss to be the empty context ⋄. Ty and – ⇒ – are parameters, Con is defined inductively (beginning of Section 2).

i.e. $u : \mathsf{Tm}\,A$, $v : \mathsf{Tm}\,B$ where A and B don't have to be the same. Similarly, we use the most general versions of the the other equations. The last four equations don't have terms as parameters, but do have implicit type parameters.

Using implicit parameters, we can write typed combinatory terms the same way as we would write untyped ones. For example, the identity combinator is defined as follows.

$$\begin{aligned}&\mathsf{I} : \mathsf{Tm}\,(A \Rightarrow A)\\&\mathsf{I} :\equiv \mathsf{S} \cdot \mathsf{K} \cdot \mathsf{K}\end{aligned}$$

If we write out the implicit parameters of the S and Ks (but not the $- \cdot -$ applications), this becomes $\mathsf{S}\,\{A\}\{A \Rightarrow A\}\{A\} \cdot \mathsf{K}\,\{A\}\{A \Rightarrow A\} \cdot \mathsf{K}\,\{A\}\{A\}$. If we provide Agda with the information that I will have type $\mathsf{Tm}\,(A \Rightarrow A)$, it is enough to specify the second parameter of the second K. This uniquely determines the other implicit parameters, so in Agda we write $\mathsf{I} :\equiv \mathsf{S} \cdot \mathsf{K} \cdot \mathsf{K}\,\{\}\{A\}$. We prove the β law for I using equational reasoning.

$$\mathsf{I}\beta : \mathsf{I} \cdot t \equiv \mathsf{S} \cdot \mathsf{K} \cdot \mathsf{K} \cdot t \overset{\mathsf{S}\beta}{=} \mathsf{K} \cdot t \cdot (\mathsf{K} \cdot t) \overset{\mathsf{K}\beta}{=} t$$

We say that C has extensionality because of the last four equations $\mathsf{lamK}\beta$, ..., η. We add these so as to be equivalent to Cwk which includes these equations so as to be equivalent to L. The origin of these equations will be partly revealed in Subsection 2.2 and fully revealed in Section 4.

Our equations $\mathsf{lamK}\beta$, $\mathsf{lamS}\beta$, $\mathsf{lamwk}\cdot$, η correspond to E-ax 4, E-ax 5, E-ax 1, E-ax 2 of [11, Definition 8.10], respectively. E-ax 3 is not needed as the I combinator is not part of our syntax. $\mathsf{lamS}\beta$, $\mathsf{lamwk}\cdot$, η correspond to A.5, A.3, A.6 of [5, Corollary 7.3.15], respectively. The equation A.4 is a modified version of $\mathsf{lamK}\beta$ because [5] considers the non-extensional case as well, and the translation of $\mathsf{lamK}\beta$ does not hold in lambda calculus without η, see [5, Remark before Lemma 7.3.8].

2.2 Combinatory logic with variables, weakenings and extensionality

Theory Cwk is defined in Figure 2. The sort of terms is now indexed by both contexts and types. Application, K, S, $\mathsf{K}\beta$ and $\mathsf{S}\beta$ are just like for C with the difference that all these work in an arbitrary context. We have two extra operations which correspond to the Peano constructors of De Bruijn indices: q is the zero index, wk is successor. For example De Bruijn index 2 is written $\mathsf{wk}\,(\mathsf{wk}\,\mathsf{q}) : \mathsf{Tm}\,(\Gamma \triangleright A \triangleright B \triangleright C)\,A$. Note that Γ, A and B are implicit arguments of wk. As wk can be applied to any term, we add equations expressing that it commutes with $- \cdot -$, K and S. Finally, the three equations $\mathsf{lamK}\beta$, $\mathsf{lamS}\beta$, $\mathsf{lamwk}\cdot$ are needed for defining lambda abstraction (lam) by recursion on the syntax (see Section 4). The equation η corresponds to the η rule in L. We restrict these equations to be only valid in the empty context because lam then vacously preserves them (the input of lam is in an extended context). Limiting to the empty context is not a real limitation when contexts are defined inductively. We prove that the equations hold in any context by adding wk as many times as the length of the context. Because wk commutes with K, S and $- \cdot -$ and the four equations do not contain anything else, the weakened terms will be the same as the original terms, just in a different context. Thus we obtain the general primed versions $\mathsf{lamK}\beta'$, $\mathsf{lamS}\beta'$, $\mathsf{lamwk}\cdot'$ and η'. For example, $\mathsf{lamK}\beta'$ is defined as follows. For readability, we merged some steps.

$$\begin{aligned}&\mathsf{lamK}\beta' : \{\Gamma : \mathsf{Con}\} \to \mathsf{S}\,\{\Gamma\} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{K})\big) = \mathsf{K}\\&\mathsf{lamK}\beta'\,\{\diamond\} \qquad : \mathsf{S}\,\{\diamond\} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{K})\big) \overset{\mathsf{lamK}\beta}{=} \mathsf{K}\,\{\diamond\}\\&\mathsf{lamK}\beta'\,\{\Gamma \triangleright A\} : \mathsf{S}\,\{\Gamma \triangleright A\} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{K})\big) \qquad\qquad =(\mathsf{wkS}, \mathsf{wkK}\ \text{3x each})\end{aligned}$$

$$\begin{aligned}
&\mathsf{wk}\,(\mathsf{S}\,\{\Gamma\})\cdot(\mathsf{wk}\,\mathsf{K}\cdot\mathsf{wk}\,\mathsf{S})\cdot\big(\mathsf{wk}\,\mathsf{S}\cdot(\mathsf{wk}\,\mathsf{K}\cdot\mathsf{wk}\,\mathsf{K})\big) &&=(\mathsf{wk}\cdot\text{ twice})\\
&\mathsf{wk}\,(\mathsf{S}\,\{\Gamma\})\cdot(\mathsf{wk}\,(\mathsf{K}\cdot\mathsf{S}))\cdot\big(\mathsf{wk}\,\mathsf{S}\cdot(\mathsf{wk}\,(\mathsf{K}\cdot\mathsf{K}))\big) &&=(\mathsf{wk}\cdot\text{ twice})\\
&\mathsf{wk}\,\big(\mathsf{S}\,\{\Gamma\}\cdot(\mathsf{K}\cdot\mathsf{S})\big)\cdot\mathsf{wk}\,(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{K})) &&=(\mathsf{wk}\cdot)\\
&\mathsf{wk}\,\big(\mathsf{S}\,\{\Gamma\}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{K}))\big) &&=(\mathsf{lamK}\beta'\,\{\Gamma\})\\
&\mathsf{wk}\,(\mathsf{K}\,\{\Gamma\}) &&=(\mathsf{wkK})\\
&\mathsf{K}\,\{\Gamma\triangleright A\}
\end{aligned}$$

We derive another version of each of the four equations which we call the pointful variants. These are the versions that will be actually used when defining lam.

$$\begin{aligned}
&\mathsf{lamK}\beta'' &&: \mathsf{S}\cdot\ (\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{K})\cdot u)\cdot v = u\\
&\mathsf{lamS}\beta'' &&: \mathsf{S}\cdot\big(\mathsf{S}\cdot(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot t)\cdot u\big)\cdot v = \mathsf{S}\cdot(\mathsf{S}\cdot t\cdot u)\cdot(\mathsf{S}\cdot u\cdot v)\\
&\mathsf{lamwk}\cdot'' &&: \mathsf{K}\cdot(t\cdot u) = \mathsf{S}\cdot(\mathsf{K}\cdot t)\cdot(\mathsf{K}\cdot u)\\
&\eta'' &&: t = \mathsf{S}\cdot(\mathsf{K}\cdot t)\cdot(\mathsf{S}\cdot\mathsf{K}\cdot\mathsf{K})
\end{aligned}$$

We obtain the pointful versions by applying $\mathsf{K}\beta$, $\mathsf{S}\beta$ multiple times to the pointfree version. Here is the proof for $\mathsf{lamK}\beta$, see Appendix A or the formalisation for the other equations.

$$\begin{aligned}
\mathsf{lamK}\beta'' : \ &\mathsf{S}\cdot\ (\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{K})\cdot u)\cdot v &&=(\mathsf{K}\beta)\\
&\mathsf{K}\cdot\mathsf{S}\cdot u\cdot\big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{K})\cdot u\big)\cdot v &&=(\mathsf{S}\beta)\\
&\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot\big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{K})\big)\cdot u\cdot v &&=\mathsf{lamK}\beta'\\
&\mathsf{K}\cdot u\cdot v &&=(\mathsf{K}\beta)\\
&u
\end{aligned}$$

2.3 Lambda calculus

Theory L is defined in Figure 3. Lambda calculus can be seen as a second order theory with one sort Tm indexed by Ty and an isomorphism $(\mathsf{Tm}\,A\to\mathsf{Tm}\,B)\cong\mathsf{Tm}\,(A\Rightarrow B)$. The left to right direction is the binder lam which takes a function as an input. We turn this second order theory into a first order theory using a substitution calculus with term-variables in which the second order operation lam becomes a first order operation with an input in an extended context. We describe all the operations in detail: Con, Sub form a category with terminal object $\diamond$. $\diamond$ is the empty context, $\mathsf{Sub}\,\Delta\,\Gamma$ is called a substitution from Δ to Γ. It is a list of terms, where all terms have free variables in Δ and their types are in Γ. For example, a $\mathsf{Sub}\,\Delta\,(\diamond\triangleright A\triangleright B)$ corresponds to a $\mathsf{Tm}\,\Delta\,A$ together with a $\mathsf{Tm}\,\Delta\,B$. Terms can be instantiated by substitutions: if we have a $t:\mathsf{Tm}\,\Gamma\,A$ and a substitution $\sigma:\mathsf{Sub}\,\Delta\,\Gamma$, then we obtain a term $t[\sigma]$ which we call the instantiation of t by σ. This operation replaces all free variables in t by terms in σ, which in turn have free variables declared by Δ. The instantiation operation is functorial, witnessed by $[\circ]$ and $[\mathsf{id}]$. A substitution can either be the unique empty substitution ϵ which targets the empty context $\diamond$ or a substitution built using $-,-$ which targets an extended context. The operations $-,-$, p, q and equations $\triangleright\beta_1$, $\triangleright\beta_2$, $\triangleright\eta$ can be summarised as an isomorphism $\mathsf{Sub}\,\Delta\,(\Gamma\triangleright A)\cong\mathsf{Sub}\,\Delta\,\Gamma\times\mathsf{Tm}\,\Delta\,A$. The variables are typed De Bruijn indices built by q and $-[\mathsf{p}]$, the latter takes the role of wk. We have application $-\cdot-$ and abstraction lam with the computation rule $\Rightarrow\beta$ and uniqueness rule $\Rightarrow\eta$. The rules for the arrow type are summarised by the isomorphism $\mathsf{Tm}\,(\Gamma\triangleright A)\,B\cong\mathsf{Tm}\,\Gamma\,(A\Rightarrow B)$ natural in Γ. Naturality is expressed by $\mathsf{lam}[]$ and $\cdot[]$. Again we stress that most operations in L take implicit arguments, e.g. lam takes Γ, A and B implicitly before its first and only explicit argument.

Sub	$: \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}$	$-,-$	$: \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Tm}\,\Delta\,A \to \mathsf{Sub}\,\Delta\,(\Gamma \triangleright A)$
$-\circ-$	$: \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Sub}\,\Theta\,\Delta \to \mathsf{Sub}\,\Theta\,\Gamma$	p	$: \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma$
ass	$: (\sigma \circ \rho) \circ \tau = \sigma \circ (\rho \circ \tau)$	q	$: \mathsf{Tm}\,(\Gamma \triangleright A)\,A$
id	$: \mathsf{Sub}\,\Gamma\,\Gamma$	$\triangleright\beta_1$	$: \mathsf{p} \circ (\sigma, t) = \sigma$
idl	$: \sigma \circ \mathsf{id} = \sigma$	$\triangleright\beta_2$	$: \mathsf{q}[\sigma, t] = t$
idr	$: \mathsf{id} \circ \sigma = \sigma$	$\triangleright\eta$	$: \{\sigma : \mathsf{Sub}\,\Delta\,(\Gamma \triangleright A)\} \to \sigma = (\mathsf{p} \circ \sigma, \mathsf{q}[\sigma])$
ϵ	$: \mathsf{Sub}\,\Gamma\,\diamond$	lam	$: \mathsf{Tm}\,(\Gamma \triangleright A)\,B \to \mathsf{Tm}\,\Gamma\,(A \Rightarrow B)$
$\diamond\eta$	$: \{\sigma : \mathsf{Sub}\,\Gamma\,\diamond\} \to \sigma = \epsilon$	$-\cdot-$	$: \mathsf{Tm}\,\Gamma\,(A \Rightarrow B) \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,B$
Tm	$: \mathsf{Con} \to \mathsf{Ty} \to \mathsf{Set}$	$\Rightarrow\beta$	$: \mathsf{lam}\,t \cdot u = t[\mathsf{id}, u]$
$-[-]$	$: \mathsf{Tm}\,\Gamma\,A \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Tm}\,\Delta\,A$	$\Rightarrow\eta$	$: \{t : \mathsf{Tm}\,\Gamma\,(A \Rightarrow B)\} \to t = \mathsf{lam}\,(t[\mathsf{p}] \cdot \mathsf{q})$
$[\circ]$	$: A[\sigma \circ \rho] = A[\sigma][\rho]$	$\mathsf{lam}[]$	$: (\mathsf{lam}\,t)[\sigma] = \mathsf{lam}\,(t[\sigma \circ \mathsf{p}, \mathsf{q}])$
$[\mathsf{id}]$	$: A[\mathsf{id}] = A$	$\cdot[]$	$: (t \cdot u)[\sigma] = (t[\sigma]) \cdot (u[\sigma])$

Figure 3 Theory L: lambda calculus. A concise description using categorical terminology: a category with terminal object where objects are Con and the terminal object is $\diamond$; for each $A : \mathsf{Ty}$ a locally representable presheaf $\mathsf{Tm} - A$ over the category where $- \triangleright -$ generates the new objects; an isomorphism $\mathsf{Tm}\,(\Gamma \triangleright A)\,B \cong \mathsf{Tm}\,\Gamma\,(A \Rightarrow B)$ natural in Γ. Note that Ty and $- \Rightarrow -$ are parameters, Con is defined inductively (beginning of Section 2).

Naturality of substitution extension holds in any model of L:

$$\begin{aligned}
{,\circ} : (\sigma, t) \circ \rho \quad & =\!(\triangleright\eta) \\
\big(\mathsf{p} \circ ((\sigma, t) \circ \rho), \mathsf{q}[(\sigma, t) \circ \rho]\big) \quad & =\!(\mathsf{ass}, [\circ]) \\
\big((\mathsf{p} \circ (\sigma, t)) \circ \rho, \mathsf{q}[\sigma, t][\rho]\big) \quad & =\!(\triangleright\beta_1, \triangleright\beta_2) \\
(\sigma \circ \rho, t[\rho]) \quad &
\end{aligned}$$

In every model of L, pointwise equal functions are equal, we call this property funext:

$$\mathsf{funext} : t[\mathsf{p}] \cdot \mathsf{q} = t'[\mathsf{p}] \cdot \mathsf{q} \to t = t'$$

$$\mathsf{funext}\,e : t \overset{\Rightarrow\eta}{=} \mathsf{lam}\,(t[\mathsf{p}] \cdot \mathsf{q}) \overset{e}{=} \mathsf{lam}\,(t'[\mathsf{p}] \cdot \mathsf{q}) \overset{\Rightarrow\eta}{=} t'$$

3 Combinators with and without weakenings are equal

In this section we prove the equivalence of the syntaxes of C and Cwk. We will define an isomorphism $\mathsf{f} : \mathsf{Tm}_{\mathsf{C}}\,A \cong \mathsf{Tm}_{\mathsf{Cwk}}\,\diamond\,A$, and via univalence obtain $\mathsf{Tm}_{\mathsf{C}}\,A = \mathsf{Tm}_{\mathsf{Cwk}}\,\diamond\,A$. The four extensionality equations do not play a role in this section. In fact, any number of closed equations are preserved by f, as long as the same equations hold in Cwk. For readability, we will just say C when we mean the syntax of C, and similarly for Cwk.

In Figure 4 we define the forward and backward directions of f by recursion on Tm_{C} and $\mathsf{Tm}_{\mathsf{Cwk}}$, respectively. We use pattern matching notation. As C is included in Cwk we just return the same operations. We overload the constructors of the two syntaxes, e.g. we write K both for K_{C} and $\mathsf{K}_{\mathsf{Cwk}}$, it should be clear from the context which is meant. With implicit arguments, the line for K is $\mathsf{f}\,(\mathsf{K}\,\{A\}\{B\}) :\equiv \mathsf{K}\,\{\diamond\}\{A\}\{B\}$, so we choose the output K to be in the empty context. In the other direction we know that the input is in the empty context, hence we define $\mathsf{f}^{-1}\,(\mathsf{K}\,\{\diamond\}\{A\}\{B\}) :\equiv \mathsf{K}\,\{A\}\{B\}$.

$$
\begin{array}{llll}
\mathsf{f} : \mathsf{Tm}_{\mathsf{C}}\,A \to \mathsf{Tm}_{\mathsf{Cwk}}\,\diamond\,A & & \mathsf{f}^{-1} : \mathsf{Tm}_{\mathsf{Cwk}}\,\diamond\,A \to \mathsf{Tm}_{\mathsf{C}}\,A & \\
\mathsf{f}\,(t \cdot u) & :\equiv \mathsf{f}\,t \cdot \mathsf{f}\,u & \mathsf{f}^{-1}\,(t \cdot u) & :\equiv \mathsf{f}^{-1}\,t \cdot \mathsf{f}^{-1}\,u \\
\mathsf{f}\,\mathsf{K} & :\equiv \mathsf{K} & \mathsf{f}^{-1}\,\mathsf{K} & :\equiv \mathsf{K} \\
\mathsf{f}\,\mathsf{S} & :\equiv \mathsf{S} & \mathsf{f}^{-1}\,\mathsf{S} & :\equiv \mathsf{S}
\end{array}
$$

Figure 4 The proof-relevant parts of the isomorphism $\mathsf{f} : \mathsf{Tm}_{\mathsf{C}}\,A \cong \mathsf{Tm}_{\mathsf{Cwk}}\,\diamond\,A$. In the f^{-1} direction we don't have to provide cases for constructors q and wk because they are not in the empty context: Con is defined inductively, hence we know that $\diamond \neq \Gamma \triangleright A$ for any Γ and A. We treat the cases for equality constructors in the main text.

Part of the definitions of f and f^{-1} are that they preserve the equality constructors. f maps each equality constructor of C to the corresponding equality constructor of Cwk. We spell out the implicit arguments for $\mathsf{K}\beta$: $\mathsf{f}\,(\mathsf{K}\beta\,\{A\}\{B\}\{u\}\{v\}) :\equiv \mathsf{K}\beta\,\{\diamond\}\{A\}\{B\}\{\mathsf{f}\,u\}\{\mathsf{f}\,v\}$. On the Cwk side we again use the empty context, the same types and we apply f to the term arguments. The other cases are simply $\mathsf{f}\,\mathsf{S}\beta :\equiv \mathsf{S}\beta$, $\mathsf{f}\,\mathsf{lamK}\beta :\equiv \mathsf{lamK}\beta$, $\dots$, $\mathsf{f}\,\eta :\equiv \eta$.

f^{-1} also preserves all the equations in Cwk by their corresponding equations in C. The three extra equations of $\mathsf{wk}\cdot$, wkK and wkS are preserved vacously because they are equating terms in open contexts.

When proving that $\mathsf{f} \circ \mathsf{f}^{-1} = \lambda t.t$ and vice versa, we only have to compare the results of the proof irrelevant parts shown in Figure 4 because the other components are equal by isSet. We prove f^{β} and f^{η} by trivial inductions on Tm_{C} and $\mathsf{Tm}_{\mathsf{Cwk}}$, respectively.

4 Lambda calculus operations in Cwk

In this section we derive the operations of L in the syntax of Cwk. We first define the operations in Figure 5.

Substitutions are defined by recursion on the target context. Then we define weakening of these substitutions by iterating wk for terms, again by recursion on the target context. Instantiation of Cwk terms by a substitution is defined by recursion on terms. We show that instantiation preserves the equations as follows.

$$
\begin{array}{ll}
\mathsf{K}\beta[\sigma] & : (\mathsf{K} \cdot u \cdot v)[\sigma] \equiv \mathsf{K} \cdot (u[\sigma]) \cdot (v[\sigma]) \overset{\mathsf{K}\beta}{=} u[\sigma] \\
\mathsf{S}\beta[\sigma] & : (\mathsf{S} \cdot t \cdot u \cdot v)[\sigma] \equiv \mathsf{S} \cdot (t[\sigma]) \cdot (u[\sigma]) \cdot (v[\sigma]) \overset{\mathsf{S}\beta}{=} t[\sigma] \cdot (v[\sigma]) \cdot (u[\sigma] \cdot (v[\sigma])) \\
\mathsf{wk}\cdot[\sigma] & : (\mathsf{wk}\,(t \cdot u))[\sigma, v] \equiv (t[\sigma]) \cdot (u[\sigma]) \equiv (\mathsf{wk}\,t \cdot \mathsf{wk}\,u)[\sigma, v] \\
\mathsf{wkK}[\sigma] & : (\mathsf{wk}\,\mathsf{K})[\sigma] \equiv \mathsf{K} \equiv \mathsf{K}[\sigma] \\
\mathsf{wkS}[\sigma] & : (\mathsf{wk}\,\mathsf{S})[\sigma] \equiv \mathsf{S} \equiv \mathsf{S}[\sigma] \\
\mathsf{lamK}\beta[\sigma] & :\equiv \mathsf{lamK}\beta' \\
\mathsf{lamS}\beta[\sigma] & :\equiv \mathsf{lamS}\beta' \\
\mathsf{lamwk}\cdot[\sigma] & :\equiv \mathsf{lamwk}\cdot' \\
\eta[\sigma] & :\equiv \eta'
\end{array}
$$

The last four equations use the generalised versions of the extensionality equations which work in arbitrary contexts (defined in Subsection 2.2). Composition $- \circ^{\Gamma} -$ is defined by induction on the target context and uses instantiation. We write the implicit argument Γ in superscript for readability. The identity substitution id is defined by induction on the context.

$$
\begin{aligned}
&\mathsf{Sub} : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}\\
&\mathsf{Sub}\,\Delta\,\diamond \quad :\equiv \top\\
&\mathsf{Sub}\,\Delta\,(\Gamma \triangleright A) :\equiv \mathsf{Sub}\,\Delta\,\Gamma \times \mathsf{Tm}\,\Delta\,A
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{wks} : \{\Gamma : \mathsf{Con}\} \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Sub}\,(\Delta \triangleright A)\,\Gamma\\
&\mathsf{wks}\,\{\diamond\}\,\mathsf{tt} \quad :\equiv \mathsf{tt}\\
&\mathsf{wks}\,\{\Gamma \triangleright A\}\,(\sigma, t) :\equiv (\mathsf{wks}\,\{\Gamma\}\,\sigma, \mathsf{wk}\,t)
\end{aligned}
$$

$$
\begin{aligned}
&-[-] : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Tm}\,\Delta\,A\\
&(t \cdot u)[\sigma] \quad :\equiv (t[\sigma]) \cdot (u[\sigma])\\
&\mathsf{K}[\sigma] \quad :\equiv \mathsf{K}\\
&\mathsf{S}[\sigma] \quad :\equiv \mathsf{S}\\
&\mathsf{q}[\sigma, u] \quad :\equiv u\\
&(\mathsf{wk}\,t)[\sigma, u] :\equiv t[\sigma]
\end{aligned}
$$

$$
\begin{aligned}
&- \circ - : \{\Gamma : \mathsf{Con}\} \to \mathsf{Sub}\,\Delta\,\Gamma \to\\
&\qquad \mathsf{Sub}\,\Theta\,\Delta \to \mathsf{Sub}\,\Theta\,\Gamma\\
&\mathsf{tt} \quad \circ^{\{\diamond\}} \quad \rho :\equiv \mathsf{tt}\\
&(\sigma, t) \circ^{\{\Gamma \triangleright A\}} \rho :\equiv (\sigma \circ^{\{\Gamma\}} \rho, t[\rho])
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{id} : \{\Gamma : \mathsf{Con}\} \to \mathsf{Sub}\,\Gamma\,\Gamma\\
&\mathsf{id}\,\{\diamond\} \quad :\equiv \mathsf{tt}\\
&\mathsf{id}\,\{\Gamma \triangleright A\} :\equiv (\mathsf{wks}\,(\mathsf{id}\,\{\Gamma\}), \mathsf{q})
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{p} : \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma\\
&\mathsf{p} :\equiv \mathsf{wks}\,\mathsf{id}
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{lam} : \mathsf{Tm}\,(\Gamma \triangleright A)\,B \to \mathsf{Tm}\,\Gamma\,(A \Rightarrow B)\\
&\mathsf{lam}\,(t \cdot u) :\equiv \mathsf{S} \cdot \mathsf{lam}\,t \cdot \mathsf{lam}\,u\\
&\mathsf{lam}\,\mathsf{K} \quad :\equiv \mathsf{K} \cdot \mathsf{K}\\
&\mathsf{lam}\,\mathsf{S} \quad :\equiv \mathsf{K} \cdot \mathsf{S}\\
&\mathsf{lam}\,\mathsf{q} \quad :\equiv \mathsf{S} \cdot \mathsf{K} \cdot \mathsf{K}\\
&\mathsf{lam}\,(\mathsf{wk}\,t) :\equiv \mathsf{K} \cdot t
\end{aligned}
$$

Figure 5 The definitions of L operations in the syntax of Cwk. See the text for the proof-irrelevant parts.

The first projection p is the weakening of identity. lam is also defined by recursion on Cwk terms. This is usually called the bracket abstraction algorithm. lam of application applies the S combinator to the results of the recursive calls, lam of q is the identity combinator, lam of K and S are constant K and S, respectively, while lam of a weakened term is constantly that term. The fact that lam preserves the equations is the source of the three equations lamKβ, lamSβ and lamwk·. The reason for the naming of these equations is thus revealed.

$$
\begin{aligned}
\mathsf{lam\,K}\beta \quad &: \mathsf{lam}\,(\mathsf{K} \cdot u \cdot v) && \equiv\\
&\quad \mathsf{S} \cdot \mathsf{lam}\,(\mathsf{K} \cdot u) \cdot \mathsf{lam}\,v && \equiv\\
&\quad \mathsf{S} \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{K}) \cdot \mathsf{lam}\,u\big) \cdot \mathsf{lam}\,v && =(\mathsf{lamK}\beta'')\\
&\quad \mathsf{lam}\,u\\
\mathsf{lam\,S}\beta \quad &: \mathsf{lam}\,(\mathsf{S} \cdot t \cdot u \cdot v) && \equiv\\
&\quad \mathsf{S} \cdot \mathsf{lam}\,(\mathsf{S} \cdot t \cdot u) \cdot \mathsf{lam}\,v && \equiv\\
&\quad \mathsf{S} \cdot \big(\mathsf{S} \cdot \mathsf{lam}\,(\mathsf{S} \cdot t) \cdot \mathsf{lam}\,u\big) \cdot \mathsf{lam}\,v && \equiv\\
&\quad \mathsf{S} \cdot \big(\mathsf{S} \cdot (\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \mathsf{lam}\,t) \cdot \mathsf{lam}\,u\big) \cdot \mathsf{lam}\,v && =(\mathsf{lamS}\beta'')\\
&\quad \mathsf{S} \cdot (\mathsf{S} \cdot \mathsf{lam}\,t \cdot \mathsf{lam}\,u) \cdot (\mathsf{S} \cdot \mathsf{lam}\,u \cdot \mathsf{lam}\,v) && \equiv\\
&\quad \mathsf{S} \cdot \mathsf{lam}\,(t \cdot u) \cdot \mathsf{lam}\,(u \cdot v) && \equiv\\
&\quad \mathsf{lam}\,\big(t \cdot v \cdot (u \cdot v)\big)
\end{aligned}
$$

$$\begin{aligned}
&\mathsf{lam\,wk\cdot} &&: \mathsf{lam}\,(\mathsf{wk}\,(t\cdot u)) \equiv \mathsf{K}\cdot(t\cdot u) \overset{\mathsf{lamwk\cdot}''}{=} \mathsf{S}\cdot(\mathsf{K}\cdot t)\cdot(\mathsf{K}\cdot u) \equiv \mathsf{lam}\,(\mathsf{wk}\,t\cdot\mathsf{wk}\,u)\\
&\mathsf{lam\,wkK} &&: \mathsf{lam}\,(\mathsf{wk}\,\mathsf{K}) \equiv \mathsf{K}\cdot\mathsf{K} \equiv \mathsf{lam}\,\mathsf{K}\\
&\mathsf{lam\,wkS} &&: \mathsf{lam}\,(\mathsf{wk}\,\mathsf{K}) \equiv \mathsf{K}\cdot\mathsf{S} \equiv \mathsf{lam}\,\mathsf{S}
\end{aligned}$$

We make use of the double-primed non-closed and pointful variants, but we put the closed pointfree variants in the definition of Cwk because they are vacuously preserved by lam as lam operates on terms in the nonempty context. The last of the four extensionality equations η will be used to prove the η law for the defined lam.

Now we prove all the equations of L in the following order. The proofs are straightforward, see Appendix B or the formalisation for the details.

$[\circ]$	$: \{t : \mathsf{Tm}\,\Gamma\,A\} \to t[\sigma\circ\rho] = t[\sigma][\rho]$	induction on t
ass	$: \{\Gamma : \mathsf{Con}\}\{\sigma : \mathsf{Sub}\,\Delta\,\Gamma\} \to (\sigma\circ\rho)\circ\tau = \sigma\circ(\rho\circ\tau)$	induction on Γ
$\mathsf{wks}\circ$	$: \{\Gamma : \mathsf{Con}\} \to \mathsf{wks}\,\{\Gamma\}\,\sigma\circ(\rho, u) = \sigma\circ\rho$	induction on Γ
idl	$: \{\Gamma : \mathsf{Con}\} \to \mathsf{id}\,\{\Gamma\}\circ\sigma = \sigma$	induction on Γ
$[\mathsf{wks}]$	$: \{t : \mathsf{Tm}\,\Gamma\,A\} \to t[\mathsf{wks}\,\sigma] = \mathsf{wk}\,(t[\sigma])$	induction on t
$[\mathsf{id}]$	$: \{t : \mathsf{Tm}\,\Gamma\,A\} \to t[\mathsf{id}] = t$	induction on t
idr	$: \{\Gamma : \mathsf{Con}\} \to \sigma\circ^{\Gamma}\mathsf{id} = \sigma$	induction on Γ
$\diamond\eta$	$: \{\sigma : \mathsf{Sub}\,\Gamma\,\diamond\} \to \sigma = \epsilon$	holds by definition
$\triangleright\beta_1$	$: \mathsf{p}\circ(\sigma, t) \equiv \mathsf{wks}\,\mathsf{id}\circ(\sigma, t) \overset{\mathsf{wks}\circ}{=} \mathsf{id}\circ\sigma \overset{\mathsf{idl}}{=} \sigma$	derivable
$\triangleright\beta_2$	$: \mathsf{q}[\sigma, t] \equiv t$	holds by definition
$\triangleright\eta$	$: (\sigma, t) \overset{\triangleright\beta_1}{=} (\mathsf{p}\circ(\sigma, t), t) \equiv (\mathsf{p}\circ(\sigma, t), \mathsf{q}[\sigma, t])$	derivable
$\Rightarrow\beta$	$: \{t : \mathsf{Tm}\,(\Gamma\triangleright A)\,B\} \to \mathsf{lam}\,t\cdot v = t[\mathsf{id}, v]$	induction on t
$\Rightarrow\eta$	$: t = \mathsf{lam}\,(t[\mathsf{p}]\cdot\mathsf{q})$	derivable using η''
$\mathsf{lam}[]$	$: (\mathsf{lam}\,t)[\sigma] = \mathsf{lam}\,(t[\sigma\circ\mathsf{p}, \mathsf{q}])$	induction on t
$\cdot[]$	$: (t\cdot u)[\sigma] \equiv (t[\sigma])\cdot(u[\sigma])$	holds by definition

Finally we show how instantiation interacts with the combinators K and S:

$$\begin{aligned}
&\mathsf{K}[] : \mathsf{K}[\sigma] \equiv \mathsf{K}[\mathsf{wks}\,\mathsf{id}] \overset{[\mathsf{wks}]}{=} \mathsf{wk}\,(\mathsf{K}[\mathsf{id}]) \overset{[\mathsf{id}]}{=} \mathsf{wk}\,\mathsf{K} \overset{\mathsf{wkK}}{=} \mathsf{K}\\
&\mathsf{S}[] : \mathsf{S}[\sigma] \equiv \mathsf{S}[\mathsf{wks}\,\mathsf{id}] \overset{[\mathsf{wks}]}{=} \mathsf{wk}\,(\mathsf{S}[\mathsf{id}]) \overset{[\mathsf{id}]}{=} \mathsf{wk}\,\mathsf{S} \overset{\mathsf{wkS}}{=} \mathsf{S}
\end{aligned}$$

5 Combinators with weakenings and lambda terms are equal

The proof-relevant parts of $\mathsf{g} : \mathsf{Tm}_{\mathsf{Cwk}}\,\Gamma\,A \cong \mathsf{Tm}_{\mathsf{L}}\,\Gamma\,A$ are defined in Figure 6.

Mapping Cwk terms to L terms (left hand side of Figure 6) is easy for those accustomed to the lambda calculus. The combinatory operations have straightforward implementations using lambda terms. It is similarly straightforward to show that g preserves the equations of Cwk. As an example we show all the steps in proving preservation of Kβ. This also illustrates working with the equational theory of L, or in other words working within a CwF. The preservation of the other equations is proven in an analogous way.

$$\begin{aligned}
\mathsf{g}\,\mathsf{K}\beta : \mathsf{g}\,(\mathsf{K}\cdot u\cdot v) \quad & \equiv\\
\mathsf{lam}\,\big(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}])\big)\cdot\mathsf{g}\,u\cdot\mathsf{g}\,v \quad & =(\Rightarrow\beta)\\
\mathsf{lam}\,(\mathsf{q}[\mathsf{p}])[\mathsf{id}, \mathsf{g}\,u]\cdot\mathsf{g}\,v \quad & =(\mathsf{lam}[])
\end{aligned}$$

$$\begin{aligned}
&\mathsf{g} : \mathsf{Tm}_{\mathsf{Cwk}}\,\Gamma\,A \to \mathsf{Tm}_{\mathsf{L}}\,\Gamma\,A\\
&\mathsf{g}\,(t \cdot u) :\equiv \mathsf{g}\,t \cdot \mathsf{g}\,u\\
&\mathsf{g}\,\mathsf{K} \quad\;\; :\equiv \mathsf{lam}\,\big(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}])\big)\\
&\mathsf{g}\,\mathsf{S} \quad\;\; :\equiv \mathsf{lam}\,\Big(\mathsf{lam}\,\Big(\mathsf{lam}\,\big(\mathsf{q}[\mathsf{p}\circ\mathsf{p}]\cdot\mathsf{q}\cdot(\mathsf{q}[\mathsf{p}]\cdot\mathsf{q})\big)\Big)\Big)\\
&\mathsf{g}\,\mathsf{q} \quad\;\; :\equiv \mathsf{q}\\
&\mathsf{g}\,(\mathsf{wk}\,t) :\equiv (\mathsf{g}\,t)[\mathsf{p}]\\
\\
&\mathsf{g} : \{\Gamma : \mathsf{Con}\} \to \mathsf{Sub}_{\mathsf{Cwk}}\,\Delta\,\Gamma \to \mathsf{Sub}_{\mathsf{L}}\,\Delta\,\Gamma\\
&\mathsf{g}\,\{\diamond\} \quad\;\; \mathsf{tt} \quad\; :\equiv \epsilon\\
&\mathsf{g}\,\{\Gamma \rhd A\}\,(\sigma, t) :\equiv (\mathsf{g}\,\sigma, \mathsf{g}\,t)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{g}^{-1} : \mathsf{Sub}_{\mathsf{L}}\,\Delta\,\Gamma \to \mathsf{Sub}_{\mathsf{Cwk}}\,\Delta\,\Gamma\\
&\mathsf{g}^{-1} : \mathsf{Tm}_{\mathsf{L}}\,\Gamma\,A \to \mathsf{Tm}_{\mathsf{Cwk}}\,\Gamma\,A\\
&\mathsf{g}^{-1}\,(\sigma\circ\rho) :\equiv \mathsf{g}^{-1}\,\sigma \circ \mathsf{g}^{-1}\,\rho\\
&\mathsf{g}^{-1}\,\mathsf{id} \qquad :\equiv \mathsf{id}\\
&\mathsf{g}^{-1}\,\epsilon \qquad\; :\equiv \mathsf{tt}\\
&\mathsf{g}^{-1}\,(t[\sigma]) \;\; :\equiv (\mathsf{g}^{-1}\,t)[\mathsf{g}^{-1}\,\sigma]\\
&\mathsf{g}^{-1}\,(\sigma, t) \;\;\, :\equiv (\mathsf{g}^{-1}\,\sigma, \mathsf{g}^{-1}\,t)\\
&\mathsf{g}^{-1}\,\mathsf{p} \qquad\; :\equiv \mathsf{p}\\
&\mathsf{g}^{-1}\,\mathsf{q} \qquad\; :\equiv \mathsf{q}\\
&\mathsf{g}^{-1}\,(\mathsf{lam}\,t) :\equiv \mathsf{lam}\,(\mathsf{g}^{-1}\,t)\\
&\mathsf{g}^{-1}\,(t \cdot u) \;\;\, :\equiv (\mathsf{g}^{-1}\,t)\cdot(\mathsf{g}^{-1}\,u)
\end{aligned}$$

Figure 6 The proof-relevant parts of the isomorphism $\mathsf{g} : \mathsf{Tm}_{\mathsf{Cwk}}\,\Gamma\,A \cong \mathsf{Tm}_{\mathsf{L}}\,\Gamma\,A$. We treat the cases for equality constructors in the main text.

$$\begin{aligned}
&\mathsf{lam}\,\big(\mathsf{q}[\mathsf{p}][(\mathsf{id}, \mathsf{g}\,u)\circ\mathsf{p}, \mathsf{q}]\big)\cdot\mathsf{g}\,v && =([\circ])\\
&\mathsf{lam}\,\big(\mathsf{q}[\mathsf{p}\circ((\mathsf{id}, \mathsf{g}\,u)\circ\mathsf{p}, \mathsf{q})]\big)\cdot\mathsf{g}\,v && =(\rhd\beta_1)\\
&\mathsf{lam}\,\big(\mathsf{q}[(\mathsf{id}, \mathsf{g}\,u)\circ\mathsf{p}]\big)\cdot\mathsf{g}\,v && =(,\circ)\\
&\mathsf{lam}\,\big(\mathsf{q}[\mathsf{id}\circ\mathsf{p}, \mathsf{g}\,u[\mathsf{p}]]\big)\cdot\mathsf{g}\,v && =(\rhd\beta_2)\\
&\mathsf{lam}\,\big((\mathsf{g}\,u)[\mathsf{p}]\big)\cdot\mathsf{g}\,v && =(\Rightarrow\beta)\\
&(\mathsf{g}\,u)[\mathsf{p}][\mathsf{id}, \mathsf{g}\,v] && =([\circ])\\
&(\mathsf{g}\,u)[\mathsf{p}\circ(\mathsf{id}, \mathsf{g}\,v)] && =(\rhd\beta_1)\\
&(\mathsf{g}\,u)[\mathsf{id}] && =([\mathsf{id}])\\
&\mathsf{g}\,u
\end{aligned}$$

We also define g on $\mathsf{Sub}_{\mathsf{Cwk}}$ by induction on the target context, and we prove that g preserves the lambda operations defined in Section 4 as follows, in the following order.

$$\begin{aligned}
&\mathsf{gwks} : \{\sigma : \mathsf{Sub}\,\Delta\,\Gamma\} \to \mathsf{g}\,(\mathsf{wks}\,\sigma) = \mathsf{g}\,\mathsf{wks}\circ\mathsf{p} && \text{induction on } \Gamma\\
&\mathsf{gid} \quad\; : \{\Gamma : \mathsf{Con}\} \to \mathsf{g}\,(\mathsf{id}\,\{\Gamma\}) = \mathsf{id} && \text{induction on } \Gamma\\
&\mathsf{gp} \quad\;\; : \mathsf{g}\,\mathsf{p} \equiv \mathsf{g}\,(\mathsf{wks}\,\mathsf{id}) \overset{\mathsf{gwks}}{=} \mathsf{g}\,\mathsf{id}\circ\mathsf{p} \overset{\mathsf{gid}}{=} \mathsf{id}\circ\mathsf{p} \overset{\mathsf{idl}}{=} \mathsf{p} && \text{derivable}\\
&\mathsf{g}[] \quad\;\; : \mathsf{g}\,(t[\sigma]) = (\mathsf{g}\,t)[\mathsf{g}\,\sigma] && \text{induction on } t\\
&\mathsf{g}\circ \quad\; : \{\sigma : \mathsf{Sub}\,\Delta\,\Gamma\} \to \mathsf{g}\,(\sigma\circ\rho) = \mathsf{g}\,\sigma\circ\mathsf{g}\rho && \text{induction on } \Gamma\\
&\mathsf{glam} : \mathsf{g}\,(\mathsf{lam}\,t) = \mathsf{lam}\,(\mathsf{g}\,t) && \text{induction on } t
\end{aligned}$$

In the other direction we use the lambda calculus operations defined in Cwk in Section 4. g^{-1} is defined by mutual recursion on $\mathsf{Sub}_{\mathsf{L}}$ and Tm_{L} on the right hand side of Figure 6. Preservation of the equations correspond to the counterparts of the equations in Cwk that we proved in Section 4. So $\mathsf{g}^{-1}\,\mathsf{ass}_{\mathsf{L}} :\equiv \mathsf{ass}_{\mathsf{Cwk}}, \ldots, \mathsf{g}^{-1}\,\cdot[]_{\mathsf{L}} :\equiv \cdot[]_{\mathsf{Cwk}}$.

The first roundtrip is proven by induction on $\mathsf{Tm}_{\mathsf{Cwk}}$ as follows.

$$\begin{aligned}
&\mathsf{g}^{\beta} : \{t : \mathsf{Tm}_{\mathsf{Cwk}}\,\Gamma\,A\} \to \mathsf{g}^{-1}\,(\mathsf{g}\,t) = t\\
&\mathsf{g}^{\beta}\,\{t\cdot u\} : \mathsf{g}^{-1}\,(\mathsf{g}\,(t\cdot u)) \equiv \mathsf{g}^{-1}\,(\mathsf{g}\,t)\cdot\mathsf{g}^{-1}\,(\mathsf{g}\,t) \overset{\mathsf{g}^{\beta}\,\{t\},\mathsf{g}^{\beta}\,\{t\}}{=} t\cdot u\\
&\mathsf{g}^{\beta}\,\{\mathsf{K}\} \quad\; : \mathsf{g}^{-1}\,(\mathsf{g}\,\mathsf{K}) \equiv \mathsf{g}^{-1}\,(\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}]))) \equiv \mathsf{lam}\,(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}])) \overset{\mathsf{funext}\,(\mathsf{funext}\,\mathsf{K}^{=})}{=} \mathsf{K}
\end{aligned}$$

$$
\begin{aligned}
&\mathsf{g}^{\beta}\,\{\mathsf{S}\} &&: \mathsf{g}^{-1}\,(\mathsf{g}\,\mathsf{S}) \equiv \mathsf{lam}\left(\mathsf{lam}\left(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}\circ\mathsf{p}]\cdot\mathsf{q}\cdot(\mathsf{q}[\mathsf{p}]\cdot\mathsf{q}))\right)\right) \overset{\mathsf{funext}\,(\mathsf{funext}\,(\mathsf{funext}\,\mathsf{S}^{=}))}{=} \mathsf{S}\\
&\mathsf{g}^{\beta}\,\{\mathsf{q}\} &&: \mathsf{g}^{-1}\,(\mathsf{g}\,\mathsf{q}) \equiv \mathsf{g}^{-1}\,\mathsf{q} \equiv \mathsf{q}\\
&\mathsf{g}^{\beta}\,\{\mathsf{wk}\,t\} &&: \mathsf{g}^{-1}\,(\mathsf{g}\,(\mathsf{wk}\,t)) \equiv (\mathsf{g}^{-1}\,(\mathsf{g}\,t))[\mathsf{p}] \overset{\mathsf{g}^{\beta}\,\{t\}}{=} t[\mathsf{p}] \equiv t[\mathsf{wks}\,\mathsf{id}] \overset{[\mathsf{wks}]}{=} \mathsf{wk}\,(t[\mathsf{id}]) \overset{[\mathsf{id}]}{=} \mathsf{wk}\,t
\end{aligned}
$$

The interesting cases are K and S: here we use function extensionality which holds in any model of L (thus in Cwk as shown in Section 4). To prove that the implementation of K using $\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}]))$ is equal to K we apply funext twice, and then we can use well-known reasoning with CwF combinators. The proof of $\mathsf{S}^{=}$ is analogous.

$$
\begin{aligned}
\mathsf{K}^{=} : \Big((&\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}])))[\mathsf{p}]\cdot\mathsf{q}\Big)[\mathsf{p}]\cdot\mathsf{q} &&=(\text{CwF reasoning})\\
&\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}]))\cdot\mathsf{q}[\mathsf{p}]\cdot\mathsf{q} &&=(\Rightarrow\beta \text{ twice and more CwF reasoning})\\
&\mathsf{q}[\mathsf{p}] &&=(\mathsf{K}\beta)\\
&\mathsf{K}\cdot\mathsf{q}[\mathsf{p}]\cdot\mathsf{q} &&=(\mathsf{K}[]\text{ twice})\\
&\mathsf{K}[\mathsf{p}][\mathsf{p}]\cdot\mathsf{q}[\mathsf{p}]\cdot\mathsf{q} &&=(\cdot[])\\
&(\mathsf{K}[\mathsf{p}]\cdot\mathsf{q})[\mathsf{p}]\cdot\mathsf{q}
\end{aligned}
$$

The second roundtrip is a mutual induction on $\mathsf{Sub}_{\mathsf{L}}$ and Tm_{L}. We make use of the fact that g preserves the lambda operations.

$$
\begin{aligned}
&\mathsf{g}^{\eta} : \{\sigma : \mathsf{Sub}_{\mathsf{L}}\,\Delta\,\Gamma\} \to \mathsf{g}\,(\mathsf{g}^{-1}\,\sigma) = \sigma\\
&\mathsf{g}^{\eta} : \{t : \mathsf{Tm}_{\mathsf{L}}\,\Gamma\,A\} \to \mathsf{g}\,(\mathsf{g}^{-1}\,t) = t\\
&\mathsf{g}^{\eta}\,\{\sigma\circ\rho\} : \mathsf{g}\,(\mathsf{g}^{-1}\,(\sigma\circ\rho)) \equiv \mathsf{g}\,(\mathsf{g}^{-1}\,\sigma\circ\mathsf{g}^{-1}\,\rho) \overset{\mathsf{g}\circ}{=} \mathsf{g}\,(\mathsf{g}^{-1}\,\sigma)\circ\mathsf{g}\,(\mathsf{g}^{-1}\,\rho) \overset{\mathsf{g}^{\eta}\,\{\sigma\},\mathsf{g}^{\eta}\,\{\rho\}}{=} \sigma\circ\rho\\
&\mathsf{g}^{\eta}\,\{\mathsf{id}\} \quad : \mathsf{g}\,(\mathsf{g}^{-1}\,\mathsf{id}) \equiv \mathsf{g}\,\mathsf{id} \overset{\mathsf{gid}}{=} \mathsf{id}\\
&\mathsf{g}^{\eta}\,\{\epsilon\} \quad : \mathsf{g}\,(\mathsf{g}^{-1}\,\epsilon) \equiv \mathsf{g}\,\{\diamond\}\mathsf{tt} \equiv \epsilon\\
&\mathsf{g}^{\eta}\,\{t[\sigma]\} \quad : \mathsf{g}\,(\mathsf{g}^{-1}\,(t[\sigma])) \equiv \mathsf{g}\left((\mathsf{g}^{-1}\,t)[\mathsf{g}^{-1}\,\sigma]\right) \overset{\mathsf{g}[]}{=} (\mathsf{g}\,(\mathsf{g}^{-1}\,t))[\mathsf{g}\,(\mathsf{g}^{-1}\,\sigma)] \overset{\mathsf{g}^{\eta}\,\{t\},\mathsf{g}^{\eta}\,\{\rho\}}{=} t[\sigma]\\
&\mathsf{g}^{\eta}\,\{\sigma,t\} \quad : \mathsf{g}\,(\mathsf{g}^{-1}\,(\sigma,t)) \equiv \mathsf{g}\,\{\Gamma\rhd A\}\,(\mathsf{g}^{-1}\,\sigma,\mathsf{g}^{-1}\,t) \equiv \left(\mathsf{g}\,(\mathsf{g}^{-1}\,\sigma),\mathsf{g}\,(\mathsf{g}^{-1}\,t)\right) \overset{\mathsf{g}^{\eta}}{=} (\sigma,t)\\
&\mathsf{g}^{\eta}\,\{\mathsf{p}\} \quad : \mathsf{g}\,(\mathsf{g}^{-1}\,\mathsf{p}) \equiv \mathsf{g}\,\mathsf{p} \overset{\mathsf{gp}}{=} \mathsf{p}\\
&\mathsf{g}^{\eta}\,\{\mathsf{q}\} \quad : \mathsf{g}\,(\mathsf{g}^{-1}\,\mathsf{q}) \equiv t\\
&\mathsf{g}^{\eta}\,\{\mathsf{lam}\,t\} : \mathsf{g}\,(\mathsf{g}^{-1}\,(\mathsf{lam}\,t)) \equiv \mathsf{g}\left(\mathsf{lam}\,(\mathsf{g}^{-1}\,t)\right) \overset{\mathsf{glam}}{=} \mathsf{lam}\,(\mathsf{g}\,(\mathsf{g}^{-1}\,t)) \overset{\mathsf{g}^{\eta}\,\{t\}}{=} \mathsf{lam}\,t\\
&\mathsf{g}^{\eta}\,\{t\cdot u\} \quad : \mathsf{g}\,(\mathsf{g}^{-1}\,(t\cdot u)) \equiv \mathsf{g}\,(\mathsf{g}^{-1}\,t)\cdot\mathsf{g}\,(\mathsf{g}^{-1}\,u) \overset{\mathsf{g}^{\eta}\,\{t\},\mathsf{g}^{\eta}\,\{u\}}{=} t\cdot u
\end{aligned}
$$

6 Lambda terms can be moved to the empty context

We define $\mathsf{h} : \mathsf{Tm}_{\mathsf{L}}\,\diamond\,(\Gamma\Rightarrow^{*}A) \cong \mathsf{Tm}_{\mathsf{L}}\,\Gamma\,A$. Both directions and the roundtrips are defined by induction on Γ.

$$
\begin{aligned}
&\mathsf{h} : \{\Gamma : \mathsf{Con}\} \to \mathsf{Tm}_{\mathsf{L}}\,\diamond\,(\Gamma\Rightarrow^{*}A) \to \mathsf{Tm}_{\mathsf{L}}\,\Gamma\,A\\
&\mathsf{h} \quad \{\diamond\} \qquad t :\equiv t\\
&\mathsf{h} \quad \{\Gamma\rhd B\}\,t :\equiv (\mathsf{h}\,\{\Gamma\}\,t)[\mathsf{p}]\cdot\mathsf{q}\\
&\mathsf{h}^{-1} : \{\Gamma : \mathsf{Con}\} \to \mathsf{Tm}_{\mathsf{L}}\,\Gamma\,A \to \mathsf{Tm}_{\mathsf{L}}\,\diamond\,(\Gamma\Rightarrow^{*}A)\\
&\mathsf{h}^{-1}\,\{\diamond\} \qquad t :\equiv t
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{h}^{-1}\,\{\Gamma \triangleright B\}\,t :\equiv \mathsf{h}^{-1}\,\{\Gamma\}\,(\mathsf{lam}\,t)\\
&\mathsf{h}^{\beta} : \{\Gamma : \mathsf{Con}\} \to \mathsf{h}^{-1}\,\{\Gamma\}\,(\mathsf{h}\,\{\Gamma\}\,t) = t\\
&\mathsf{h}^{\beta}\quad \{\diamond\} \qquad\quad : \mathsf{h}^{-1}\,\{\diamond\}\,(\mathsf{h}\,\{\diamond\}\,t) \equiv t\\
&\mathsf{h}^{\beta}\quad \{\Gamma \triangleright B\} : \mathsf{h}^{-1}\,\{\Gamma \triangleright B\}\,(\mathsf{h}\,\{\Gamma \triangleright B\}\,t) \equiv \mathsf{h}^{-1}\,\big(\mathsf{lam}\,((\mathsf{h}\,t)[\mathsf{p}]\cdot\mathsf{q})\big) \overset{\Rightarrow\eta}{=} \mathsf{h}^{-1}\,(\mathsf{h}\,t) \overset{\mathsf{h}^{\beta}\,\{\Gamma\}}{=} t\\
&\mathsf{h}^{\eta} : \{\Gamma : \mathsf{Con}\} \to \mathsf{h}\,\{\Gamma\}\,(\mathsf{h}^{-1}\,\{\Gamma\}\,t) = t\\
&\mathsf{h}^{\eta}\quad \{\diamond\} \qquad\quad : \mathsf{h}\,\{\diamond\}\,(\mathsf{h}^{-1}\,\{\diamond\}\,t) \equiv t\\
&\mathsf{h}^{\eta}\quad \{\Gamma \triangleright B\}\ : \mathsf{h}\,\{\Gamma \triangleright B\}\,(\mathsf{h}^{-1}\,\{\Gamma \triangleright B\}\,t) \equiv\\
&\qquad\qquad \big(\mathsf{h}\,(\mathsf{h}^{-1}\,(\mathsf{lam}\,t))\big)[\mathsf{p}]\cdot\mathsf{q} \qquad =(\mathsf{h}^{\eta}\,\{\Gamma\})\\
&\qquad\qquad (\mathsf{lam}\,t)[\mathsf{p}]\cdot\mathsf{q} \qquad\qquad\quad\ \ =(\mathsf{lam}[])\\
&\qquad\qquad \mathsf{lam}\,(t[\mathsf{p}\circ\mathsf{p},\mathsf{q}])\cdot\mathsf{q} \qquad\quad =(\Rightarrow\beta)\\
&\qquad\qquad t[\mathsf{p}\circ\mathsf{p},\mathsf{q}][\mathsf{id},\mathsf{q}] \qquad\qquad =(\text{CwF reasoning})\\
&\qquad\qquad t[\mathsf{id}] \qquad\qquad\qquad\qquad\quad\ =([\mathsf{id}])\\
&\qquad\qquad t
\end{aligned}
$$

Putting together f from Section 3, g from Section 5 and h, we obtain $\mathsf{Tm}_{\mathsf{C}}\,(\Gamma \Rightarrow^{*} A) \cong \mathsf{Tm}_{\mathsf{L}}\,\Gamma\,A$.

7 Conclusions and further work

We proved the equivalence (and thus, in a univalent setting, equality) of the syntax of combinatory logic and lambda calculus in an abstract setting. In this algebraic setting we do not refer to specific representations of the syntaxes. In particular, the bracket abstraction algorithm (defining lambda for combinators) preserves all equations. We believe that avoiding talking about representations is beneficial because the proof is more general, it applies to all representations. Thus we can extend the title of Selinger's paper [21] saying that lambda calculus is algebraic and (at least some) proofs about lambda calculus can be done algebraically. Moreover, we can run all the proofs even in this abstract setting using QIITs of Cubical Agda. For example, given a formalisation of normalisation for lambda calculus, we also obtain an algorithm for normalisation of combinator terms up to extensionality. In the future, it would be interesting to characterise normal forms of extensional combinatory logic using this technique.

There are several remaining open questions regarding the algebraic presentation of combinatory logic. For example, we do not know how to define lambda by recursion on the syntax of the following theories, even though they are all equivalent to Cwk: extensional combinatory logic with only variables; combinatory logic without the four extensionality equations but with funext; simply typed CwF with application, K, S and extensionality. In the future we would like to describe combinatory logic algebraically with ξ but without η following [7]. In the other direction, we would like to define an algebraic presentation of lambda calculus that is equivalent to combinatory logic without the extensionality equations. We only related the syntaxes of lambda calculus and combinatory logic, but more generally, we can turn any model of lambda calculus into a model of combinatory logic, while the other way we have to restrict to definable terms. It would be interesting to formalise this more general correspondence.

Another future research direction is the algebraic presentation of dependently typed combinatory logic following [23, 3].

References

1 Thorsten Altenkirch and Ambrus Kaposi. Normalisation by evaluation for dependent types. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, volume 52 of *LIPIcs*, pages 6:1–6:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.FSCD.2016.6.

2 Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 18–29. ACM, 2016. doi:10.1145/2837614.2837638.

3 Thorsten Altenkirch, Ambrus Kaposi, Artjoms Šinkarovs, and Tamás Végh. The Münchhausen method and combinatory type theory. In Delia Kesner and Pierre-Marie Pédrot, editors, *28th International Conference on Types for Proofs and Programs (TYPES 2022)*. University of Nantes, 2022. URL: https://types22.inria.fr/files/2022/06/TYPES_2022_paper_8.pdf.

4 Robert Atkey. Syntax and semantics of quantitative type theory. 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 56–65. ACM, 2018. doi:10.1145/3209108.3209189.

5 Hendrik Pieter Barendregt. *The lambda calculus - its syntax and semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1985.

6 Katalin Bimbó. *Combinatory Logic: Pure, Applied and Typed*. Taylor & Francis, 2011.

7 Martin W. Bunder, J. Roger Hindley, and Jonathan P. Seldin. On adding ξ to weak equality in combinatory logic. *J. Symb. Log.*, 54(2):590–607, 1989. doi:10.2307/2274872.

8 John Cartmell. Generalised algebraic theories and contextual categories. *Ann. Pure Appl. Log.*, 32:209–243, 1986. doi:10.1016/0168-0072(86)90053-9.

9 Simon Castellan, Pierre Clairambault, and Peter Dybjer. Categories with families: Unityped, simply typed, and dependently typed. *CoRR*, abs/1904.00827, 2019. arXiv:1904.00827.

10 Haskell B. Curry, Robert Feys, and William Craig. Combinatory logic, volume i. *Philosophical Review*, 68(4):548–550, 1959. doi:10.2307/2182503.

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

12 J. M. E. Hyland. Classical lambda calculus in modern dress. *Math. Struct. Comput. Sci.*, 27(5):762–781, 2017. doi:10.1017/S0960129515000377.

13 Ambrus Kaposi. Formalisation of type checking into algebraic syntax. https://bitbucket.org/akaposi/tt-in-tt/src/master/Typecheck.agda, 2018.

14 Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.*, 3(POPL):2:1–2:24, 2019. doi:10.1145/3290315.

15 András Kovács. *Type-Theoretic Signatures for Algebraic Theories and Inductive Types*. PhD thesis, Eötvös Loránd University, Hungary, 2022. URL: https://andraskovacs.github.io/pdfs/phdthesis_compact.pdf.

16 J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*. Cambridge University Press, USA, 1986.

17 Conor McBride. Outrageous but meaningful coincidences: dependent type-safe syntax and evaluation. In Bruno C. d. S. Oliveira and Marcin Zalewski, editors, *Proceedings of the ACM SIGPLAN Workshop on Generic Programming, WGP 2010, Baltimore, MD, USA, September 27-29, 2010*, pages 1–12. ACM, 2010. doi:10.1145/1863495.1863497.

18 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013. URL: https://homotopytypetheory.org/book/.

19 Jonathan P. Seldin. The search for a reduction in combinatory logic equivalent to $\lambda\beta$-reduction. *Theor. Comput. Sci.*, 412(37):4905–4918, 2011. doi:10.1016/j.tcs.2011.02.002.

20 Jonathan P. Seldin. The search for a reduction in combinatory logic equivalent to $\lambda\beta$-reduction, part II. *Theor. Comput. Sci.*, 663:34–58, 2017. `doi:10.1016/j.tcs.2016.12.016`.

21 Peter Selinger. The lambda calculus is algebraic. *J. Funct. Program.*, 12(6):549–566, 2002. `doi:10.1017/S0956796801004294`.

22 Wouter Swierstra. A correct-by-construction conversion to combinators, 2021. https://webspace.science.uu.nl/ swier004/publications/2021-jfp-submission-2.pdf. URL: `https://webspace.science.uu.nl/~swier004/publications/2021-jfp-submission-2.pdf`.

23 William W Tait. Variable-free formalization of the Curry-Howard Theory. In *Twenty Five Years of Constructive Type Theory*. Oxford University Press, October 1998. `doi:10.1093/oso/9780198501275.003.0016`.

24 Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical agda: A dependently typed programming language with univalence and higher inductive types. *J. Funct. Program.*, 31:e8, 2021. `doi:10.1017/S0956796821000034`.

25 Philip Wadler, Wen Kokke, and Jeremy G. Siek. *Programming Language Foundations in Agda.* August 2022. URL: `https://plfa.inf.ed.ac.uk/22.08/`.

A Derivation of the pointful versions of the extensionality equations from the pointfree variants

In this section we derive $\mathsf{lamS}\beta''$ from $\mathsf{lamS}\beta'$, $\mathsf{lamwk}{\cdot}\beta''$ from $\mathsf{lamwk}{\cdot}\beta'$, η'' from η. See Subsection 2.2 for the derivation of $\mathsf{lamK}\beta'$ from $\mathsf{lamK}\beta''$.

$$
\begin{array}{lll}
\mathsf{lamS}\beta'' & : \mathsf{S}\cdot\big(\mathsf{S}\cdot(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot t)\cdot u\big)\cdot v & =(\mathsf{K}\beta)\\
& \mathsf{K}\cdot\mathsf{S}\cdot u\cdot\Big(\mathsf{S}\cdot\big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot t\big)\cdot u\Big)\cdot v & =(\mathsf{S}\beta)\\
& \mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot\Big(\mathsf{S}\cdot\big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot t\big)\Big)\cdot u\cdot v & =(\mathsf{K}\beta)\\
& \mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot\Big(\mathsf{K}\cdot\mathsf{S}\cdot t\cdot\big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot t\big)\Big)\cdot u\cdot v & =(\mathsf{K}\beta,\mathsf{S}\beta)\\
& \mathsf{K}\cdot\big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\big)\cdot t\cdot\Big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot\big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\big)\cdot t\Big)\cdot u\cdot v & =(\mathsf{S}\beta)\\
& \mathsf{S}\cdot\Big(\mathsf{K}\cdot\big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\big)\Big)\cdot\Big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot\big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\big)\Big)\cdot t\cdot u\cdot v & =(\mathsf{lamS}\beta')\\
& \mathsf{S}\cdot\Big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot\Big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{K})\cdot\big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot(\mathsf{S}\cdot(\mathsf{K}\cdot(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})))\cdot\mathsf{S})\big)\Big)\Big)\cdot & \\
& (\mathsf{K}\cdot\mathsf{S})\cdot t\cdot u\cdot v & =(\mathsf{S}\beta)\\
& \mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot\Big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{K})\cdot\big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot(\mathsf{S}\cdot(\mathsf{K}\cdot(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})))\cdot\mathsf{S})\big)\Big)\cdot t\cdot & \\
& (\mathsf{K}\cdot\mathsf{S}\cdot t)\cdot u\cdot v & =(\mathsf{S}\beta,\mathsf{K}\beta)\\
& \mathsf{S}\cdot\Big(\mathsf{K}\cdot\mathsf{K}\cdot t\cdot\big(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot(\mathsf{S}\cdot(\mathsf{K}\cdot(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})))\cdot\mathsf{S})\cdot t\big)\Big)\cdot\mathsf{S}\cdot u\cdot v & =(\mathsf{K}\beta,\mathsf{S}\beta)\\
& \mathsf{S}\cdot\Big(\mathsf{K}\cdot\big(\mathsf{K}\cdot\mathsf{S}\cdot t\cdot(\mathsf{S}\cdot(\mathsf{K}\cdot(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})))\cdot\mathsf{S}\cdot t)\big)\Big)\cdot\mathsf{S}\cdot u\cdot v & =(\mathsf{K}\beta,\mathsf{S}\beta)\\
& \mathsf{S}\cdot\Big(\mathsf{K}\cdot\big(\mathsf{S}\cdot(\mathsf{K}\cdot(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S}))\cdot t\cdot(\mathsf{S}\cdot t))\big)\Big)\cdot\mathsf{S}\cdot u\cdot v & =(\mathsf{K}\beta)\\
& \mathsf{S}\cdot\Big(\mathsf{K}\cdot\big(\mathsf{S}\cdot(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot(\mathsf{S}\cdot t))\big)\Big)\cdot\mathsf{S}\cdot u\cdot v & =(\mathsf{S}\beta)\\
& \mathsf{K}\cdot\big(\mathsf{S}\cdot(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot(\mathsf{S}\cdot t))\big)\cdot u\cdot(\mathsf{S}\cdot u)\cdot v & =(\mathsf{K}\beta)\\
& \mathsf{S}\cdot(\mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot(\mathsf{S}\cdot t))\cdot(\mathsf{S}\cdot u)\cdot v & =(\mathsf{S}\beta)\\
& \mathsf{S}\cdot(\mathsf{K}\cdot\mathsf{S})\cdot(\mathsf{S}\cdot t)\cdot v\cdot(\mathsf{S}\cdot u\cdot v) & =(\mathsf{S}\beta)\\
& \mathsf{K}\cdot\mathsf{S}\cdot v\cdot(\mathsf{S}\cdot t\cdot v)\cdot(\mathsf{S}\cdot u\cdot v) & =(\mathsf{K}\beta)\\
& \mathsf{S}\cdot(\mathsf{S}\cdot t\cdot u)\cdot(\mathsf{S}\cdot u\cdot v) &
\end{array}
$$

$$
\begin{aligned}
\mathsf{lamwk}{\cdot}'' &: \mathsf{K} \cdot (t \cdot u) && =(\mathsf{K}\beta)\\
&\ \mathsf{K} \cdot \mathsf{K} \cdot u \cdot (t \cdot u) && =(\mathsf{S}\beta)\\
&\ \mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{K}) \cdot t \cdot u && =(\mathsf{lamwk}{\cdot}')\\
&\ \mathsf{S} \cdot \Big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{K}) \cdot (\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \mathsf{K})\big)\Big) \cdot (\mathsf{K} \cdot \mathsf{K}) \cdot t \cdot u && =(\mathsf{S}\beta)\\
&\ \mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{K}) \cdot (\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \mathsf{K})\big) \cdot t \cdot (\mathsf{K} \cdot \mathsf{K} \cdot t) \cdot u && =(\mathsf{S}\beta, \mathsf{K}\beta)\\
&\ \mathsf{K} \cdot \mathsf{S} \cdot t \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{K}) \cdot (\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \mathsf{K}) \cdot t\big) \cdot \mathsf{K} \cdot u && =(\mathsf{K}\beta, \mathsf{S}\beta)\\
&\ \mathsf{S} \cdot \big(\mathsf{K} \cdot \mathsf{K} \cdot t \cdot (\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \mathsf{K} \cdot t)\big) \cdot \mathsf{K} \cdot u && =(\mathsf{K}\beta, \mathsf{S}\beta)\\
&\ \mathsf{S} \cdot \big(\mathsf{K} \cdot (\mathsf{K} \cdot \mathsf{S} \cdot t \cdot (\mathsf{K} \cdot t))\big) \cdot \mathsf{K} \cdot u && =(\mathsf{K}\beta)\\
&\ \mathsf{S} \cdot \big(\mathsf{K} \cdot (\mathsf{S} \cdot (\mathsf{K} \cdot t))\big) \cdot \mathsf{K} \cdot u && =(\mathsf{S}\beta)\\
&\ \mathsf{K} \cdot (\mathsf{S} \cdot (\mathsf{K} \cdot t)) \cdot u \cdot (\mathsf{K} \cdot u) && =(\mathsf{K}\beta)\\
&\ \mathsf{S} \cdot (\mathsf{K} \cdot t) \cdot (\mathsf{K} \cdot u)\\
\eta'' &: t && =(\mathsf{K}\beta)\\
&\ \mathsf{K} \cdot t \cdot (\mathsf{K} \cdot t) && =(\mathsf{S}\beta)\\
&\ \mathsf{S} \cdot \mathsf{K} \cdot \mathsf{K} \cdot t && =(\eta')\\
&\ \mathsf{S} \cdot \big(\mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \mathsf{K}\big) \cdot \big(\mathsf{K} \cdot (\mathsf{S} \cdot \mathsf{K} \cdot \mathsf{K})\big) \cdot t && =(\mathsf{S}\beta)\\
&\ \mathsf{S} \cdot (\mathsf{K} \cdot \mathsf{S}) \cdot \mathsf{K} \cdot t \cdot \big(\mathsf{K} \cdot (\mathsf{S} \cdot \mathsf{K} \cdot \mathsf{K}) \cdot t\big) && =(\mathsf{S}\beta, \mathsf{K}\beta)\\
&\ \mathsf{K} \cdot \mathsf{S} \cdot t \cdot (\mathsf{K} \cdot t) \cdot (\mathsf{S} \cdot \mathsf{K} \cdot \mathsf{K}) && =(\mathsf{K}\beta)\\
&\ \mathsf{S} \cdot (\mathsf{K} \cdot t) \cdot (\mathsf{S} \cdot \mathsf{K} \cdot \mathsf{K})
\end{aligned}
$$

B Proofs of lambda calculus equations in Cwk

We prove all the equations stated in Section 4.

$$[\circ] : \{t : \mathsf{Tm}\,\Gamma\,A\} \to t[\sigma \circ \rho] = t[\sigma][\rho]$$

$$
\begin{aligned}
[\circ]\,\{t \cdot u\} &: (t \cdot u)[\sigma \circ \rho] \equiv (t[\sigma \circ \rho]) \cdot (u[\sigma \circ \rho]) \overset{[\circ]}{=} (t[\sigma][\rho]) \cdot (u[\sigma][\rho]) \equiv (t \cdot u)[\sigma][\rho]\\
[\circ]\,\{\mathsf{K}\} &: \mathsf{K}[\sigma \circ \rho] \equiv \mathsf{K} \equiv \mathsf{K}[\sigma][\rho]\\
[\circ]\,\{\mathsf{S}\} &: \mathsf{S}[\sigma \circ \rho] \equiv \mathsf{S} \equiv \mathsf{S}[\sigma][\rho]\\
[\circ]\,\{\mathsf{q}\} &: \mathsf{q}[(\sigma, u) \circ \rho] \equiv u[\rho] \equiv \mathsf{q}[\sigma, u][\rho]\\
[\circ]\,\{\mathsf{wk}\,t\} &: (\mathsf{wk}\,t)[(\sigma, u) \circ \rho] \equiv t[\sigma \circ \rho] \overset{[\circ]}{=} t[\sigma][\rho] \equiv (\mathsf{wk}\,t)[\sigma, u][\rho]
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{ass} : \{\Gamma : \mathsf{Con}\}\{\sigma : \mathsf{Sub}\,\Delta\,\Gamma\} \to (\sigma \circ \delta) \circ \tau = \sigma \circ (\delta \circ \tau)\\
&\mathsf{ass}\,\{\diamond\} \quad : (\mathsf{tt} \circ \rho) \circ \tau \equiv \rho \circ \tau \equiv \mathsf{tt} \circ (\rho \circ \tau)\\
&\mathsf{ass}\,\{\Gamma \triangleright A\} : ((\sigma, t) \circ \rho) \circ \tau && \equiv\\
&\qquad ((\sigma \circ \rho) \circ \tau, t[\rho][\tau]) && =(\mathsf{ass}\,\{\Gamma\}, [\circ])\\
&\qquad (\sigma \circ (\rho \circ \tau), t[\rho \circ \tau]) && \equiv\\
&\qquad (\sigma, t) \circ (\rho \circ \tau)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{wks}{\circ} : \{\Gamma : \mathsf{Con}\} \to \mathsf{wks}\,\{\Gamma\}\,\sigma \circ (\rho, u) = \sigma \circ \rho\\
&\mathsf{wks}{\circ}\,\{\diamond\} \quad : \mathsf{wks}\,\{\diamond\}\,\mathsf{tt} \circ (\rho, u) \equiv \mathsf{tt} \circ (\rho, u) \equiv \mathsf{tt} \equiv \mathsf{tt} \circ \rho\\
&\mathsf{wks}{\circ}\,\{\Gamma \triangleright A\} : \mathsf{wks}\,\{\Gamma \triangleright A\}\,(\sigma, t) \circ (\rho, u) && \equiv\\
&\qquad (\mathsf{wks}\,\sigma, \mathsf{wk}\,t) \circ (\rho, u) && \equiv\\
&\qquad (\mathsf{wks}\,\{\Gamma\}\,\sigma \circ (\rho, u), t[\rho]) && =(\mathsf{wks}{\circ}\,\{\Gamma\})\\
&\qquad (\sigma \circ \rho, t[\rho]) && \equiv\\
&\qquad (\sigma, t) \circ \rho
\end{aligned}
$$

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathsf{idl} : \{\Gamma : \mathsf{Con}\} \to \mathsf{id}\,\{\Gamma\} \circ \sigma = \sigma} \\
\mathsf{idl}\,\{\diamond\} & : \mathsf{id}\,\{\diamond\} \circ \mathsf{tt} \equiv \mathsf{tt} \circ \mathsf{tt} \equiv \mathsf{tt} & \\
\mathsf{idl}\,\{\Gamma \triangleright A\} & : \mathsf{id}\,\{\Gamma \triangleright A\} \circ (\sigma, t) & \equiv \\
 & \;\; (\mathsf{wks}\,\mathsf{id}, \mathsf{q}) \circ (\sigma, t) & \equiv \\
 & \;\; (\mathsf{wks}\,\mathsf{id} \circ (\sigma, t), t) & =(\mathsf{wks}\circ) \\
 & \;\; (\mathsf{id}\,\{\Gamma\} \circ \sigma, t) & =(\mathsf{idl}\,\{\Gamma\}) \\
 & \;\; (\sigma, t) &
\end{array}
$$

$$
\begin{array}{lll}
\multicolumn{3}{l}{[\mathsf{wks}] : \{t : \mathsf{Tm}\,\Gamma\,A\} \to t[\mathsf{wks}\,\sigma] = \mathsf{wk}\,(t[\sigma])} \\
[\mathsf{wks}]\,\{t \cdot u\} & : (t \cdot u)[\mathsf{wks}\,\sigma] & \equiv \\
 & \;\; (t[\mathsf{wks}\,\sigma]) \cdot (u[\mathsf{wks}\,\sigma]) & =([\mathsf{wks}]\,\{t\}, [\mathsf{wks}]\,\{u\}) \\
 & \;\; \mathsf{wk}\,(t[\sigma]) \cdot \mathsf{wk}\,(u[\sigma]) & =(\mathsf{wk}\cdot) \\
 & \;\; \mathsf{wk}\,((t[\sigma]) \cdot (u[\sigma])) & \equiv \\
 & \;\; \mathsf{wk}\,((t \cdot u)[\sigma]) & \\
[\mathsf{wks}]\,\{\mathsf{K}\} & : \mathsf{K}[\mathsf{wks}\,\sigma] \equiv \mathsf{K} \overset{\mathsf{wkK}}{=} \mathsf{wk}\,\mathsf{K} \equiv \mathsf{wk}\,(\mathsf{K}[\sigma]) & \\
[\mathsf{wks}]\,\{\mathsf{S}\} & : \mathsf{S}[\mathsf{wks}\,\sigma] \equiv \mathsf{S} \overset{\mathsf{wkS}}{=} \mathsf{wk}\,\mathsf{S} \equiv \mathsf{wk}\,(\mathsf{S}[\sigma]) & \\
[\mathsf{wks}]\,\{\mathsf{q}\} & : \mathsf{q}[\mathsf{wks}\,(\sigma, u)] \equiv \mathsf{q}[\mathsf{wks}\,\sigma, \mathsf{wk}\,u] \equiv \mathsf{wk}\,u \equiv \mathsf{wk}\,(\mathsf{q}[\sigma, u]) & \\
[\mathsf{wks}]\,\{\mathsf{wk}\,t\} & : (\mathsf{wk}\,t)[\mathsf{wks}\,(\sigma, u)] & \equiv \\
 & \;\; (\mathsf{wk}\,t)[\mathsf{wks}\,\sigma, \mathsf{wk}\,u] & \equiv \\
 & \;\; t[\mathsf{wks}\,\sigma] & =([\mathsf{wks}]\,\{t\}) \\
 & \;\; \mathsf{wk}\,(t[\sigma]) & \equiv \\
 & \;\; \mathsf{wk}\,((\mathsf{wk}\,t)[\sigma, u]) &
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{[\mathsf{id}] : \{t : \mathsf{Tm}\,\Gamma\,A\} \to t[\mathsf{id}] = t} \\
[\mathsf{id}]\,\{t \cdot u\} & : (t \cdot u)[\mathsf{id}] \equiv (t[\mathsf{id}]) \cdot (\mathsf{id}) \overset{[\mathsf{id}]\,\{t\},[\mathsf{id}]\,\{u\}}{=} t \cdot u \\
[\mathsf{id}]\,\{\mathsf{K}\} & : \mathsf{K}[\mathsf{id}] \equiv \mathsf{K} \\
[\mathsf{id}]\,\{\mathsf{S}\} & : \mathsf{S}[\mathsf{id}] \equiv \mathsf{S} \\
[\mathsf{id}]\,\{\mathsf{q}\} & : \mathsf{q}[\mathsf{id}] \equiv \mathsf{q}[\mathsf{wks}\,\mathsf{id}, \mathsf{q}] \equiv \mathsf{q} \\
[\mathsf{id}]\,\{\mathsf{wk}\,t\} & : (\mathsf{wk}\,t)[\mathsf{id}] \equiv (\mathsf{wk}\,t)[\mathsf{wks}\,\mathsf{id}, \mathsf{q}] \equiv t[\mathsf{wks}\,\mathsf{id}] \overset{[\mathsf{wks}]}{=} \mathsf{wk}\,(t[\mathsf{id}]) \overset{[\mathsf{id}]\,\{t\}}{=} \mathsf{wk}\,t
\end{array}
$$

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathsf{idr} : \{\Gamma : \mathsf{Con}\} \to \sigma \circ^{\Gamma} \mathsf{id} = \sigma} \\
\mathsf{idr}\,\{\diamond\} & : \mathsf{tt} \circ \mathsf{id} \equiv \mathsf{tt} \\
\mathsf{idr}\,\{\Gamma \triangleright A\} & : (\sigma, t) \circ^{\Gamma \triangleright A} \mathsf{id} \equiv (\sigma \circ^{\Gamma} \mathsf{id}, t[\mathsf{id}]) \overset{\mathsf{idr}\,\{\Gamma\},[\mathsf{id}]}{=} (\sigma, t)
\end{array}
$$

$$
\begin{array}{lll}
\multicolumn{3}{l}{\Rightarrow\!\beta : \{t : \mathsf{Tm}\,(\Gamma \triangleright A)\,B\} \to \mathsf{lam}\,t \cdot v = t[\mathsf{id}, v]} \\
\Rightarrow\!\beta\,\{t \cdot u\} & : \mathsf{lam}\,(t \cdot u) \cdot v & \equiv \\
 & \;\; \mathsf{S} \cdot \mathsf{lam}\,t \cdot\, \mathsf{lam}\,u \cdot v & =(\mathsf{S}\beta) \\
 & \;\; \mathsf{lam}\,t \cdot v \cdot (\mathsf{lam}\,u \cdot v) & =(\Rightarrow\!\beta\,\{t\}, \Rightarrow\!\beta\,\{u\}) \\
 & \;\; (t[\mathsf{id}, v]) \cdot (u[\mathsf{id}, v]) & \equiv \\
 & \;\; (t \cdot u)[\mathsf{id}, v] & \\
\Rightarrow\!\beta\,\{\mathsf{K}\} & : \mathsf{lam}\,\mathsf{K} \cdot v \equiv \mathsf{K} \cdot \mathsf{K} \cdot v \overset{\mathsf{K}\beta}{=} \mathsf{K} \equiv \mathsf{K}[\mathsf{id}, v] &
\end{array}
$$

$$\Rightarrow\beta\,\{\mathsf{S}\} \quad : \mathsf{lam}\,\mathsf{S}\cdot v \equiv \mathsf{K}\cdot\mathsf{S}\cdot v \overset{\mathsf{K}\beta}{=} \mathsf{S} \equiv \mathsf{S}[\mathsf{id}, v]$$

$$\Rightarrow\beta\,\{\mathsf{q}\} \quad : \mathsf{lam}\,\mathsf{q}\cdot v \equiv \mathsf{S}\cdot\mathsf{K}\cdot\mathsf{K}\cdot v \overset{\mathsf{S}\beta}{=} \mathsf{K}\cdot v\cdot(\mathsf{K}\cdot v) \overset{\mathsf{K}\beta}{=} v \equiv \mathsf{q}[\mathsf{id}, v]$$

$$\Rightarrow\beta\,\{\mathsf{wk}\,t\} \quad : \mathsf{lam}\,(\mathsf{wk}\,t)\cdot v \equiv \mathsf{K}\cdot t\cdot v \overset{\mathsf{K}\beta}{=} t \overset{[\mathsf{id}]}{=} t[\mathsf{id}] \equiv (\mathsf{wk}\,t)[\mathsf{id}, v]$$

$$\Rightarrow\eta : t = \mathsf{lam}\,(t[\mathsf{p}]\cdot\mathsf{q})$$

$$\begin{aligned}
\Rightarrow\eta \quad : {} & t && =(\eta'')\\
& \mathsf{S}\cdot(\mathsf{K}\cdot t)\cdot(\mathsf{S}\cdot\mathsf{K}\cdot\mathsf{K}) && \equiv\\
& \mathsf{S}\cdot\mathsf{lam}\,(\mathsf{wk}\,t)\cdot(\mathsf{S}\cdot\mathsf{K}\cdot\mathsf{K}) && =([\mathsf{id}])\\
& \mathsf{S}\cdot\mathsf{lam}\,(\mathsf{wk}\,(t[\mathsf{id}]))\cdot(\mathsf{S}\cdot\mathsf{K}\cdot\mathsf{K}) && =([\mathsf{wks}])\\
& \mathsf{S}\cdot\mathsf{lam}\,(t[\mathsf{wks}\,\mathsf{id}])\cdot(\mathsf{S}\cdot\mathsf{K}\cdot\mathsf{K}) && \equiv\\
& \mathsf{lam}\,(t[\mathsf{p}]\cdot\mathsf{q})
\end{aligned}$$

$$\mathsf{lam}[] : (\mathsf{lam}\,t)[\sigma] = \mathsf{lam}\,(t[\sigma\circ\mathsf{p},\mathsf{q}])$$

$$\begin{aligned}
\mathsf{lam}[]\,\{t\cdot u\} \quad : {} & (\mathsf{lam}\,(t\cdot u))[\sigma] && \equiv\\
& \mathsf{S}\cdot((\mathsf{lam}\,t)[\sigma])\cdot((\mathsf{lam}\,u)[\sigma]) && =(\mathsf{lam}[]\,\{t\},\mathsf{lam}[]\,\{u\})\\
& \mathsf{S}\cdot\mathsf{lam}\,(t[\sigma\circ\mathsf{p},\mathsf{q}])\cdot\mathsf{lam}\,(u[\sigma\circ\mathsf{p},\mathsf{q}]) && \equiv\\
& \mathsf{lam}\,((t[\sigma\circ\mathsf{p},\mathsf{q}])\cdot(u[\sigma\circ\mathsf{p},\mathsf{q}])) && \equiv\\
& \mathsf{lam}\,((t\cdot u)[\sigma\circ\mathsf{p},\mathsf{q}])
\end{aligned}$$

$$\mathsf{lam}[]\,\{\mathsf{K}\} \quad : (\mathsf{lam}\,\mathsf{K})[\sigma] \equiv (\mathsf{K}\cdot\mathsf{K})[\sigma] \equiv \mathsf{K}\cdot\mathsf{K} \equiv \mathsf{lam}\,\mathsf{K} \equiv \mathsf{lam}\,(\mathsf{K}[\sigma\circ\mathsf{p},\mathsf{q}])$$

$$\mathsf{lam}[]\,\{\mathsf{S}\} \quad : (\mathsf{lam}\,\mathsf{S})[\sigma] \equiv (\mathsf{K}\cdot\mathsf{S})[\sigma] \equiv \mathsf{K}\cdot\mathsf{S} \equiv \mathsf{lam}\,\mathsf{S} \equiv \mathsf{lam}\,(\mathsf{S}[\sigma\circ\mathsf{p},\mathsf{q}])$$

$$\mathsf{lam}[]\,\{\mathsf{q}\} \quad : (\mathsf{lam}\,\mathsf{q})[\sigma] \equiv (\mathsf{S}\cdot\mathsf{K}\cdot\mathsf{K})[\sigma] \equiv \mathsf{S}\cdot\mathsf{K}\cdot\mathsf{K} \equiv \mathsf{lam}\,\mathsf{q} \equiv \mathsf{lam}\,(\mathsf{q}[\sigma\circ\mathsf{p},\mathsf{q}])$$

$$\begin{aligned}
\mathsf{lam}[]\,\{\mathsf{wk}\,t\} \quad : {} & (\mathsf{lam}\,(\mathsf{wk}\,t))[\sigma] && \equiv\\
& (\mathsf{K}\cdot t)[\sigma] && \equiv\\
& \mathsf{K}\cdot(t[\sigma]) && \equiv\\
& \mathsf{lam}\,(\mathsf{wk}\,(t[\sigma])) && =([\mathsf{id}])\\
& \mathsf{lam}\,(\mathsf{wk}\,(t[\sigma][\mathsf{id}])) && =([\mathsf{wks}])\\
& \mathsf{lam}\,(t[\sigma][\mathsf{wks}\,\mathsf{id}]) && =([\circ])\\
& \mathsf{lam}\,(t[\sigma\circ\mathsf{wks}\,\mathsf{id}]) && \equiv\\
& \mathsf{lam}\,(t[\sigma\circ\mathsf{p}]) && \equiv\\
& \mathsf{lam}\,(\mathsf{wk}\,t[\sigma\circ\mathsf{p},\mathsf{q}])
\end{aligned}$$

Quotients and Extensionality in Relational Doctrines

Francesco Dagnino
DIBRIS, University of Genova, Italy

Fabio Pasquali
DIMA, University of Genova, Italy

Abstract

Taking a quotient roughly means changing the notion of equality on a given object, set or type. In a quantitative setting, equality naturally generalises to a distance, measuring how much elements are similar instead of just stating their equivalence. Hence, quotients can be understood quantitatively as a change of distance. Quotients are crucial in many constructions both in mathematics and computer science and have been widely studied using categorical tools. Among them, Lawvere's doctrines stand out, providing a fairly simple functorial framework capable to unify many notions of quotient and related constructions. However, abstracting usual predicate logics, they cannot easily deal with quantitative settings. In this paper, we show how, combining doctrines and the calculus of relations, one can unify quantitative and usual quotients in a common picture. More in detail, we introduce relational doctrines as a functorial description of (the core of) the calculus of relations. Then, we define quotients and a universal construction adding them to any relational doctrine, generalising the quotient completion of existential elementary doctrine and also recovering many quantitative examples. This construction deals with an intensional notion of quotient and breaks extensional equality of morphisms. Then, we describe another construction forcing extensionality, showing how it abstracts several notions of separation in metric and topological structures.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Categorical semantics

Keywords and phrases Quantitative Reasoning, Calculus of Relations, Hyperdoctrines, Metric Spaces

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.25

Acknowledgements We are grateful to the anonymous referees for their many useful comments. They have been very helpful in improving the paper.

1 Introduction

Quotients are pervasive both in mathematic and computer science, as they are crucial in carrying out many fundamental arguments. Quotients have been widely studied and several constructions have been refined to allow one to work with quotients even though they are not natively available in the setting in which one is reasoning (such as within a type theory, where usually quotients are not a primitive concept). The intuition behind these constructions is that taking a quotient changes the notion of equality on an object to a given equivalence relation. Then, to work with (formal) quotients, one just endows each object (set, type, space, ...) with an (abstract) equivalence relation and forces the object to "believe" that that equivalence relation is the equality. This idea underlies the construction of setoids in type theories [7, 30], which are the common solution to work with quotients in that setting and underlies also the exact completion of a category with weak finite limits [14, 15].

Quantitative methods are increasingly used in many different domains, such as differential privacy [51, 9, 59, 8], denotational semantics [5, 21], algebraic theories [45, 46, 47, 1, 4], program/behavioural metrics [17, 19, 20, 22, 26, 57], and rewriting [27]. This is mainly due

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 25; pp. 25:1–25:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to the fact that these methods better deal with the imprecision arising when one reasons about the behaviour of complex software systems, especially when interacting with physical processes. In a quantitative setting, equivalence relations naturally generalise to distances, which measure how much two elements are similar instead of just saying whether they are equivalent or not. Hence, quotients could be seen quantitatively as a change of distance. Indeed, this operation is often used when dealing with metric structures, see for instance the construction of monads associated with quantitative equational theories [1, 45, 46].

A unified view of quotients covering both usual and quantitative settings is missing. The aim of this paper is to develop a notion of quotient, related concepts and constructions extending known results and incorporating new quantitative examples.

Many mathematical tools have been adopted to study quotients. Among them, Lawvere's doctrines [36, 37] stand out as a simple and powerful framework capable to cope with a large variety of situations (see [31, 50, 58] and references therein). Doctrines provide a functorial description of logical theories, abstracting the essential algebraic structure shared by both syntax and semantics of logics.

In particular, Maietti and Rosolini [42, 41] identified doctrines modelling the conjunctive fragment of first order logic with equality as the minimal setting where to define equivalence relations and quotients. Then they defined a universal construction, named elementary quotient completion, that freely adds quotients to such doctrines, showing that it subsumes many others, such as setoids and the exact completion.

In order to move this machinery to a quantitative setting, one may try to work with doctrines where the usual conjunction is replaced by its linear counterpart. In this way, equivalence relations becomes distances as transitivity becomes a triangular inequality. However smooth, this transition is less innocent than it appears. As shown in [18], to properly deal with a quantitative notion of equality one needs a more sophisticated structure, which however fails to capture important examples like the category of metric spaces and non-expansive maps. The main difficulty in working with Lawvere's doctrines is that doctrines, modelling usual predicate logic, take care of variables. This is problematic in a quantitative setting as the use of variables usually has an impact on the considered distances.

For these reasons, in this paper we take a different approach: we work with doctrines abstracting the *calculus of relations* [3, 49, 55] which is a variable-free alternative to first order logic. Here one takes as primitive concept (binary) relations instead of (unary) predicates, together with some basic operations, such as relational identities, composition and the converse of a relation. Even though in general it is less expressive than first order logic,[1] it is still quite expressive, for instance, one can axiomatise set theory in it [54]. Moreover, being variable-free, it scales well to quantitative settings, as witnessed by the fruitful adoption of relational techniques to develop quantitative methods [19, 26, 27].

Then, in this paper, we introduce *relational doctrines*, as a functorial description of the calculus of relations. Relying on this structure, we define a notion of quotient capable to deal with also quantitative settings. We present a universal construction to add such quotients to any relational doctrine. The construction extends the one in [42, 41] and can also capture quantitative instances such as the category of metric spaces and non-expansive maps.

Furthermore, related to quotients, we study the notion of extensional equality. Roughly, two functions or morphisms are extensionally equal if their outputs coincide on equal inputs. Even if quotients and extensionality are independent concepts, several known constructions that add quotients often force extensionality (see e.g., Bishop's sets, setoids over a type theory

[1] The calculus of relations is equivalent to first order logic with three variables [28].

or the ex/lex completion). Therefore the study of extensionality is essential to cover these well-known examples. We show that the relational quotient completion, changing the notion of equality on objects without affecting plain equality on arrows in the base category, may break this property. Thus, we define another universal construction that forces extensionality. We show also how this logical principle captures many notions of separation in metric and topological structures.

These results are developed using the language of 2-categories [33]. To this end, we organise relational doctrines in a suitable 2-category where morphisms abstract the usual notion of relation lifting [32]. Since many categorical concepts can be defined internally to any 2-category, in this way we get them for free also for relational doctrines. For instance, following [53], we can define (co)monads on relational doctrines, which nicely corresponds to (co)monadic relation liftings used to reason about (co)effectful programs [26, 19]. The universality of our constructions is then expressed in terms of (lax) 2-adjunctions [10], thus describing their action not only on relational doctrines, but on their morphisms as well.

The paper is organised as follows. In Section 2 we introduce relational doctrines with their basic properties, presenting several examples. In Section 3 we define quotients and the relational quotient completion, proving it is universal. In Section 4 we discuss extensionality, its connection with separation and the universal construction forcing it, showing also how it interacts with quotients. In Section 5 we compare our approach with two important classes of examples: ordered categories with involution [35], which are a generalisation of both allegories and cartesian bicategories, and elementary existential doctrines [42, 41]. Finally, Section 6 summarises our contributions and discusses directions for future work.

2 Relational Doctrines: Definition and First Properties

Doctrines are a simple and powerful framework introduced by Lawvere [36, 37] to study several kinds of logics using categorical tools. A *doctrine* P on $\mathcal{C}$ is a contravariant functor $P : \mathcal{C}^{\mathrm{op}} \to \mathcal{P}os$, where $\mathcal{P}os$ denotes the category of posets and monotone functions. The category $\mathcal{C}$ is named the *base* of the doctrine and, for X in $\mathcal{C}$, the poset $P(X)$ is called *fibre over* X. For $f : X \to Y$ an arrow in $\mathcal{C}$, the monotone function $P_f : P(Y) \to P(X)$ is called *reindexing along* f. Roughly, the base category collects the objects one is interested in with their transformations, a fibre $P(X)$ collects predicates over the object X ordered by logical entailment and reindexing allows to transport predicates between objects according to their transformations. An archetypal example of a doctrine is the contravariant powerset functor $\mathcal{P} : \mathcal{S}et^{\mathrm{op}} \to \mathcal{P}os$, where predicates are represented by subsets ordered by set inclusion.

Doctrines capture the essence of predicate logic. In this section, we will introduce *relational doctrines* as a functorial description of the essential structure of relational logics. To this end, since binary relations can be seen as predicates over a pair of objects, we will need to index posets over pairs of objects, that is, to consider functors $R : (\mathcal{C} \times \mathcal{C})^{\mathrm{op}} \to \mathcal{P}os$, where each fibre $R(X, Y)$ collects relations from X to Y. Here the reference example are set-theoretic relations: they can be organised into a functor $\mathsf{Rel} : (\mathcal{S}et \times \mathcal{S}et)^{\mathrm{op}} \to \mathcal{P}os$ where $\mathsf{Rel}(X, Y) = \mathcal{P}(X \times Y)$ and sending f, g to the inverse image $(f \times g)^{-1}$.

We endow these functors with a structure modelling a core fragment of the calculus of relations given by relational identities, composition and converse [3, 49, 55]. For set-theoretic relations, the identity relation on a set X is the diagonal $\mathsf{d}_X = \{\langle x, x' \rangle \in X \times X \mid x = x'\}$, the composition of $\alpha \in \mathsf{Rel}(X, Y)$ with $\beta \in \mathsf{Rel}(Y, Z)$ is the set $\alpha \,;\beta = \{\langle x, z \rangle \in X \times Z \mid \langle x, y \rangle \in \alpha, \langle y, z \rangle \in \beta \text{ for some } y \in Y\}$, and the converse of $\alpha \in \mathsf{Rel}(X, Y)$ is the set $\alpha^{\perp} = \{\langle y, x \rangle \in Y \times X \mid \langle x, y \rangle \in \alpha\}$. These operations interact with reindexing, *i.e.* inverse images, by the following inclusions: $\mathsf{d}_X \subseteq (f \times f)^{-1}(\mathsf{d}_Y)$ and $(f \times g)^{-1}(\alpha)\,;(g \times h)^{-1}(\beta) \subseteq (f \times h)^{-1}(\alpha\,;\beta)$ and also $((f \times g)^{-1}(\alpha))^{\perp} \subseteq (g \times f)^{-1}(\alpha^{\perp})$. The first two inclusions are not equalities in general: the former is an equality when f is injective, while the latter is an equality when g is surjective. These observations lead us to the following definition.

▶ **Definition 1.** *A* relational doctrine *consists of the following data:*

- *a base category $\mathcal{C}$,*
- *a functor $R : (\mathcal{C} \times \mathcal{C})^{\mathrm{op}} \to \mathcal{P}os$,*
- *an element $\mathsf{d}_X \in R(X,X)$, for every object X in $\mathcal{C}$, such that $\mathsf{d}_X \leqslant R_{f,f}(\mathsf{d}_Y)$, for every arrow $f : X \to Y$ in $\mathcal{C}$,*
- *a monotone function $-;- : R(X,Y) \times R(Y,Z) \to R(X,Z)$, for every triple of objects X, Y, Z in $\mathcal{C}$, such that $R_{f,g}(\alpha)\,; R_{g,h}(\beta) \leqslant R_{f,h}(\alpha\,;\beta)$, for all $\alpha \in R(A,B)$, $\beta \in R(B,C)$ and $f : X \to A$, $g : Y \to B$ and $h : Z \to C$ arrows in $\mathcal{C}$,*
- *a monotone function $(-)^{\perp} : R(X,Y) \to R(Y,X)$, for every pair of objects X, Y in $\mathcal{C}$, such that $(R_{f,g}(\alpha))^{\perp} \leqslant R_{g,f}(\alpha^{\perp})$, for all $\alpha \in R(A,B)$ and $f : X \to A$ and $g : Y \to B$,*

satisfying the following equations for all $\alpha \in R(X,Y)$, $\beta \in R(Y,Z)$ and $\gamma \in R(Z,W)$

$$\alpha\,;(\beta\,;\gamma) = (\alpha\,;\beta)\,;\gamma \qquad \mathsf{d}_X\,;\alpha = \alpha \qquad \alpha\,;\mathsf{d}_Y = \alpha$$
$$(\alpha\,;\beta)^{\perp} = \beta^{\perp}\,;\alpha^{\perp} \qquad \mathsf{d}_X^{\perp} = \mathsf{d}_X \qquad \alpha^{\perp\perp} = \alpha$$

The element d_X is the *identity* or *diagonal* relation on X, $\alpha\,;\beta$ is the *relational composition* of α followed by β, and $\alpha^{\perp}$ is the *converse* of the relation α. Note that all relational operations are lax natural transformations, but the operation of taking the converse, being an involution, is actually strictly natural. Also, each one of the two axioms stating that d is the neutral element of the composition, together with the other axioms, implies the other.

▶ Remark 2. The data defining a relational doctrine $R : (\mathcal{C} \times \mathcal{C})^{\mathrm{op}} \to \mathcal{P}os$ determine the following diagram in the category of doctrines and lax natural transformations, describing an *internal dagger category*:

$$\begin{tikzcd}
R^2 \arrow[r, "{\langle\langle\pi_1,\pi_3\rangle,-\,;-\rangle}"] & R \arrow[loop above, "{\langle\langle\pi_2,\pi_1\rangle,(-)^{\perp}\rangle}"] \arrow[r, bend left, "{\langle\pi_2,\zeta\rangle}"] \arrow[r, bend right, "{\langle\pi_1,\zeta\rangle}"'] & \mathbf{1}_{\mathcal{C}} \arrow[l, "{\langle\Delta,\mathsf{d}\rangle}"']
\end{tikzcd}$$

Here, $\mathbf{1}_{\mathcal{C}} : \mathcal{C}^{\mathrm{op}} \to \mathcal{P}os$ is the trivial doctrine, mapping every object of $\mathcal{C}$ to the singleton poset, ζ is the natural transformation whose components are the unique maps into the singleton poset, and R^2 is the pullback of $\langle\pi_1,\zeta\rangle$ against $\langle\pi_2,\zeta\rangle$, that is, the functor $R^2 : (\mathcal{C} \times \mathcal{C} \times \mathcal{C})^{\mathrm{op}} \to \mathcal{P}os$ defined by $R^2(X,Y,Z) = R(X,Y) \times R(Y,Z)$ and $R^2_{f,g,h} = R_{f,g} \times R_{g,h}$.

The following list of examples is meant to give a broad range of situations that can be described by relational doctrines. Order categories and existential elementary doctrines provide two large classes of examples which are intentionally omitted as, due to their relevance, they will be discussed separately in Section 5.

▶ **Example 3.**

1. Let $V = \langle |V|, \preceq, \cdot, 1\rangle$ be a commutative quantale. A V*-relation* [29] between sets X and Y is a function $\alpha : X \times Y \to |V|$, where $\alpha(x,y) \in |V|$ intuitively measures how much elements x and y are related by α. Then, we consider the functor V-$\mathsf{Rel} : (\mathcal{S}et \times \mathcal{S}et)^{\mathrm{op}} \to \mathcal{P}os$

where $V\text{-}\mathsf{Rel}(X,Y) = |V|^{X\times Y}$ is the set of V-relations from X to Y with the pointwise order, $V\text{-}\mathsf{Rel}_{f,g}$ is precomposition with $f \times g$ and The identity relation, composition and converse are defined as follows:

$$\mathsf{d}_X(x,x') = \begin{cases} 1 & x = x' \\ \bot & x \neq x' \end{cases} \qquad (\alpha\,;\beta)(x,z) = \bigvee_{y\in Y}(\alpha(x,y)\cdot\beta(y,z)) \qquad \alpha^{\perp}(y,x) = \alpha(x,y)$$

where $\alpha \in V\text{-}\mathsf{Rel}(X,Y)$ and $\beta \in V\text{-}\mathsf{Rel}(Y,Z)$. Special cases of this doctrine are $\mathsf{Rel} : (\mathcal{S}et \times \mathcal{S}et)^{\mathrm{op}} \to \mathcal{P}os$, when the quantale is $\mathbb{B} = \langle\{0,1\},\leqslant,\wedge,1\rangle$, and metric relations, when one considers the Lawvere's quantale $\mathbb{R}_{\geqslant 0} = \langle[0,\infty],\geqslant,+,0\rangle$ as in [38].

2. Let $R = \langle |R|,\preceq,+,\cdot,0,1\rangle$ be a continuous semiring [34, 48], that is, an ordered semiring where $\langle |R|,\preceq\rangle$ is a directed complete partial order (DCPO), 0 is the least element and $+$ and $\cdot$ are Scott-continuous functions. In this setting, we can compute sums of arbitrary arity. For a function $f : X \to |R|$, we can define its sum $\sum f$, also denoted by $\sum_{x\in X} f(x)$, as

$$\sum f = \bigvee_{I\in\mathcal{P}_\omega(X)} \sum_{i\in I} f(i)$$

where $\mathcal{P}_\omega(X)$ is the finite powerset of X. Consider $R\text{-}\mathsf{Mat} : (\mathcal{S}et \times \mathcal{S}et)^{\mathrm{op}} \to \mathcal{P}os$ where $R\text{-}\mathsf{Mat}(X,Y)$ is the set of functions $X \times Y \to |R|$ with the pointwise order, $\mathbb{R}\text{-}\mathsf{Mat}_{f,g}$ is precomposition with $f \times g$. Elements in $R\text{-}\mathsf{Mat}(X,Y)$ are a matrices with entries in $|R|$ and indices for rows and columns taken from X and Y. The identity relation, composition and converse are given by the Kronecker's delta (*i.e.* the identity matrix), matrix multiplication and transpose, defined as follows:

$$\mathsf{d}_X(x,x') = \begin{cases} 1 & x = x' \\ 0 & x \neq x' \end{cases} \qquad (\alpha\,;\beta)(x,z) = \sum_{y\in Y}(\alpha(x,y)\cdot\beta(y,z)) \qquad \alpha^{\perp}(y,x) = \alpha(x,y)$$

where $\alpha \in R\text{-}\mathsf{Mat}(X,Y)$ and $\beta \in R\text{-}\mathsf{Mat}(Y,Z)$. This relational doctrine generalises V-relations since any quantale is a continuous semiring (binary/arbitrary joins give addition/infinite sum). The paradigmatic example of a continuous semiring which is not a quantale is that of extended non-negative real numbers $[0,\infty]$, with the usual order, addition and multiplication. Restricting the base to finite sets all sums become finite, hence the definition works also for a plain ordered semiring.

3. Let $\mathcal{C}$ be a category with weak pullbacks. Denote by $\mathsf{Spn}^{\mathcal{C}}(X,Y)$ the poset reflection of the preorder whose objects are spans in $\mathcal{C}$ between X and Y and $X \xleftarrow{p_1} A \xrightarrow{p_2} Y \leqslant X \xleftarrow{q_1} B \xrightarrow{q_2} Y$ iff there is an arrow $f : A \to B$ such that $p_1 = q_1 \circ f$ and $p_2 = q_2 \circ f$. Given a span $\alpha = X \xleftarrow{p_1} A \xrightarrow{p_2} Y$ and arrows $f : X' \to X$ and $g : Y' \to Y$ in $\mathcal{C}$, define $\mathsf{Spn}_{f,g}(\alpha) \in \mathsf{Spn}^{\mathcal{C}}(X',Y')$ by one of the following equivalent diagrams:

$$\begin{tikzcd}
& & W \arrow[dl] \arrow[ddrr] & & \\
& W' \arrow[dl] \arrow[dr] & & wpb & \\
X' \arrow[dr, "f"'] & wpb & A \arrow[dl, "p_1"] \arrow[dr, "p_2"'] & & Y' \arrow[dl, "g"] \\
& X & & Y &
\end{tikzcd}
\qquad
\begin{tikzcd}
& & W \arrow[ddll] \arrow[dr] & & \\
& wpb & & W' \arrow[dl] \arrow[dr] & \\
X' \arrow[dr, "f"'] & & A \arrow[dl, "p_1"] \arrow[dr, "p_2"'] & wpb & Y' \arrow[dl, "g"] \\
& X & & Y &
\end{tikzcd}$$

The functor $\mathsf{Spn}^{\mathcal{C}} : (\mathcal{C} \times \mathcal{C})^{\mathrm{op}} \to \mathcal{P}os$ is a relational doctrine where, for $\alpha = X \xleftarrow{p_1} A \xrightarrow{p_2} Y$ and $\beta = X \xleftarrow{q_1} B \xrightarrow{q_2} Y$ it is

$$\mathsf{d}_X = \begin{tikzcd} & X \arrow[dl, "\mathsf{id}_X"'] \arrow[dr, "\mathsf{id}_X"] & \\ X & & X \end{tikzcd} \qquad \alpha\,;\beta = \begin{tikzcd} & & W \arrow[dl] \arrow[dr] & & \\ & A \arrow[dl, "p_1"] \arrow[dr, "p_2"'] & wpb & B \arrow[dl, "q_1"] \arrow[dr, "q_2"'] & \\ X & & Y & & Z \end{tikzcd} \qquad \alpha^{\perp} = \begin{tikzcd} & A \arrow[dl, "p_2"'] \arrow[dr, "p_1"] & \\ Y & & X \end{tikzcd}$$

One can do a similar construction for jointly monic spans, provided that the category $\mathcal{C}$ has strong pullbacks and a proper factorisation system. In particular, the relational doctrine of jointly monic spans over $\mathcal{S}et$ is the relational doctrine Rel of set-based relations already mentioned in Item 1.

4. Let $\mathcal{V}ec$ the category of vector spaces over real numbers and linear maps. Write $|X|$ for the underlying set of the vector space X. The functor $\mathsf{Vec} : (\mathcal{V}ec \times \mathcal{V}ec)^{\mathrm{op}} \to \mathcal{P}os$ sends X, Y to the suborder of $\mathbb{R}_{\geqslant 0}$-$\mathsf{Rel}(|X|, |Y|)$ on those α that are subadditive functions, *i.e.* $\alpha(\mathbf{x}+\mathbf{x}', \mathbf{y}+\mathbf{y}') \geqslant \alpha(\mathbf{x},\mathbf{y}) + \alpha(\mathbf{x}',\mathbf{y}')$ and homogeneous, $\alpha(a\mathbf{x}, a\mathbf{y}) = |a|\alpha(\mathbf{x},\mathbf{y})$. The functor Vec is a relational doctrine where

$$\mathsf{d}_X(\mathbf{x},\mathbf{x}') = \begin{cases} 0 & \mathbf{x} = \mathbf{x}' \\ \infty & \mathbf{x} \neq \mathbf{x}' \end{cases} \quad (\alpha\,;\beta)(\mathbf{x},\mathbf{z}) = \inf_{\mathbf{y}\in|Y|}(\alpha(\mathbf{x},\mathbf{y}) + \beta(\mathbf{y},\mathbf{z})) \quad \alpha^{\perp}(\mathbf{y},\mathbf{x}) = \alpha(\mathbf{x},\mathbf{y})$$

5. Let $R : (\mathcal{C} \times \mathcal{C})^{\mathrm{op}} \to \mathcal{P}os$ be a relational doctrine and $F : \mathcal{D} \to \mathcal{C}$ a functor. The change-of-base of R along F is the relational doctrine $F^{\star}R : (\mathcal{D} \times \mathcal{D})^{\mathrm{op}} \to \mathcal{P}os$ obtained precomposing R with $(F \times F)^{\mathrm{op}}$. The change of base allows to use relations of R to reason about the category $\mathcal{D}$. For example the forgetful functor $U : \mathcal{C} \to \mathcal{S}et$ of a concrete category $\mathcal{C}$ allows the use of set-theoretic relations to reason about $\mathcal{C}$, considering the doctrine $U^{\star}\mathsf{Rel}$ which maps a pair of objects X, Y in $\mathcal{C}$ to $\mathcal{P}(UX \times UY)$.

Let R be a relational doctrine on $\mathcal{C}$ and $\alpha \in R(X,Y)$ a relation, α is *functional* if $\alpha^{\perp}\,;\alpha \leqslant \mathsf{d}_Y$, *total* if $\mathsf{d}_X \leqslant \alpha\,;\alpha^{\perp}$, *injective* if $\alpha\,;\alpha^{\perp} \leqslant \mathsf{d}_X$, and *surjective* if $\mathsf{d}_Y \leqslant \alpha^{\perp}\,;\alpha$. The next proposition shows that functional and total relations are discretely ordered.

▶ **Proposition 4.** *For functional and total relations $\alpha, \beta \in R(X,Y)$ if $\alpha \leqslant \beta$, then $\alpha = \beta$.*

Every arrow $f : X \to Y$ defines a relation $\Gamma_f = R_{f,\mathsf{id}_Y}(\mathsf{d}_Y) \in R(X,Y)$, called the *graph* of f whose converse is given by $\Gamma_f^{\perp} = R_{f,\mathsf{id}_Y}(\mathsf{d}_Y)^{\perp} = R_{\mathsf{id}_Y,f}(\mathsf{d}_Y^{\perp}) = R_{\mathsf{id}_Y,f}(\mathsf{d}_Y)$.

▶ **Proposition 5.** *Let $f : X \to Y$ an arrow in $\mathcal{C}$. Then, Γ_f is functional and total.*

Relational composition allows us to express reindexing in relational terms and to show it has left adjoints, as proved below. Recall that in $\mathcal{P}os$ a left adjoint of a monotone function $g : K \to H$ is a monotone function $f : H \to K$ such that for every x in K and y in H, both $y \leqslant gf(y)$ and $fg(x) \leqslant x$ hold, or, equivalently, $y \leqslant g(x)$ if and only if $f(y) \leqslant x$.

▶ **Proposition 6.** *For $f : A \to X$ and $g : B \to Y$ in $\mathcal{C}$ the reindexing $R_{f,g} : R(X,Y) \to R(A,B)$ has a left adjoint $\exists^R_{f,g} : R(A,B) \to R(X,Y)$ and for $\alpha \in R(X,Y)$ and $\beta \in R(A,B)$*

$$R_{f,g}(\alpha) = \Gamma_f\,;\alpha\,;\Gamma_g^{\perp} \qquad \exists^R_{f,g}(\beta) = \Gamma_f^{\perp}\,;\beta\,;\Gamma_g$$

We conclude the section describing the 2-category $\mathbf{RD}$ of relational doctrines. Objects are relational doctrines, while a 1-arrow $F : R \to S$, where $R : (\mathcal{C} \times \mathcal{C})^{\mathrm{op}} \to \mathcal{P}os$ and $S : (\mathcal{D} \times \mathcal{D})^{\mathrm{op}} \to \mathcal{P}os$, is a pair $\langle \widehat{F}, \overline{F} \rangle$ consisting of a functor $\widehat{F} : \mathcal{C} \to \mathcal{D}$ and a natural transformation $\overline{F} : R \overset{\cdot}{\to} S \circ (\widehat{F} \times \widehat{F})^{\mathrm{op}}$, laxly preserving relational identities, composition and converse, that is, satisfying $\mathsf{d}_{\widehat{F}X} \leqslant \overline{F}_{X,X}(\mathsf{d}_X)$ and $\overline{F}_{X,Y}(\alpha)\,;\overline{F}_{Y,Z}(\beta) \leqslant \overline{F}_{X,Z}(\alpha\,;\beta)$ and $(\overline{F}_{X,Y}(\alpha))^{\perp} \leqslant \overline{F}_{Y,X}(\alpha^{\perp})$, for $\alpha \in R(X,Y)$ and $\beta \in R(Y,Z)$. A 2-arrow $\theta : F \Rightarrow G$ is a natural transformation $\theta : \widehat{F} \overset{\cdot}{\to} \widehat{G}$ such that $\overline{F}_{X,Y} \leqslant S_{\theta_X,\theta_Y} \circ \overline{G}_{X,Y}$, for all objects X, Y in the base of R. By Propositions 5 and 6 the condition of a 2-arrow $\theta : F \Rightarrow G$ is equivalent to both $\overline{F}_{X,Y}(\alpha) \leqslant \Gamma_{\theta_X}\,;\overline{G}_{X,Y}(\alpha)\,;\Gamma^{\perp}_{\theta_Y}$ and $\overline{F}_{X,Y}(\alpha)\,;\Gamma_{\theta_Y} \leqslant \Gamma_{\theta_X}\,;\overline{G}_{X,Y}(\alpha)$, for $\alpha \in R(X,Y)$.

It is easy to see that 1-arrows actually strictly preserve the converse, since it is an involution, and laxly preserve graphs of arrows, that is, $\Gamma_{\widehat{F}f} \leqslant \overline{F}_{X,Y}(\Gamma_f)$ and $\Gamma^{\perp}_{\widehat{F}f} \leqslant \overline{F}_{Y,X}(\Gamma^{\perp}_f)$, for every arrow $f : X \to Y$ in the base of R. A 1-arrow is called *strict* if it strictly preserves relational identities and composition. In this case, it also strictly preserves graphs of arrows. We denote by $\mathbf{RD_s}$ tns the 2-full 2-subcategory of $\mathbf{RD}$ where 1-arrows are strict.

▶ **Example 7** (Relation lifting)**.** A key notion used in relational methods is that of *relation lifting* or *lax extension* or *relator* [6, 32, 56]. It can be used to formulate bisimulation for coalgebras or other notions of program equivalence. A (conversive) relation lifting of a functor $F : \mathcal{S}et \to \mathcal{S}et$ is a family of monotonic maps $\overline{F}_{X,Y} : \mathsf{Rel}(X,Y) \to \mathsf{Rel}(FX,FY)$, indexed by sets X and Y, such that $\overline{F}_{X;Y}(\alpha)^{\perp} \subseteq \overline{F}_{Y,X}(\alpha^{\perp})$, $\overline{F}_{X,Y}(\alpha)\,;\overline{F}_{Y,Z}(\beta) \subseteq \overline{F}_{X,Z}(\alpha\,;\beta)$ and $Ff \subseteq \overline{F}_{X,Y}(f)$, where α and β are relations and $f : X \to Y$ is a function. Note that in the last condition we are using the function to denote its graph, which is perfectly fine since set-theoretic functions coincide with their graph. It is easy to see that these requirements ensure that $\langle F, \overline{F}\rangle : \mathsf{Rel} \to \mathsf{Rel}$ is a 1-arrow in $\mathbf{RD}$. Conversely any 1-arrow $G : \mathsf{Rel} \to \mathsf{Rel}$ is such that $\overline{G}$ is a relation lifting of $\widehat{G}$, showing that 1-arrows between Rel and Rel are exactly the relation liftings. Hence, 1-arrows of the form $F : R \to R$ in $\mathbf{RD}$ can be regarded as a generalisation of relation lifting to an arbitrary relational doctrine R.

Finally, relying on the 2-categorical structure of $\mathbf{RD}$, we get for free a notion of monad on a relational doctrine. A monad consists of a 1-arrow $T : R \to R$ together with 2-arrows $\eta : \mathsf{Id}_R \Rightarrow T$ and $\mu : T \circ T \Rightarrow T$ satisfying usual diagrams:

$$\begin{tikzcd}
T \arrow[r, Rightarrow, "\eta T"] \arrow[dr, Rightarrow, "\mathsf{id}"'] & T^2 \arrow[d, Rightarrow, "\mu"] & T \arrow[l, Rightarrow, "T\eta"'] \arrow[dl, Rightarrow, "\mathsf{id}"] \\
 & T &
\end{tikzcd}
\qquad
\begin{tikzcd}
T^3 \arrow[r, Rightarrow, "T\mu"] \arrow[d, Rightarrow, "\mu T"'] & T^2 \arrow[d, Rightarrow, "\mu"] \\
T^2 \arrow[r, Rightarrow, "\mu"] & T
\end{tikzcd}$$

Thanks to the conditions that 2-arrows in $\mathbf{RD}$ have to satisfy, such monads capture precisely the notion of monadic relation lifting used to reason about effectful programs [26]. Similarly, comonads in $\mathbf{RD}$ abstracts comonadic relation liftings [19].

▶ **Example 8.** Recall V-$\mathsf{Rel} : (\mathcal{S}et \times \mathcal{S}et)^{\mathrm{op}} \to \mathcal{P}os$ the doctrine of V-relations from Example 3(1). Consider the 1-arrow $P : V\text{-}\mathsf{Rel} \to V\text{-}\mathsf{Rel}$ where $\widehat{P} : \mathcal{S}et \to \mathcal{S}et$ is the covariant powerset functor and $\overline{P}_{X,Y} : V\text{-}\mathsf{Rel}(X,Y) \to V\text{-}\mathsf{Rel}(\widehat{P}X, \widehat{P}Y)$ maps a V-relation α to the function $\overline{P}_{X,Y}(\alpha)(A,B) = h_{\alpha}(A,B) \wedge h_{\alpha^{\perp}}(B,A)$ where $\wedge$ denotes the binary meet operation in V and for every $\beta : Z \times W \to |V|$, we set

$$h_{\beta}(A,B) = \bigwedge_{x \in A} \bigvee_{y \in B} \beta(x,y) \qquad \text{for } A \subseteq Z \text{ and } B \subseteq W$$

It is easy to check that this is indeed a 1-arrow. In particular, when considering the boolean quantale $\mathbb{B}$, given $\alpha : X \times Y \to \{0,1\}$ we have that $\overline{P}_{X,Y}(\alpha)$ relates A and B iff for all $x \in A$, there is $y \in B$ s.t. $\alpha(x,y) = 1$ and viceversa; considering instead Lawvere's quantale $\mathbb{R}_{\geqslant 0}$, $\overline{P}_{X,Y}(\alpha)$ is a generalisation to arbitrary $\mathbb{R}_{\geqslant 0}$-relations of the Hausdorff pseudometric on subsets of (pseudo)metric spaces.

▶ **Example 9.** Let Ω be a signature of function symbols with finite arity. Denote by Ω_X the signature obtained from Ω by adding a constant symbol for every element in X. Write $\widehat{T_\Omega}X$ for the set of closed Ω_X-terms. It is known that $\widehat{T_\Omega}$ extends to a monad on $\mathcal{S}et$. Consider the doctrine V-Rel of V-relations (cf. Example 3(1)). Every V-relation $\alpha \in V\text{-}\mathsf{Rel}(X,Y)$ can be extended to a V-relation $\alpha^\star \in V\text{-}\mathsf{Rel}(\widehat{T_\Omega}X, \widehat{T_\Omega}Y)$ by induction on the structure of terms: $\alpha^\star(x,) = \alpha(x,y)$, if $x \in X$ and $y \in Y$, $\alpha^\star(f(t_1,\dots,t_n), f(s_1,\dots,s_n)) = \bigwedge_{i\in 1..n} \alpha^\star(t_i,s_i)$, if f is an n-ary symbol of Ω, and $\alpha^\star(t,s) = \bot$, otherwise. We set $\overline{T_\Omega}_{X,Y}(\alpha) = \alpha^\star$. Then, it is not difficult to see that $T_\Omega : \mathbb{R}_{\geqslant 0}\text{-}\mathsf{Rel} \to \mathbb{R}_{\geqslant 0}\text{-}\mathsf{Rel}$ is a monad in $\mathbf{RD}$.

▶ **Example 10** (Bisimulations)**.** We can express the notion of bisimulation for coalgebras in an arbitrary relational doctrine, thus covering both usual and quantitative versions of bisimulation. If $F : R \to R$ is a 1-arrow in $\mathbf{RD}$ and $\langle X, c\rangle$ and $\langle Y, d\rangle$ two $\widehat{F}$-coalgebras, then a relation $\alpha \in R(X,Y)$ is a F-bisimulation from $\langle X, c\rangle$ to $\langle Y, d\rangle$ if $\alpha \leqslant \Gamma_c \,;\overline{F}_{X,Y}(\alpha)\,;\Gamma_d^\perp$ or, equivalently, $\alpha\,;\Gamma_d \leqslant \Gamma_c\,;\overline{F}_{X,Y}(\alpha)$. This means that α has to agree with the dynamics of the two coalgebras. Indeed, if R is Rel (the doctrine of set-theoretic relations), this condition states that, if $x \in X$ is related to $y \in Y$ by α and y evolves to $B \in \widehat{F}Y$ through d, then x evolves to some $A \in \widehat{F}X$ through c and A is related to B by the lifted relation $\overline{F}_{X,Y}(\alpha)$. This definition looks very much like that of simulation, but, since 1-arrows preserve the converse, it is easy to check that, if α is a bisimulation, then $\alpha^\perp$ is a bisimulation as well, thus justifying the name. Furthermore, one can easily check that F-bisimulations are closed under relational identities and composition. Then, the category of $\widehat{F}$-coalgebras is the base of a relational doctrine bisim^F where relations in $\mathsf{bisim}^F(\langle X, c\rangle, \langle Y, d\rangle)$ are F-bisimulations between coalgebras $\langle X, c\rangle$ and $\langle Y, d\rangle$.

As a concrete example, let us consider the 1-arrow $P : V\text{-}\mathsf{Rel} \to V\text{-}\mathsf{Rel}$ of Example 8. A $\widehat{P}$-coalgebra is a usual (non-deterministic) transition system and a P-bisimulation from $\langle X, c\rangle$ to $\langle Y, d\rangle$ is a V-relation $\alpha : X \times Y \to |V|$ such that $\alpha(x,y) \preceq h_\alpha(c(x), d(y)) \wedge h_\alpha(d(y), c(x))$, for all $x \in X$ and $y \in Y$. Roughly, this means that similar states reduce to similar states. When considering the boolean quantale $\mathbb{B}$, we get the usual notion of bisimulation, while considering Lawvere's quantale $\mathbb{R}_{\geqslant 0}$ we get a form of metric bisimulation.

▶ **Example 11** (Barr lifting)**.** Let $\mathcal{C}$ be a category with weak pullbacks and $F : \mathcal{C} \to \mathcal{C}$ a weak pullbacks preserving functor. It induces a strict 1-arrow $\langle F, \overline{F}\rangle : \mathsf{Spn}^{\mathcal{C}} \to \mathsf{Spn}^{\mathcal{C}}$ mapping a span $X \xleftarrow{p_1} A \xrightarrow{o_2} Y$ to $FX \xleftarrow{Fp_1} FA \xrightarrow{Fp_2} FY$. This provides an abstract version of the well-known Barr lifting for set-theoretic relations. It is easy to see that this construction extends to a 2-functor Spn^{-} from the 2-category of categories with weak pullbacks, functor preserving them and natural transformations to the 2-category $\mathbf{RD}$. Hence, every weak pullbacks preserving monad on a category $\mathcal{C}$ with weak pullbacks, induces a monad on $\mathsf{Spn}^{\mathcal{C}}$.

3 The Relational Quotient Completion

Here we show how one can deal with quotients in relational doctrines extending the quotient completion in [41, 42] which we used as inspiration for many notions and constructions. We present instances having a quantitative flavour that usual doctrines do not cover, showing that quotients are the key structure characterising them.

In a relational doctrine $R : (\mathcal{C} \times \mathcal{C})^{\mathrm{op}} \to \mathcal{P}os$ an *R-equivalence relation* on an object X in $\mathcal{C}$ is a relation $\rho \in R(X,X)$ satisfying the following properties:

$$\text{reflexivity: } \mathsf{d}_X \leqslant \rho \qquad \text{symmetry: } \rho^\perp \leqslant \rho \qquad \text{transitivity: } \rho\,;\rho \leqslant \rho$$

▶ **Example 12.**

1. In the doctrine of V-relations V-Rel (cf. Example 3(1)), an equivalence relation $\rho : X \times X \to |V|$ on a set X is a (symmetric) V-metric [29]: reflexivity is $1 \preceq \rho(x,x)$, for all $x \in X$, symmetry is $\rho(x,y) \preceq \rho(y,x)$, for all $x,y \in X$, and transitivity is $\bigvee_{y \in X} \rho(x,y) \cdot \rho(y,z) \preceq \rho(x,z)$, which is equivalent to $\rho(x,y) \cdot \rho(y,z) \preceq \rho(x,z)$, for all $x,y,z \in X$, by properties of suprema. For the boolean quantale $\mathbb{B}$ these are usual equivalence relations, while for the Lawvere's quantale $\mathbb{R}_{\geqslant 0}$ these are the so-called pseudometrics as the transitivity property is exactly the triangular inequality.
2. In the doctrine $\mathsf{Spn}^{\mathcal{C}}$ (cf. Example 3(3)) of spans in a category with weak pullbacks, an equivalence relation on X is a pair of parallels arrows $r_1, r_2 : A \to X$ such that there are arrows $r : X \to A$ with $r_1 r = r_2 r = \mathsf{id}_X$ (reflexivity), $s : A \to A$ with $r_1 s = r_2$ and $r_2 s = r_1$ (symmetry), and $t : W \to A$ with $r_1 t = r_1 d_1$ and $r_2 t = r_2 d_2$ where

$$\begin{tikzcd}
& W \arrow[dl, "d_1"'] \arrow[dr, "d_2"] & \\
A \arrow[dr, "r_2"'] & wpb & A \arrow[dl, "r_1"] \\
& X &
\end{tikzcd}$$

   is a weak pullback. These spans are the pseudo-equivalence relations of [14, 15].
3. In the relational doctrine $\mathsf{Vec} : (\mathcal{V}ec \times \mathcal{V}ec)^{\mathrm{op}} \to \mathcal{P}os$ (cf. Example 3(4)) an equivalence relation over a vector space X is a subadditive and homogeneous function $\rho : |X| \times |X| \to [0,\infty]$ such that $\rho(\mathbf{x},\mathbf{x}) = 0$ as reflexivity suffices to get symmetry and transitivity. Indeed one can prove that $\rho(\mathbf{x},\mathbf{y}) = \rho(\mathbf{0}, \mathbf{y}-\mathbf{x})$. Symmetry follows from $\rho(\mathbf{x},\mathbf{y}) = \rho(\mathbf{0},\mathbf{y}-\mathbf{x}) = |-1|\rho(\mathbf{0},\mathbf{x}-\mathbf{y}) = \rho(\mathbf{y},\mathbf{x})$ and transitivity from $\rho(\mathbf{x},\mathbf{y}) + \rho(\mathbf{y},\mathbf{z}) = \rho(\mathbf{0},\mathbf{y}-\mathbf{x}) + \rho(\mathbf{0},\mathbf{z}-\mathbf{y}) \geqslant \rho(\mathbf{0}+\mathbf{0}, (\mathbf{y}-\mathbf{x}) + (\mathbf{z}-\mathbf{y})) = \rho(\mathbf{0},\mathbf{z}-\mathbf{x}) = \rho(\mathbf{x},\mathbf{z})$. Hence a Vec-equivalence on a vector space X is a subadditive and homogeneous pseudometric on it.

Every arrow $f : X \to Y$ in $\mathcal{C}$ induces a R-equivalence relation on X, dubbed *kernel* of f, given by $\Gamma_f \,;\Gamma_f^{\perp}$. The fact that this is an equivalence follows immediately since Γ_f is a total and functional relation. Roughly, the kernel of f relates those elements which are identified by f; indeed, for the relational doctrine $\mathsf{Rel} : (\mathcal{S}et \times \mathcal{S}et)^{\mathrm{op}} \to \mathcal{P}os$ it is defined exactly in this way. Kernels are crucial to talk about quotients as the following definition shows.

▶ **Definition 13.** *Let $R : (\mathcal{C} \times \mathcal{C})^{\mathrm{op}} \to \mathcal{P}os$ be a relational doctrine on $\mathcal{C}$ and ρ a R-equivalence relation on an object X in $\mathcal{C}$. A* quotient arrow *of ρ is an arrow $q : X \to W$ in $\mathcal{C}$ such that $\rho \leqslant \Gamma_q \,;\Gamma_q^{\perp}$ and, for every arrow $f : X \to Z$ with $\rho \leqslant \Gamma_f \,;\Gamma_f^{\perp}$, there is a unique arrow $h : W \to Z$ such that $f = h \circ q$. The quotient arrow q is* effective *if $\rho = \Gamma_q \,;\Gamma_q^{\perp}$ and it is* descent *if $\mathsf{d}_W \leqslant \Gamma_q^{\perp} \,;\Gamma_q$.*

We say that R has quotients *if every R-equivalence relation admits an effective descent quotient arrow.*

Intuitively, a quotient of ρ is the "smallest" arrow q which transforms the equivalence ρ into the relational identity, that is, such that ρ is smaller than the kernel of q. The quotient q is effective when its kernel $\Gamma_q \,;\Gamma_q^{\perp}$ coincides with the equivalence relation ρ and it is descent when its graph is surjective.

▶ **Example 14.** To exemplify the definition above, let us unfold it for the relational doctrine $\mathsf{Rel} : (\mathcal{S}et \times \mathcal{S}et)^{\mathrm{op}} \to \mathcal{P}os$, which has quotients. Recall from Example 12(1) that a Rel-equivalence is just a usual equivalence relation. Here, a quotient arrow for an equivalence relation ρ on a set X is a function $q : X \to W$ which is universal among those functions f

whose kernel includes the equivalence ρ, that is, such that $\rho(x,x')$ implies $f(x) = f(x')$. Effectiveness requires the converse inclusion, *i.e.* $q(x) = q(x')$ implies $\rho(x,x')$. Finally, the descent condition amounts to requiring q to be surjective in the usual sense. A choice for such a function q is the usual quotient projection from X to the set X/ρ of ρ-equivalence classes, which maps $x \in X$ to its equivalence class $[x]$. Indeed, by definition this function is surjective and $\rho(x,x')$ holds iff $[x] = [x']$. Moreover, for every function f such that $\rho(x,x')$ implies $f(x) = f(x')$, the function $[x] \mapsto f(x)$ turns out to be well-defined, proving that the quotient projection is universal.

▶ **Example 15.** Consider the relational doctrine $\mathbb{R}_{\geqslant 0}$-Rel of $\mathbb{R}_{\geqslant 0}$-relations where $\mathbb{R}_{\geqslant 0}$ is the Lawvere's quantale $\langle [0,\infty], \geqslant, +, 0\rangle$ (cf. Example 3(1)) and suppose $\rho : X \times X \to [0,\infty]$ is a $\mathbb{R}_{\geqslant 0}$-Rel-equivalence relation , *i.e.* a pseudometric on X (cf. Example 12(1)). Define an equivalence relation on X setting $x \sim_\rho y$ whenever $\rho(x,x') \neq \infty$, that is, when x and x' are connected. The canonical surjection $q : X \to X/\sim$ mapping x to $q(x) = [x]$ is a quotient arrow for ρ. It is immediate to see that $\rho(x,x') \geqslant \mathsf{d}_{X/\sim_\rho}([x],[x'])$ as $\mathsf{d}_X([x],[x'])$ is either 0 or ∞ and $\mathsf{d}_{X/\sim_\rho}([x],[x']) = \infty$ precisely when x and x' are not connected, that is, when $\rho(x,x') = \infty$. The universality of q easily follows from its universal property as a quotient of $\sim_\rho$ in Rel (cf. Example 14). This shows that $\mathbb{R}_{\geqslant 0}$-Rel has quotient arrows for all pseudometrics, which are descent: for $q : X \to X/\sim_\rho$ a quotient of ρ, the descent condition becomes $\mathsf{d}_{X/\sim_\rho}(y,y') \geqslant \inf_{x\in X}\left(\mathsf{d}_{X/\sim_\rho}(y,q(x)) + \mathsf{d}_{X/\sim_\rho}(q(x),y')\right)$, which trivially holds since q is surjective and $\mathsf{d}_{X/\sim_\rho}$ is either 0 or ∞. However, such quotients arrows cannot be effective. Indeed, if $f : X \to Y$ is a function, since the relational identity d_Y is only either 0 or ∞, the kernel of f is given by $x, x' \mapsto \mathsf{d}_Y(f(x), f(x'))$, thus it takes values in $\{0,\infty\}$. Hence, if a quotient arrow q for a pseudometric ρ was effective, then ρ would be either 0 or ∞, as it would coincide with the kernel of q and clearly this is not the case in general. This shows that $\mathbb{R}_{\geqslant 0}$-Rel has not quotients in the sense of Definition 13.

Example 15 shows that relational doctrines need not have quotients in general. Hence, we now describe a free construction that takes a relational doctrine $R : (\mathcal{C} \times \mathcal{C})^{\text{op}} \to \mathcal{P}os$ and builds a new one $(R)^q : (\mathcal{Q}_R \times \mathcal{Q}_R)^{\text{op}} \to \mathcal{P}os$ which has (effective descent) quotients for all equivalence relations. The construction is inspired by the quotient completion in [41, 42] and a comparison with it is delayed to Section 5.

The category $\mathcal{Q}_R$ is defined as follows:

- an object is a pair $\langle X, \rho\rangle$, where X is an object in $\mathcal{C}$ and ρ is a R-equivalence relation on X,
- an arrow $f : \langle X, \rho\rangle \to \langle Y, \sigma\rangle$ is an arrow $f : X \to Y$ in $\mathcal{C}$ such that $\rho \leqslant R_{f,f}(\sigma)$, and
- composition and identities are those of $\mathcal{C}$.

By Proposition 6 the condition $\rho \leqslant R_{f,f}(\sigma)$ is equivalent to both $\rho \leqslant \Gamma_f \,;\sigma\,; \Gamma_f^\perp$ and $\Gamma_f^\perp \,;\rho\,;\Gamma_f \leqslant \sigma$.

Given R-equivalence relations ρ and σ over X and Y the suborder $\mathcal{D}es_{\rho,\sigma}(X,Y)$ of $R(X,Y)$ of *descent data* with respect to ρ and σ is defined by

$$\mathcal{D}es_{\rho,\sigma}(X,Y) = \{\alpha \in R(X,Y) \mid \rho^\perp \,;\alpha\,;\sigma \leqslant \alpha\}$$

Roughly, a descent datum is a relation which is closed w.r.t. ρ on the left and σ on the right. For every arrow $f : \langle X, \rho\rangle \to \langle X', \rho'\rangle$ and $g : \langle Y, \sigma\rangle \to \langle Y', \sigma'\rangle$ in $\mathcal{Q}_R$, the monotone function $R_{f,g} : R(X',Y') \to R(X,Y)$ applies $\mathcal{D}es_{\rho',\sigma'}(X',Y')$ into $\mathcal{D}es_{\rho,\sigma}(X,Y)$ as Indeed, for $\alpha \in \mathcal{D}es_{\rho',\sigma'}(X',Y')$, we have

$$\rho^\perp \,; R_{f,g}(\alpha)\,;\sigma \leqslant R_{f,f}(\rho'^\perp)\,; R_{f,g}(\alpha)\,; R_{g,g}(\sigma') \leqslant R_{f,g}(\rho'^\perp \,;\alpha\,;\sigma') \leqslant R_{f,g}(\alpha)$$

Therefore the assignments $(R)^q(\langle X,\rho\rangle,\langle Y,\sigma\rangle) = \mathcal{D}es_{\rho,\sigma}(X,Y)$ and $(R)^q_{f,g} = R_{f,g}$ determine a functor $(R)^q : (\mathcal{Q}_R \times \mathcal{Q}_R)^{\mathrm{op}} \to \mathcal{P}os$.

▶ **Proposition 16.** *The functor $(R)^q : (\mathcal{Q}_R \times \mathcal{Q}_R)^{\mathrm{op}} \to \mathcal{P}os$ is a relational doctrine, where composition and converse are those of R and $\mathsf{d}_{\langle X,\rho\rangle} = \rho$.*

A $(R)^q$-equivalence relation over an object $\langle X,\rho\rangle$ is a R-equivalence σ over X such that $\rho \leqslant \sigma$. Note that these conditions imply that σ is a descent datum in $\mathcal{D}es_{\rho,\rho}(X,X)$. Then, $\langle X,\sigma\rangle$ is an object of $\mathcal{Q}_R$ and $\mathsf{id}_X : \langle X,\rho\rangle \to \langle X,\sigma\rangle$ is a well-defined arrow in $\mathcal{Q}_R$, which turns out to be an effective descent quotient arrow for σ. In this way we construct quotient arrows for all $(R)^q$-equivalence relations, thus obtaining the following result.

▶ **Proposition 17.** *The relational doctrine $(R)^q$ over $\mathcal{Q}_R$ has effective descent quotients.*

▶ **Example 18.**

1. For the doctrine V-Rel of V-relations, the category $\mathcal{Q}_{V\text{-}\mathsf{Rel}}$ is the category of V-metric spaces with non-expansive maps. By Example 12(1), an object $\langle X,\rho\rangle$ is a V-metric space and $f : \langle X,\rho\rangle \to \langle Y,\sigma\rangle$ has to satisfy $\rho(x,x') \preceq \sigma(f(x),f(x'))$.
2. For the relational doctrine Vec over the category of real vector spaces, $\mathcal{Q}_{\mathsf{Vec}}$ is the category of semi-normed vector spaces with short maps. An object $\langle X,\rho\rangle$ in $\mathcal{Q}_{\mathsf{Vec}}$ is a vector space with a subadditive and homogeneous pseudometric on it. Such a pseudometric satisfies $\rho(\mathbf{x},\mathbf{y}) = \rho(\mathbf{0},\mathbf{y}-\mathbf{x})$ (see Example 12(3)), so $\|\mathbf{x}\| = \rho(\mathbf{0},\mathbf{x})$ defines a semi-norm on X.

Following Lawvere's structural approach to logic, we can characterise the property of having effective descent quotients by an adjunction in **RD**. First observe that the doctrine R is embedded into $(R)^q$ by the 1-arrow $E^R : R \to (R)^q$ in **RD** defined as follows: the functor $\widehat{E^R} : \mathcal{C} \to \mathcal{Q}_R$ maps $f : X \to Y$ in $\mathcal{C}$ to $f : \langle X,\mathsf{d}_X\rangle \to \langle Y,\mathsf{d}_Y\rangle$; the natural transformation $\overline{E^R} : R \xrightarrow{\cdot} (R)^q \circ (\widehat{E^R} \times \widehat{E^R})^{\mathrm{op}}$ is the family of identities $R(X,Y) = \mathcal{D}es_{\mathsf{d}_X,\mathsf{d}_Y}(X,Y)$. The 1-arrow E^R shows that constructing $(R)^q$ "extends" R adding (effective descent) quotients for any equivalence relation.

▶ **Lemma 19.** *A relational doctrine R has effective descent quotients if and only if E^R has a strict reflection left adjoint $F : (R)^q \to R$.*

This means that the 1-arrow $F : (R)^q \to R$ is strict and it is a left adjoint of E^R in **RD** and the counit of this adjunction is an isomorphism, hence $F \circ E^R \cong \mathsf{Id}_R$. Intuitively, the 1-arrow $F : (R)^q \to R$ computes quotients of R-equivalence relations: the object $\widehat{F}\langle X,\rho\rangle$ is the codomain of a quotient arrow obtained by applying $\widehat{F}$ to $\mathsf{id}_X : \langle X,\mathsf{d}_X\rangle \to \langle X,\rho\rangle$ which is the quotient arrow of ρ in $(R)^q$ viewed as a $(R)^q$-equivalence over $\langle X,\mathsf{d}_X\rangle$.

The construction of $(R)^q$ is universal as it is part of a lax 2-adjunction [10]. To show this, we first introduce the 2-category **QRD** as the 2-full 2-subcategory of **RD** whose objects are relational doctrines with quotients and whose 1-arrows are those of **RD** that preserve quotient arrows, *i.e.* 1-arrows $F : R \to S$ in **RD** mapping a quotient arrow for a R-equivalence ρ over X to a quotient arrow for $\overline{F}_{X,X}(\rho)$, which can be easily proved to be a S-equivalence over $\widehat{F}X$. There is an obvious inclusion 2-functor $\mathrm{U_q} : \mathbf{QRD} \to \mathbf{RD}$ which simply forgets quotients. Moreover, the construction above determines a 2-functor $\mathrm{Q} : \mathbf{RD} \to \mathbf{QRD}$, defined as follows: for a 1-arrow $F : R \to S$ in **RD**, the 1-arrow $\mathrm{Q}(F) = (F)^q : (R)^q \to (S)^q$ is given by $\widehat{(F)^q}\langle X,\rho\rangle = \langle \widehat{F}X, \overline{F}_{X,X}(\rho)\rangle$ and $\widehat{(F)^q}f = \widehat{F}f$ and $\overline{(F)^q}_{\langle X,\rho\rangle,\langle Y,\sigma\rangle}(\alpha) = \overline{F}_{X,Y}(\alpha)$, and for a 2-arrow $\theta : F \Rightarrow G$ in **RD**, the 2-arrow $\mathrm{Q}(\theta) = (\theta)^q : (F)^q \Rightarrow (G)^q$ is given by $(\theta)^q_{\langle X\rho\rangle} = \theta_X$.

▶ **Theorem 20.** *The 2-functors Q and $\mathrm{U_q}$ are such that $\mathrm{Q} \dashv_l \mathrm{U_q}$ is a lax 2-adjunction.*

This means that, for every relational doctrine R and every relational doctrine with quotients S, the functor

$$\mathrm{U_q}(-) \circ E^R : \mathbf{QRD}((R)^q, S) \to \mathbf{RD}(R, \mathrm{U_q}(S)) \tag{1}$$

has a left adjoint.

▶ **Example 21.** Let R be a relational doctrine with quotients and $F : R \to R$ be a 1-arrow in $\mathbf{QRD}$, that is, it preserves quotient arrows. Recall from Example 10 the doctrine bisim^F on the category $\mathcal{CoAlg}(\widehat{F})$ of $\widehat{F}$-coalgebras, where relations between coalgebras are F-bisimulations. It is easy to see that bisim^F has quotients. Indeed, a bisim^F-equivalence relation ρ on a $\widehat{F}$-coalgebra $\langle X, c\rangle$ is an F-bisimulation which is also a R-equivalence relaiton on X. Since R has quotients, ρ admits an effective descent quotient arrow $q : X \to W$ in the base of R. To conclude, it suffices to endow W with an $\widehat{F}$-coalgebra structure, making q an $\widehat{F}$-coalgebra homomorphism. To this end, note that, since ρ is a F-bisimulation and $\widehat{F}q$ is a quotient arrow for $\overline{F}_{X,X}(\rho)$, we get $\rho \leqslant \Gamma_{\widehat{F}q\circ c} ; \Gamma^{\perp}_{\widehat{F}q\circ c}$. Thus by the universal property of quotients, we get a unique arrow $c_\rho : W \to \widehat{F}W$ making the following diagram commute:

$$\begin{tikzcd}
X \arrow[r, "q"] \arrow[d, "c"'] & W \arrow[d, dotted, "c_\rho"] \\
\widehat{F}X \arrow[r, "\widehat{F}q"] & \widehat{F}W
\end{tikzcd}$$

This shows that the doctrine of F bisimulations inherits quotients, provided that F preserves them. If however quotients are not available in R and/or F does not preserve them, we can use the relational quotient completion to freely add them to bisim^F. In this way, we get the doctrine $(\mathsf{bisim}^F)^q$ whose base category has as objects triple $\langle X, c, \rho\rangle$ where $\langle X, c\rangle$ is an $\widehat{F}$-coalgebra and ρ is an F-bisimulation equivalence on it. Notice that, applying Q to the 1-arrow F, we get a 1-arrow $(F)^q : (R)^q \to (R)^q$. Then, we can construct the doctrine $\mathsf{bisim}^{(F)^q}$ of $(F)^q$-bisimulations. It is easy to check that $(\mathsf{bisim}^F)^q$ is isomorphic to $\mathsf{bisim}^{(F)^q}$, that is, the costruction of coalgebras commutes with the quotient completion.

▶ **Example 22.** Let Ω be a signature of function symbols with finite arity. Recall from Example 9 that we have the monad $T_\Omega : V\text{-}\mathsf{Rel} \to V\text{-}\mathsf{Rel}$ of terms over Ω. Applying the relational quotient completion, since it is a 2-functor, we get a monad $(T_\Omega)^q : (V\text{-}\mathsf{Rel})^q \to (V\text{-}\mathsf{Rel})^q$. In particular, we get a monad $\widehat{(T_\Omega)^q} : \mathcal{Q}_{V\text{-}\mathsf{Rel}} \to \mathcal{Q}_{V\text{-}\mathsf{Rel}}$ on the category of V-metric spaces (the base of $(V\text{-}\mathsf{Rel})^q$), which is a slight generalisation of the free monad for quantitative algebras over Ω described in [1, 2].

We conjecture that a similar construction should be possible also when considering a Quantitative Equational Theory [45, 46, 4, 1] over Ω, extending the construction in that papers to V-Rel. However, this is still an open problem, we leave for future work.

The 2-adjunction of Theorem 20, being lax, establishes a weak correspondence between $\mathbf{RD}$ and $\mathbf{QRD}$: between their hom-categories there is neither an isomorphism, nor an equivalence, but just an adjunction. Moreover, the family of 1-arrows E^R is only a lax natural transformation. This is essentially due to the fact that 1-arrows of $\mathbf{RD}$ and $\mathbf{QRD}$ laxly preserve relational operations, in particular, relational identities. Hence, a way to recover a stronger correspondence may be to restrict to strict 1-arrows.

Denote by $\mathbf{QRD_s}$ the 2-full 2-subcategory of $\mathbf{QRD}$ whose 1-arrows are strict. Then, it is easy to see that Q applies $\mathbf{RD_s}$ into $\mathbf{QRD_s}$, obtaining the following result.

▶ **Theorem 23.** *The lax 2-adjunction* $\mathrm{Q} \dashv_l \mathrm{U_q}$ *restricts to a (pseudo) 2-adjunction between* $\mathbf{QRD_s}$ *and* $\mathbf{RD_s}$*.*

This means that the family of 1-arrows E^R becomes a strict 2-natural transformation and the functor in Equation (1) becomes an equivalence of categories when restricted to $\mathbf{QRD_s}$ and $\mathbf{RD_s}$.

4 Extensionality and separation

An important logical principle commonly assumed is the *extensionality* of equality. Intuitively, it means that two functions f and g are equal exactly when their outputs coincide on equal inputs, that is, whenever $x = y$ implies $f(x) = g(y)$. This is the usual notion of equality for set-theoretic functions, however, if we move to more constructive settings such as Type Theory, it is not necessarily the case that extensionality holds. Relational doctrines are able to distinguish the two notions of equality of arrows.

▶ **Definition 24.** *Let* $R : (\mathcal{C} \times \mathcal{C})^{\mathrm{op}} \to \mathcal{P}os$ *be a relational doctrine and* $f, g : X \to Y$ *two parallel arrows in* $\mathcal{C}$*. We say that* f *and* g *are* R-equal*, notation* $f \approx g$*, if* $\mathsf{d}_X \leqslant R_{f,g}(\mathsf{d}_Y)$*. We say that* R *is* extensional *if for every* f, g *in* $\mathcal{C}$*,* $f \approx g$ *implies* $f = g$*.*

That is, R is extensional if R-equality implies equality of arrows. The other implication always holds, therefore in an extensional relational doctrine $f \approx g$ if and only if $f = g$.

▶ **Proposition 25.** *Let* $R : (\mathcal{C} \times \mathcal{C})^{\mathrm{op}} \to \mathcal{P}os$ *be a relational doctrine and* $f, g : X \to Y$ *two parallel arrows in* $\mathcal{C}$*. Then,* $f \approx g$ *iff* $\Gamma_f = \Gamma_g$*.*

Proposition 25 with Proposition 6 mean that R-equal arrows cannot be distinguished by the logic of R since they behave in the same way w.r.t. reindexing. Indeed given $f, f' : X \to A$ and $g, g' : Y \to B$ in the base, $f \approx f'$ and $g \approx g'$ imply $R_{f,g} = R_{f',g'}$.

From a quantitative or topological perspective, extensional equality is related to various notions of *separation*. Take for example the doctrine $(\mathbb{R}_{\geqslant 0}\text{-}\mathsf{Rel})^q$ over the category $\mathcal{Q}_{\mathbb{R}_{\geqslant 0}\text{-}\mathsf{Rel}}$ of pseudometric spaces and non-expansive maps (cf. Example 18(1)). Functions $f, g : \langle X, \rho\rangle \to \langle Y, \sigma\rangle$ are $(\mathbb{R}_{\geqslant 0}\text{-}\mathsf{Rel})^q$-equal iff $\sigma(f(x), g(x)) = 0$, which implies $f = g$ exactly when $\langle Y, \sigma\rangle$ satisfies the identity of indiscernibles, *i.e.* the axiom stating that $\sigma(x, y) = 0$ implies $x = y$. This requirement turns a pseudometric space into a usual metric space and forces a strong separation property: the topology associated with the metric space is Hausdorff.

This observation shows that the relational quotient completion does not preserve extensionality. Indeed the relational doctrine $\mathbb{R}_{\geqslant 0}\text{-}\mathsf{Rel}$ on $\mathcal{S}et$ is extensional, while $(\mathbb{R}_{\geqslant 0}\text{-}\mathsf{Rel})^q$ is not as not all pseudometric spaces are separated. This is due to the fact that the relational quotient completion changes equality, as it modifies identity relations, while the equality between arrows of the base category remains unchanged. The following completion forces extensionality or, in quantitative terms, separation. As for the relational quotient completion, it is inspired by the extensional collapse of an elementary doctrines introduced in [42].

▶ **Proposition 26.** *Let* R *be a relational doctrine and* $f, f' : X \to Y$ *and* $g, g' : Y \to Z$ *are arrows in the base* $\mathcal{C}$*. Then* $f \approx f'$ *and* $g \approx g'$ *imply* $g \circ f \approx g' \circ f'$*.*

It shows that $\approx$ is a congruence on $\mathcal{C}$. Let $\mathcal{E}_R$ be the quotient of $\mathcal{C}$ modulo $\approx$, notably, objects are those of $\mathcal{C}$ and arrows are equivalence classes of arrows in $\mathcal{C}$ modulo $\approx$, denoted by $[f]$. Define a functor $(R)^e : (\mathcal{E}_R \times \mathcal{E}_R)^{\mathrm{op}} \to \mathcal{P}os$ by $(R)^e(X, Y) = R(X, Y)$ and $(R)^e_{[f],[g]}(\alpha) = R_{f,g}(\alpha)$. It is well-defined on arrows by Propositions 6 and 25.

▶ **Lemma 27.** *The functor $(R)^e : (\mathcal{E}_R \times \mathcal{E}_R)^{\mathrm{op}} \to \mathcal{Pos}$ together with relational operations of R is an extensional relational doctrine.*

Taking terminology from [42], the doctrine $(R)^e$ is the *extensional collapse* of R. The following examples show some connections between the extensional collapse and notions of separation in metric and topological structures.

▶ **Example 28.**

1. Let $V = \langle |V|, \preceq, \cdot, 1\rangle$ be a commutative quantale. Recall from Example 18(1) that the category $\mathcal{Q}_{V\text{-}\mathsf{Rel}}$ is the category of V-metric spaces and non-expansive maps. It is the base of the doctrine $(V\text{-}\mathsf{Rel})^q$, whose identity relation is given by $\mathsf{d}_{\langle X,\rho\rangle} = \rho$ for every V-metric space $\langle X, \rho\rangle$. A V-metric sapce $\langle X, \rho\rangle$ is *separated* if $1 \preceq \rho(x, y)$ implies $x = y$. Notice that a separated $\mathbb{R}_{\geqslant 0}$-metric space is the usual notion of metric space. Denote by $V\text{-}\mathcal{Met}_{\mathsf{s}}$ the full subcategory of $\mathcal{Q}_{V\text{-}\mathsf{Rel}}$ of separated V-metric spaces. Applying the extensional collapse to $(V\text{-}\mathsf{Rel})^q$ we get $((V\text{-}\mathsf{Rel})^q)^e$ where two arrows $[f], [g] : \langle X, \rho\rangle \to \langle Y, \sigma\rangle$ of its base $\mathcal{E}_{(V\text{-}\mathsf{Rel})^q}$ are equal when $\rho(x, y) \preceq \sigma(f(x), g(y))$. The fully faithful inclusion of $V\text{-}\mathcal{Met}_{\mathsf{s}}$ into $\mathcal{E}_{(V\text{-}\mathsf{Rel})^q}$ is an equivalence: for any V-metric space $\langle X, \rho\rangle$, write $x \sim y$ when $1 \preceq \rho(x, y)$ and take the quotient space $\langle X/\sim, \rho_\sim\rangle$, where $\rho_\sim([x], [y]) = \rho(x, y)$, is separated. The projection map $[q] : \langle X, \rho\rangle \to \langle X/\sim, \rho_\sim\rangle$ is an isomorphism whose inverse is represented by any chosen section $s : X/\sim \to X$ of q.
2. Recall from Example 18(2) that the base $\mathcal{Q}_{\mathsf{Vec}}$ of the relational doctrine $(\mathsf{Vec})^q$ is the category of semi-normed real vector spaces and short linear maps: an object $\langle X, \rho\rangle$ is a real vector space X with a subadditive and homogeneous pseudometric ρ that gives a semi-norm $\|\mathbf{x}\| = \rho(\mathbf{0}, \mathbf{x})$. A semi-norm is a norm when $\|\mathbf{x}\| = 0$ implies $\mathbf{x} = \mathbf{0}$, which is equivalent to ρ being separated. The category $\mathcal{NVec}$ of normed vector spaces is equivalent to the base category $\mathcal{E}_{(\mathsf{Vec})^q}$ of the extensional collapse of $(\mathsf{Vec})^q$. The proof of the essential surjectivity of the obvious inclusion of $\mathcal{NVec}$ into $\mathcal{E}_{(\mathsf{Vec})^q}$ uses arguments similar to those used in Example 28(1). In particular it relies on the axiom of choice. There is only a little care in taking sections $s : X/\sim \to X$ of a quotient map q in $\mathcal{Vec}$ as these have to be linear. But from a section s one cane take its values on the vectors of a chosen base of $X/\sim$ and generate from this assignment a linear map $s' : X/\sim \to X$ which is easily proved to be a section of q.

▶ **Example 29.** Let $\mathcal{Top}$ be the category of topological spaces and continuous functions and $\mathsf{TRel} : (\mathcal{Top} \times \mathcal{Top})^{\mathrm{op}} \to \mathcal{Pos}$ be the change-of-base $U^\star\mathsf{Rel}$ along the forgetful functor $U : \mathcal{Top} \to \mathcal{Set}$ as in Example 3(5). The base $\mathcal{Q}_{\mathsf{TRel}}$ of the relational quotient completion of TRel provides an "intensional" version of Scott's equilogical spaces[2] [52]. Objects of $\mathcal{Q}_{\mathsf{TRel}}$ are pairs $\langle X, \rho\rangle$ of a topological space X and an equivalence relation ρ on the underlying set of X and arrows are continuous maps preserving the equivalences. Any section $S : \mathcal{Top} \to \mathcal{Q}_{\mathsf{TRel}}$ of the forgetful functor $\mathcal{Q}_{\mathsf{TRel}} \to \mathcal{Top}$ picks an equivalence relation over every space in a way that relations are compatible with continuous maps. The change-of-base $S^\star(\mathsf{TRel})^q$ provides a new logic on $\mathcal{Top}$ where identity relations are changed according to S. For a space X, the doctrine $S^\star\mathsf{TRel}$ can not distinguish points which are related by ρ_X, while such points may differ in the base. The extensional collapse makes such points indistinguishable in the base as well. Instances of this construction are the category $\mathcal{Top}_0$ of T_0-spaces and the homotopy category $h\mathcal{Top}$. The former is given by defining ρ_X as follows: $\langle x, y\rangle \in \rho_X$ iff x and y are topologically indistinguishable, that is, for every open subset $U \subseteq X$, $x \in U$ iff $y \in U$. The latter is given by defining ρ_X as follows: $\langle x, y\rangle \in \rho_X$ iff there is a continuous path from x to y, that is, there is a continuous function $h : [0, 1] \to X$ such that $h(0) = x$ and $h(1) = y$.

[2] Applying the extensional collapse we get exactly the category of equilogical spaces.

The relational doctrine $(R)^e$ comes together with a 1-arrow $C^R : R \to (R)^e$ where $\widehat{C^R} : \mathcal{C} \to \mathcal{E}_R$ maps $f : A \to B$ to $[f] : A \to B$ and $\overline{C^R}_{X,Y}$ maps $R(X,Y)$ to itself. The extensional collapse is universal if we restrict to strct 1-arrows. Let $\mathbf{ERD}_s$ denote the full 2-subcategory of $\mathbf{RD_s}$ whose objects are extensional relational doctrines and $\mathrm{U_e} : \mathbf{ERD}_s \to \mathbf{RD_s}$ the obvious inclusion 2-functor.

▶ **Theorem 30.** *The 2-functor* $\mathrm{U_e} : \mathbf{ERD}_s \to \mathbf{RD_s}$ *has a left 2-adjoint* $\mathrm{E} : \mathbf{RD_s} \to \mathbf{ERD}_s$ *such that* $\mathrm{E}(R) = (R)^e$.

The extensional collapse interacts well with quotients. Indeed, if R has (effective descent) quotients, its extensional collapse $(R)^e$ has (effective descent) quotients as well. More precisely, let us denote by $\mathbf{EQRD}_s$ the full 2-subcategory of $\mathbf{QRD_s}$ whose objects are extensional relational doctrines with quotients. We get two obvious inclusion 2-functors $\mathrm{U_q}' : \mathbf{EQRD}_s \to \mathbf{QRD_s}$ and $\mathbf{EQRD}_s : \mathbf{ERD}_s \to$ which respectively forget extensionality and quotients.[3] Then, we get the following result.

▶ **Theorem 31.** *The 2-adjunction* $\mathrm{E} \dashv \mathrm{U_e}$ *restricts to a 2-adjunction between* $\mathbf{EQRD}_s$ *and* $\mathbf{QRD_s}$.

In summary, by Theorems 23, 30, and 31, we get the following diagram

$$\begin{tikzcd}
\mathbf{EQRD}_s \arrow[r, bend right, "\mathrm{U_q}'"'] \arrow[r, phantom, "\perp"] \arrow[d, "\mathrm{U_e}'"'] & \mathbf{QRD_s} \arrow[l, bend right, "\mathrm{E}'"'] \arrow[d, bend right, "\mathrm{U_q}"'] \arrow[d, phantom, "\vdash"] \\
\mathbf{ERD}_s \arrow[r, bend left, "\mathrm{U_e}"] \arrow[r, phantom, "\top"] & \mathbf{RD_s} \arrow[l, bend left, "\mathrm{E}"] \arrow[u, bend right, "\mathrm{Q}"']
\end{tikzcd}$$

where the external square commutes and E' is a lifting of E, that is, $\mathrm{U_e}' \circ \mathrm{E}' = \mathrm{E} \circ \mathrm{U_q}$. The composite $\mathrm{E}' \circ \mathrm{Q} : \mathbf{RD_s} \to \mathbf{EQRD}_s$ gives a universal construction adding (effective descent) quotients and forcing extensionality. Finally note that the relational quotient completion does not preserve extensionality. Therefore the restriction of Q to $\mathbf{ERD}_s$, namely, the composite $\mathrm{Q} \circ \mathrm{U_e}$, may not provide a left 2-adjoint to $\mathrm{U_e}'$. To get such a left 2-adjoint, we need to force extensionality again, that is, we need the 2-functor $\mathrm{E}' \circ \mathrm{Q} \circ \mathrm{U_e}$, which however is not the lifting of Q (in other words, the diagram of left 2-adjoints would not commute).

▶ **Example 32.**

1. Recall from [11, 12] that a Bishop's set, or setoid, is a pair $\langle A, \rho \rangle$ of a set A and an equivalence relation $\rho \subseteq A \times A$. A Bishop's function from the setoid $\langle A, \rho \rangle$ to the setoid $\langle B, \sigma \rangle$ is an equivalence class of functions $f : A \to B$ preserving the equivalence relations, where f and g belong to the same equivalence class if $f(a) \sigma g(a)$ for all $a \in A$. A relation from $\langle A, \rho \rangle$ to $\langle B, \sigma \rangle$ is a subset $U \subset A \times B$ such that $(a, b) \in U$, $a \rho a'$, $b \sigma b'$ imply $(a', b') \in U$. Call $\mathcal{BSet}$ the category of Bishop's sets and functions and $\mathsf{BRel} : (\mathcal{BSet} \times \mathcal{BSet})^{\mathrm{op}} \to \mathcal{Pos}$ the relational doctrine that maps two setoids to the collection of relations between them. The relational doctrine $((\mathsf{Rel})^q)^e$ (obtained completing $\mathsf{Rel} : (\mathcal{Set} \times \mathcal{Set})^{\mathrm{op}} \to \mathcal{Pos}$ first with quotients and then forcing extensionality) is BRel.

[3] $\mathbf{EQRD}_s$ can be seen as the pullback of $\mathrm{U_e}$ against $\mathrm{U_q}$.

2. One of the most widely used constructions to complete a category with quotients is the exact completion of a weakly lex category presented in $\mathcal{C}$ [14, 15]. This is an instance of our constructions. Recall the relational doctrine $\mathsf{Spn}^{\mathcal{C}}$ from Example 3(3). Complete it first with quotients and then force extensionality. One get the relational doctrine $((\mathsf{Spn}^{\mathcal{C}})^q)^e$ whose base is $\mathcal{C}_{\mathrm{ex/wlex}}$. If products of $\mathcal{C}$ are strong, the construction coincides with the elementary quotient completion of the doctrines of weak subobjects of $\mathcal{C}$ shown in [41, 42]. A comparison between these two constructions is in Section 5.

5 Related Structures

There are many categorical models abstracting the essence of the calculus of relations, such as cartesian bicategories [16] or allegories [25] which are both special cases of ordered categories with involution [35]. Also existential and elementary doctrines, *i.e.* those doctrines that model $(\exists, \wedge, \top, =)$-fragment of first order logic, encode a calculus of relations. A natural question is how relational doctrines differ from these models.

We show that when working with an ordered category, one implicitly accepts two logical principles, which are not necessarily there in a relational doctrine, and we show that when working with existential elementary doctrines, one implicitly accepts to work with variables, which are not necessarily there in relational doctrines. These comparison are carried out restricting to the 2-category $\mathbf{RD_s}$ where 1-arrows are strict.

Ordered categories with involution. An ordered category with involution [35] is a $\mathcal{P}os$-enriched category $\mathcal{C}$ together with an identity-on-objects and self inverse $\mathcal{P}os$-functor $(-)^{\perp} : \mathcal{C}^{\mathrm{op}} \to \mathcal{C}$. Intuitively, arrows can be seen as relations whose inverse is given by the involution.

A relational doctrine $R : (\mathcal{C} \times \mathcal{C})^{\mathrm{op}} \to \mathcal{P}os$ defines an ordered category with involution $\mathcal{O}_R$ as follows: objects are those of $\mathcal{C}$, the poset of arrows between X and Y is the fibre $R(X, Y)$, composition and identities are given by relational ones and the involution is given by the converse operation. The assignment extends to a 2-functor $\mathcal{O} : \mathbf{RD_s} \to \mathbf{OCI}$, where $\mathbf{OCI}$ is the 2-category of ordered categories with involution whose 1-arrows $F : \mathcal{C} \to \mathcal{D}$ are ordered functors preserving involution and a 2-arrows $\theta : F \Rightarrow G$ are lax natural transformations.

To see how to obtain a relational doctrine from an ordered category, first note that any ordered category with involution $\mathcal{C}$ induces a category $\mathcal{M}ap(\mathcal{C})$, called the category of maps in $\mathcal{C}$, whose objects are those of $\mathcal{C}$ and an arrow $f : X \to Y$ is an arrow in $\mathcal{C}$ such that $f^{\perp} : Y \to X$ is its right adjoint, that is $f \circ f^{\perp} \leqslant \mathsf{id}_Y$ and $\mathsf{id}_X \leqslant f^{\perp} \circ f$. We define a relational doctrine $\mathsf{Map}^{\mathcal{C}} : (\mathcal{M}ap(\mathcal{C}) \times \mathcal{M}ap(\mathcal{C}))^{\mathrm{op}} \to \mathcal{P}os$ where $\mathsf{Map}^{\mathcal{C}}(X, Y) = \mathcal{C}(X, Y)$ is the poset of all arrows in $\mathcal{C}$ from X to Y and, for $f : A \to X$ and $g : B \to Y$ arrows in $\mathcal{M}ap(\mathcal{C})$, the map $\mathsf{Map}^{\mathcal{C}}_{f,g} : \mathsf{Map}^{\mathcal{C}}(X, Y) \to \mathsf{Map}^{\mathcal{C}}(A, B)$ sends α to the composition $g^{\perp} \circ\ \alpha \circ f$. Relational composition and identities are composition and identities of $\mathcal{C}$ and the relational converse is given by the involution $(-)^{\perp}$. The assignment extends to a 2-functor $\mathrm{Map} : \mathbf{OCI} \to \mathbf{RD_s}$.

Relational doctrines of the form $\mathsf{Map}^{\mathcal{C}}$ have extensional equality. They also validates the rule of unique choice which says that whenever a relation is functional and total, there is a function that for every x in the domain picks the unique y related to x. Formally $R : (\mathcal{C} \times \mathcal{C})^{\mathrm{op}} \to \mathcal{P}os$ satisfies the *rule of unique choice*, (RUC) for short, if for every $\alpha \in R(X, Y)$ such that $\mathsf{d}_X \leqslant \alpha\,;\alpha^{\perp}$ and $\alpha^{\perp}\,;\alpha \leqslant \mathsf{d}_Y$, there is $f : X \to Y$ in $\mathcal{C}$ with $\Gamma_f \leqslant \alpha$.

Next theorem shows that extensionality and (RUC) are exactly the two logical principles that a relational doctrine needs to coincide with an ordered category. Indeed the essential image of 2-functor Map is $\mathbf{ERD}_{(\mathrm{RUC})}$, the full 2-subcategory of $\mathbf{RD_s}$ on extensional doctrines satisfying (RUC) and its inverse is the restriction of $\mathcal{O} : \mathbf{RD_s} \to \mathbf{OCI}$ to $\mathbf{ERD}_{(\mathrm{RUC})}$.

▶ **Theorem 33.** *The 2-categories* **OCI** *and* $\mathbf{ERD}_{(\text{RUC})}$ *are 2-equivalent.*

The equivalence stated in Theorem 33 generalises a similar result proved in [13], which compares cartesian bicategories and existential elementary doctrines. The way we built the two functors of the equivalence shows also that **OCI** is a 1-full subcategory of $\mathbf{ERD}_{(\text{RUC})}$. Examples of relational doctrines that are not in **OCI** because they are not extensional were given in Section 4. The following example presents a relational doctrine outside **OCI** because it does not satisfy (RUC).

▶ **Example 34.** Take a set A with more than one element. The set $\mathcal{P}(A)$ of subsets of A is a complete Heyting algebra, therefore a commutative quantale. Recall from Item 1 that in the relational doctrine $\mathcal{P}(A)\text{-}\mathsf{Rel} : (\mathcal{S}et \times \mathcal{S}et)^{\text{op}} \to \mathcal{P}os$ of $\mathcal{P}(A)$-relations, for every set X the relation d_X maps (x, x') to A if $x = x'$ and to $\varnothing$ if $x \neq x'$. This relational doctrine does not satisfy the (RUC). Consider $\alpha \in \mathcal{P}(A)\text{-}\mathsf{Rel}(1, A)$ given by $\alpha(*, a) = \{a\}$, it holds

$$\mathsf{d}_1 = A = \bigcup_{a \in A} \{a\} = \alpha \,;\alpha^{\perp} \quad \text{and} \quad (\alpha^{\perp}\,; \alpha)(a, a') = \{a\} \cap \{a'\} \subseteq \mathsf{d}_A$$

Suppose $f : 1 \to A$ is such that $\Gamma_f \subseteq \alpha$, *i.e.* $\mathsf{d}_A(f(*), a) \subseteq \alpha(*, a) = \{a\}$. Then $A = \mathsf{d}_A(f(*), f(*)) \subseteq \{f(*)\}$, but this inclusion is contradictory with the assumption that A has more than one element.

Existential elementary doctrines. Doctrines $P : \mathcal{C}^{\text{op}} \to \mathcal{P}os$ are algebraic representations of fragment of first order predicate logic, where objects and arrows of $\mathcal{C}$ are contexts and terms and fibres $P(X)$ collect the predicates with free variables over X ordered by logical entailment. To sustain this intuition in practice the base category $\mathcal{C}$ needs finite products to model context concatenation (see also [50]). Once $\mathcal{C}$ is assumed to have finite products, an easy way to extract a relational doctrine out of P is to consider the functor $\mathsf{Rel}^P : (\mathcal{C} \times \mathcal{C})^{\text{op}} \to \mathcal{P}os$ mapping $\langle X, Y\rangle$ to $P(X \times Y)$ and $\langle f, g\rangle$ to $P_{f\times g}$. To define relational composition mimicking the standard definition one needs to restrict to those doctrines that models at least and the $(\exists, \wedge, \top, =)$-fragment of first order logic. These are called elementary existential doctrines.

A doctrine $P : \mathcal{C}^{\text{op}} \to \mathcal{P}os$ is existential elementary if all the following hold: $\mathcal{C}$ has finite products; every fibre has finite meets and these are preserved by reindexing; for every $f : X \to Y$ in $\mathcal{C}$ the reindexing P_f has a left adjoint $\mathcal{I}_f : P(X) \to P(Y)$ such that for every $\phi \in P(X)$ and every $\psi \in P(Y)$ it holds that $\mathcal{I}_f(\phi) \wedge \psi = \mathcal{I}_f(\phi \wedge P_f\psi)$ (*Frobenius reciprocity*); for every arrow $f : A \to B$ in $\mathcal{C}$ and every object X in $\mathcal{C}$ it holds that $P_f\mathcal{I}_{\pi_B} = \mathcal{I}_{\pi_A}P_{\mathsf{id}_X\times f}$, where $\pi_A : X \times A \to A$ and $\pi_B : X \times B \to B$ are projections (*Beck-Chevalley condition*).

▶ **Example 35.** An archetypal example of existential elementary doctrine is the contravariant powerset functor $\mathcal{P} : \mathcal{S}et^{\text{op}} \to \mathcal{P}os$. For a function $f : X \to Y$, the left adjoint $\mathcal{I}_f$ is the direct image mapping. Two instances are of interest. The first is when f is the diagonal $\Delta_X : X \to X \times X$. In this case the direct image evaluated on the the top element (*i.e.* the whole X) is the diagonal relation, that is $\mathcal{I}_{\Delta_X}(\top_X) = \{(x, x') \in X \times X \mid x = x'\}$. The other is when f is a projection $\pi_2 : X \times Y \to Y$. In this case $\mathcal{I}_{\pi_2}(\phi) = \{y \in Y \mid \exists_{x\in X}\,(x, y) \in \phi\}$.

The previous example shows the underling idea that, in an existential elementary doctrine, left adjoints along diagonals compute diagonal relations, lefts adjoints along projections compute existential quantifications. So every existential elementary doctrine $P : \mathcal{C}^{\text{op}} \to \mathcal{P}os$ generates a relational doctrine $\mathsf{Rel}^P : (\mathcal{C} \times \mathcal{C})^{\text{op}} \to \mathcal{P}os$ setting $\mathsf{Rel}^P(X, Y) = P(X \times Y)$ and $\mathsf{Rel}^P_{f,g} = P_{f\times g}$ and

$$\mathsf{d}_X = \mathcal{I}_{\Delta_X}(\top) \qquad \alpha\,;\beta = \mathcal{I}_{\langle\pi_1,\pi_3\rangle}(P_{\langle\pi_1,\pi_2\rangle}(\alpha) \wedge P_{\langle\pi_2,\pi_3\rangle}(\beta)) \qquad \alpha^{\perp} = P_{\langle\pi_2,\pi_1\rangle}(\alpha)$$

for $\alpha \in P(X \times Y)$ and $\beta \in P(Y \times Z)$. The assignment extends to a 2-functor $\mathsf{Rel} : \mathbf{EED} \to \mathbf{RD_s}$ where $\mathbf{EED}$ denotes the 2-category whose objects are existential elementary doctrines, 1-arrows $F : P \to Q$ are pairs $\langle \widehat{F}, \overline{F} \rangle$ where tha functor $\widehat{F} : \mathcal{C} \to \mathcal{D}$ preserves finite products and $\overline{F} : P \dot{\to} Q \circ \widehat{F}$ preserves finite meets and commutes with left adjoints.

▶ **Example 36.** Consider the powerset functor as an existential and elementary doctrine as in Example 35. It is immediate to see that $\mathsf{Rel}^{\mathcal{P}}$ is Rel.

From a relational doctrine of the form Rel^P one recovers P mapping A to $\mathsf{Rel}^P(A, 1) = P(A \times 1) \simeq P(A)$. This suggests where to look for the inverse of Rel.

First of all note that existential elementary doctrines have finite products in the base, finite meets on all fibres preserved by reindexing, while relational doctrines need not have. These structures have a neat algebraic description that uses the finite products in 2-category $\mathbf{Dtn}$: a doctrine P is based on a category with finite products, has finite meets on each fibre and these are stable under reindexing if and only if both the unique arrow $!_P$ and the diagonal Δ_P have a right adjoint in $\mathbf{Dtn}$. Since the 2-category $\mathbf{RD_s}$ of relational doctrines has finite products too, we take advantage of this characterisation and we say that a relational doctrine R is *cartesian* if the 1-arrows $!_R$ and Δ_R have right adjoints in $\mathbf{RD_s}$.

▶ **Example 37.** The doctrine $\mathsf{Rel}^{\mathcal{P}} = \mathsf{Rel} : (\mathcal{Set} \times \mathcal{Set})^{\mathrm{op}} \to \mathcal{Pos}$ is cartesian. The right adjoint to Δ_{Rel} is given using products. Indeed for $\langle \langle A, B \rangle, \langle X, Y \rangle \rangle$ the base of $\mathsf{Rel} \times \mathsf{Rel}$ the natural transformation $\mathsf{Rel}(A, B) \times \mathsf{Rel}(X, Y) \dot{\to} \mathsf{Rel}(A \times X, B \times Y)$ maps $\alpha \in \mathsf{Rel}(A, B)$ and $\beta \in \mathsf{Rel}(X, Y)$ to $\{\langle \langle a, x \rangle, \langle b, y \rangle \rangle \mid \langle a, b \rangle \in \alpha \text{ and } \langle x, y \rangle \in \beta\}$.

For a cartesian relational doctrine R denote by Doc^R the doctrine obtained by the composition of R with $\langle -, 1 \rangle : \mathcal{C}^{\mathrm{op}} \to (\mathcal{C} \times \mathcal{C})^{\mathrm{op}}$. Proposition 6 shows that Doc^R has left adjoints to all reindexing maps. One can also show that the left adjoints along projections satisfies the Beck-Chevalley condition, the only missing ingredient is the Frobenius reciprocity. We say that a relational doctrine R is *cartesian Frobenius* if it is cartesian and for every X and Y in $\mathcal{C}$ and every $\alpha \in R(X, Y)$ it holds

$$\Gamma^{\perp}_{\Delta_X} \,;\Gamma_{\Delta_X} = \Gamma_{\Delta_X \times \mathsf{id}_X} \,;\Gamma^{\perp}_{\mathsf{id}_X \times \Delta_X}$$

$$\Gamma_{\Delta_X \times \mathsf{id}_Y} \,;(\mathsf{d}_X \bullet \alpha \bullet \mathsf{d}_Y) \,;\Gamma^{\perp}_{\mathsf{id}_X \times \Delta_Y} = (\Gamma_{\Delta_X \times \mathsf{id}_Y} \,;(\mathsf{d}_X \bullet \alpha \bullet \mathsf{d}_Y) \,;\Gamma^{\perp}_{\mathsf{id}_X \times \Delta_Y})^{\perp}$$

where the first condition is inspired by [13]. In general relational doctrines need not be cartesian Frobenius as shown by the following example.

▶ **Example 38.** Consider the relational doctrine $\mathbb{R}_{\geqslant 0}\text{-}\mathsf{Rel} : (\mathcal{Set} \times \mathcal{Set})^{\mathrm{op}} \to \mathcal{Pos}$ of metric relations, where $\mathbb{R}_{\geqslant 0} = \langle [0, \infty], \geqslant, +, 0 \rangle$ is the Lawvere's quantale as in Example 3. This doctrine is not cartesian. Indeed to be cartesian would imply the existence of a 1-arrow $\times : \mathbb{R}_{\geqslant 0}\text{-}\mathsf{Rel} \times \mathbb{R}_{\geqslant 0}\text{-}\mathsf{Rel} \to \mathbb{R}_{\geqslant 0}\text{-}\mathsf{Rel}$ which is right adjoint to $\Delta_{\mathbb{R}_{\geqslant 0}\text{-}\mathsf{Rel}}$. The right adjoint should be a pair $\langle \widehat{\times}, \overline{\times} \rangle$ where $\overline{\times}$ commutes with relational composition, that is it satisfies equations of the form $(\alpha \overline{\times} \beta) \,;(\alpha' \overline{\times} \beta') = (\alpha \,; \alpha') \overline{\times} (\beta \,; \beta')$. For $\alpha \in \mathbb{R}_{\geqslant 0}\text{-}\mathsf{Rel}(A, B)$ and $\beta \in \mathbb{R}_{\geqslant 0}\text{-}\mathsf{Rel}(X, Y)$ the relation $\alpha \overline{\times} \beta$ is computed as follows

$$(\alpha \overline{\times} \beta)(a, x, b, y) = \sup \{\alpha(a, b), \beta(x, y)\}$$

Suppose α and β are constant functions, and take two other constant functions for $\alpha' \in \mathbb{R}_{\geqslant 0}\text{-}\mathsf{Rel}(B, C)$ and $\beta' \in \mathbb{R}_{\geqslant 0}\text{-}\mathsf{Rel}(Y, Z)$. The equation $(\alpha \overline{\times} \beta) \,;(\alpha' \overline{\times} \beta') = (\alpha \,; \alpha') \overline{\times} (\beta \,; \beta')$ reduces to $\sup \{\alpha, \beta\} + \sup \{\alpha', \beta'\} = \sup \{\alpha + \alpha', \beta + \beta'\}$ that need not hold (take $\alpha = \beta' = 0$ and $\alpha' = \beta = 1$).

Relational doctrines of the form Rel^P are cartesian Frobenius, therefore the essential image of Rel is **FRD**, the 2-full 2-subcategory of $\mathbf{RD_s}$ on cartesian Frobenius relational doctrines and 1-arrows that preserve the cartesian structure. Moreover doctrines of the form Doc^R are existential elementary if and only if R is cartesian Frobenius. This determines a 2-functor $\mathrm{Doc} : \mathbf{FRD} \to \mathbf{EED}$ that, together with $\mathsf{Rel} : \mathbf{EED} \to \mathbf{FRD}$, determine the equivalence stated by the following theorem.

▶ **Theorem 39.** *The 2-categories* **FRD** *and* $\mathbf{ERD}_{(\textsc{ruc})}$ *are 2-equivalent.*

Relying on the equivalence proved in Theorem 39, the completion of an elementary existential doctrine with quotients introduced in [42] is equivalent to the relational quotient completion of a cartesian Frobenius relational doctrine. The elementary quotient completion of an existential elementary doctrine introduced in [41] is equivalent to the extensional collapse of the relational quotient completion of cartesian Frobenius relational doctrines. This results in a wide range of examples of relational doctrines such as realisability doctrines, doctrines of (strong/weak) subobjects and syntactic doctrines [31, 50, 58]. Also dependent Types Theories give rise to existential elementary doctrines whose elementary quotient completion is the category of setoids [41, 42].

We proved that existential elementary doctrines and cartesian Frobenius relational doctrines are equivalent, and the completions introduced in this paper coincide with the corresponding ones introduced by Maietti and Rosolini. Both of them work on larger classes of doctrines. More specifically, the completions proposed by Maietti and Rosolini can be applied to doctrines that need not be existential in the sense that they need not have left adjoints to all the reindexing maps, but they need finite products in the base and finite meets in the fibres. On the other hand relational doctrines intrinsically have left adjoints to all reindexing maps, but the completions described in this paper work also on relational doctrines that need not be cartesian and need not have a base with finite products. This is a crucial ingredient to cover the quantitative examples.

6 Conclusions

We introduced relational doctrines as a functorial description of the essence of the calculus of relations. Relying on this structure, we defined quotients and a universal construction adding them capable to cover quantitative settings as well. Then, we studied extensional equality in relational doctrines, showing it captures various notion of separation in metric and topological structures. Moreover, we described a universal construction forcing extensionality, thus separation, analysing how it interacts with quotients. Finally, we compared relational doctrines with two important classes of examples: ordered categories with involution, proving these correspond to relational doctrines having both extensional equality and the rule of unique choice, and existential elementary doctrines, showing they correspond to cartesian relational doctrines satisfying suitable Frobenius rules.

There are many directions for future work. The first one is the study of choice rules in the framework of relational doctrines, extending known results for doctrines [40, 44], giving them a quantitative interpretation, for instance in terms of completeness, following the connection between (ruc) and Cauchy completeness pointed out in [39]. Moreover, this could lead us to the definition of a quantitative counterpart of the tripos-to-topos construction, generalising known results [24, 43], which could generate categories of complete (partial) metric spaces.

We also plan to bring the study of relations to the proof-relevant setting of type theories. Algebraically this can be done moving from doctrines to arbitrary fibrations as it is common practice. On the syntactic side, instead, things are much less clear: developing a proper syntax and rules for a "relational type theory" is something interesting per se. Actually, we do not even have a syntactic calculus behind relational doctrines. Then, another interesting direction is to design it, possibly in a diagrammatic way, for instance in the style of string diagrams.

Another interesting direction is to study relational doctrines with tools coming from the theory of double categories. Indeed, from Remark 2 one can easily read a relational doctrine as a special double category and equivalence relations looks similar to monads in such a double category [23]. Thus, it would be interesting applying general results for monads in double categories to this specific case, possibly deriving properties and constructions related to equivalences and quotients.

Finally, a promising direction would be the use of relational doctrines as an abstract framework where to formulate and develop relational techniques used in the study of programming languages and software systems, such as (coalgebraic) bisimulation, program equivalence or operational semantics, as well as, quantitative equational theories and rewriting.

References

1 Jirí Adámek. Varieties of quantitative algebras and their monads. In Christel Baier and Dana Fisman, editors, *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2022*, pages 9:1–9:10. ACM, 2022. doi:10.1145/3531130.3532405.

2 Jiří Adámek, Matěj Dostál, and Jiří Velebil. Quantitative algebras and a classification of metric monads, 2023. arXiv:2210.01565.

3 Hajnal Andréka, Steven Givant, Peter Jipsen, and Istvan Németi. On tarski's axiomatic foundations of the calculus of relations. *The Journal of Symbolic Logic*, 82(3):966–994, 2017.

4 Giorgio Bacci, Radu Mardare, Prakash Panangaden, and Gordon D. Plotkin. An algebraic theory of markov processes. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, pages 679–688. ACM, 2018. doi:10.1145/3209108.3209177.

5 Christel Baier and Mila E. Majster-Cederbaum. Denotational semantics in the CPO and metric approach. *Theoretical Computer Science*, 135(2):171–220, 1994. doi:10.1016/0304-3975(94)00046-8.

6 Michael Barr. Relational algebras. In S. MacLane, H. Applegate, M. Barr, B. Day, E. Dubuc, Phreilambud, A. Pultr, R. Street, M. Tierney, and S. Swierczkowski, editors, *Reports of the Midwest Category Seminar IV*, pages 39–55, Berlin, Heidelberg, 1970. Springer Berlin Heidelberg.

7 Gilles Barthe, Venanzio Capretta, and Olivier Pons. Setoids in type theory. *Journal of Functional Programming*, 13(2):261–293, 2003. doi:10.1017/S0956796802004501.

8 Gilles Barthe, Marco Gaboardi, Justin Hsu, and Benjamin C. Pierce. Programming language techniques for differential privacy. *ACM SIGLOG News*, 3(1):34–53, 2016. doi:10.1145/2893582.2893591.

9 Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. *ACM Transactions on Programming Languages and Systems*, 35(3):9:1–9:49, 2013. doi:10.1145/2492061.

10 Renato Betti and John Power. On local adjointness of distributive bicategories. *Bollettino della Unione Matematica Italiana*, 2(4):931–947, 1988.

11 E. Bishop. *Foundations of Constructive Analysis.* McGraw-Hill series in higher mathematics. McGraw-Hill, 1967.

12 E. Bishop and D. Bridges. *Constructive Analysis.* Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg, 2012.

13 Filippo Bonchi, Alessio Santamaria, Jens Seeber, and Pawel Sobocinski. On doctrines and cartesian bicategories. In Fabio Gadducci and Alexandra Silva, editors, *9th Conference on Algebra and Coalgebra in Computer Science, CALCO 2021*, volume 211 of *LIPIcs*, pages 10:1–10:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CALCO.2021.10.

14 A. Carboni and R. Celia Magno. The free exact category on a left exact one. *Journal of the Australian Mathematical Society. Series A. Pure Mathematics and Statistics*, 33:295–301, 1982.

15 Aurelio Carboni and E.M. Vitale. Regular and exact completions. *Journal of Pure and Applied Algebra*, 125:79–117, 1998.

16 Aurelio Carboni and Rober Walters. Cartesian bicategories I. *Journal of Pure and Applied Algebra*, 49(1-2):11–32, 1987.

17 Konstantinos Chatzikokolakis, Daniel Gebler, Catuscia Palamidessi, and Lili Xu. Generalized bisimulation metrics. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, volume 8704 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2014. doi:10.1007/978-3-662-44584-6_4.

18 Francesco Dagnino and Fabio Pasquali. Logical foundations of quantitative equality. In Christel Baier and Dana Fisman, editors, *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2022*, pages 16:1–16:13. ACM, 2022. doi:10.1145/3531130.3533337.

19 Ugo Dal Lago and Francesco Gavazzo. A relational theory of effects and coeffects. *Proceedings of the ACM on Programming Languages*, 6(POPL):1–28, 2022. doi:10.1145/3498692.

20 Ugo Dal Lago, Francesco Gavazzo, and Akira Yoshimizu. Differential logical relations, part I: the simply-typed case. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, volume 132 of *LIPIcs*, pages 111:1–111:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.111.

21 Arthur Azevedo de Amorim, Marco Gaboardi, Justin Hsu, Shin-ya Katsumata, and Ikram Cherigui. A semantic account of metric preservation. 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 545–556. ACM, 2017. doi:10.1145/3009837.3009890.

22 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled markov processes. *Theoretical Computer Science*, 318(3):323–354, 2004. doi:10.1016/j.tcs.2003.09.013.

23 Thomas M. Fiore, Nicola Gambino, and Joachim Kock. Monads in double categories. *Journal of Pure and Applied Algebra*, 215(6):1174–1197, 2011. doi:10.1016/j.jpaa.2010.08.003.

24 Jonas Frey. Triposes, q-toposes and toposes. *Annals of Pure and Applied Logic*, 166(2):232–259, 2015. doi:10.1016/j.apal.2014.10.005.

25 Peter J. Freyd and Andre Scedrov. *Categories, allegories*, volume 39 of *North-Holland mathematical library*. North-Holland, 1990.

26 Francesco Gavazzo. Quantitative behavioural reasoning for higher-order effectful programs: Applicative distances. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, pages 452–461. ACM, 2018. doi:10.1145/3209108.3209149.

27 Francesco Gavazzo and Cecilia Di Florio. Elements of quantitative rewriting. *Proceedings of the ACM on Programming Languages*, 7(POPL):1832–1863, 2023. doi:10.1145/3571256.

28 Steven Givant. The calculus of relations as a foundation for mathematics. *Journal of Automated Reasoning*, 37(4):277–322, 2006. doi:10.1007/s10817-006-9062-x.

29 Dirk Hofmann, Gavin J Seal, and Walter Tholen. *Monoidal Topology: A Categorical Approach to Order, Metric, and Topology*, volume 153. Cambridge University Press, 2014.

30 Martin. Hofmann. *Extensional constructs in intensional type theory*. CPHC/BCS Distinguished Dissertations. Springer-Verlag London Ltd., London, 1997.

31 Bart P. F. Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in logic and the foundations of mathematics*. North-Holland, 2001. URL: http://www.elsevierdirect.com/product.jsp?isbn=9780444508539.

32 Alexander Kurz and Jiri Velebil. Relation lifting, a survey. *J. Log. Algebraic Methods Program.*, 85(4):475–499, 2016. doi:10.1016/j.jlamp.2015.08.002.

33 Stephen Lack. *A 2-Categories Companion*, pages 105–191. Springer New York, New York, NY, 2010. doi:10.1007/978-1-4419-1524-5_4.

34 Jim Laird, Giulio Manzonetto, Guy McCusker, and Michele Pagani. Weighted relational models of typed lambda-calculi. In *Proceedings of the 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013*, pages 301–310. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.36.

35 Joachim Lambek. Diagram chasing in ordered categories with involution. *Journal of Pure and Applied Algebra*, 143(1):293–307, 1999. doi:10.1016/S0022-4049(98)00115-7.

36 F. Wiliam Lawvere. Adjointness in foundations. *Dialectica*, 23:281–296, 1969. also available as Repr. Theory Appl. Categ., 16 (2006) 1–16. doi:10.1111/j.1746-8361.1969.tb01194.x.

37 F. William Lawvere. Equality in hyperdoctrines and comprehension schema as an adjoint functor. In A. Heller, editor, *Proceedings of the New York Symposium on Application of Categorical Algebra*, pages 1–14. American Mathematical Society, 1970.

38 F. William Lawvere. Metric spaces, generalized logic, and closed categories. *Rend. Sem. Mat. Fis. Milano*, 43:135–166, 1973.

39 F.W. Lawvere. Metric spaces, generalized logic, and closed categories. *Rend. Sem. Mat. Fis. Milano*, 43:135–166, 1973.

40 Maria Emilia Maietti, Fabio Pasquali, and Giuseppe Rosolini. Triposes, exact completions, and Hilbert's epsilon-operator. *Tbilisi Mathematical Journal*, 10(3):141–166, 2017. doi:10.1515/tmj-2017-0106.

41 Maria Emilia Maietti and Giuseppe Rosolini. Elementary quotient completion. *Theory Appl. Categ.*, 27(17):445–463, 2013.

42 Maria Emilia Maietti and Giuseppe Rosolini. Quotient completion for the foundation of constructive mathematics. *Logica Universalis*, 7(3):371–402, 2013. doi:10.1007/s11787-013-0080-2.

43 Maria Emilia Maietti and Giuseppe Rosolini. Unifying exact completions. *Appl. Categorical Struct.*, 23(1):43–52, 2015. doi:10.1007/s10485-013-9360-5.

44 M.E. Maietti and G. Rosolini. Relating quotient completions via categorical logic. In Dieter Probst and Peter Schuster (eds.), editors, *Concepts of Proof in Mathematics, Philosophy, and Computer Science*, pages 229–250. De Gruyter, 2016.

45 Radu Mardare, Prakash Panangaden, and Gordon D. Plotkin. Quantitative algebraic reasoning. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2016*, pages 700–709. ACM, 2016. doi:10.1145/2933575.2934518.

46 Radu Mardare, Prakash Panangaden, and Gordon D. Plotkin. On the axiomatizability of quantitative algebras. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005102.

47 Radu Mardare, Prakash Panangaden, and Gordon D. Plotkin. Fixed-points for quantitative equational logics. In *Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470662.

48 C.-H. Luke Ong. Quantitative semantics of the lambda calculus: Some generalisations of the relational model. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005064.

49 Charles S. Peirce. The logic of relatives. *The Monist*, 7(2):161–217, 1897.

50 Andrew M. Pitts. Categorical logic. In *Handbook of logic in computer science, Vol. 5*, volume 5 of *Handbook of Logic in Computer Science*, pages 39–128. Oxford Univ. Press, New York, 2000.

51 Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: a calculus for differential privacy. In Paul Hudak and Stephanie Weirich, editors, *Proceedings of the 15th ACM International Conference on Functional programming, ICFP 2010*, pages 157–168. ACM, 2010. doi:10.1145/1863543.1863568.

52 D.S. Scott. A new category? Domains, spaces and equivalence relations. Available at http://www.cs.cmu.edu/Groups/LTC/, 1996.

53 R. Street. The formal theory of monads. *Journal of Pure and Applied Algebra*, 2(2):149–168, 1972. doi:10.1016/0022-4049(72)90019-9.

54 A. Tarski and S.R. Givant. *A Formalization of Set Theory Without Variables*. Number v. 41 in A formalization of set theory without variables. American Mathematical Soc., 1988.

55 Alfred Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6(3):73–89, 1941. doi:10.2307/2268577.

56 Albert Thijs. *Simulations and fixpoint semantics*. PhD thesis, Rijksuniversiteit Groningen, 1996.

57 Franck van Breugel and James Worrell. A behavioural pseudometric for probabilistic transition systems. *Theoretical Computer Science*, 331(1):115–142, 2005. doi:10.1016/j.tcs.2004.09.035.

58 Jaap van Oosten. *Realizability: An Introduction to its Categorical Side*, volume 152 of *Studies in Logic and the Foundations of Mathematics*. North Holland Publishing Company, 2008.

59 Lili Xu, Konstantinos Chatzikokolakis, and Huimin Lin. Metrics for differential privacy in concurrent systems. In Erika Ábrahám and Catuscia Palamidessi, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 34th IFIP WG 6.1 International Conference, FORTE 2014, Held as Part of the 9th International Federated Conference on Distributed Computing Techniques, DisCoTec 2014, Berlin, Germany, June 3-5, 2014. Proceedings*, volume 8461 of *Lecture Notes in Computer Science*, pages 199–215. Springer, 2014. doi:10.1007/978-3-662-43613-4_13.

Type Isomorphisms for Multiplicative-Additive Linear Logic

Rémi Di Guardia
Univ Lyon, EnsL, UCBL, CNRS, LIP, F-69342, LYON Cedex 07, France

Olivier Laurent
Univ Lyon, EnsL, UCBL, CNRS, LIP, F-69342, LYON Cedex 07, France

Abstract

We characterize type isomorphisms in the multiplicative-additive fragment of linear logic (MALL), and thus for $\star$-autonomous categories with finite products, extending a result for the multiplicative fragment by Balat and Di Cosmo [2]. This yields a much richer equational theory involving distributivity and annihilation laws. The unit-free case is obtained by relying on the proof-net syntax introduced by Hughes and Van Glabbeek [10]. We then use the sequent calculus to extend our results to full MALL (including all units).

2012 ACM Subject Classification Theory of computation → Linear logic

Keywords and phrases Linear Logic, Type Isomorphisms, Multiplicative-Additive fragment, Proof nets, Sequent calculus, Star-autonomous categories with finite products

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.26

Related Version *Full version with proofs*: `https://hal.science/hal-04082204`

Funding This work was performed within the framework of the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007), and supported by the projects DyVerSe (ANR-19-CE48-0010), QuaReMe (ANR-20-CE48-0005) and ReCiProg (ANR-21-CE48-0019), all operated by the French National Research Agency (ANR). This work was also supported by the IRN Linear Logic.

1 Introduction

The question of type isomorphisms consists in trying to understand when two types in a type system (or two formulas in a logic) are "the same". The general question can be described in category theory: two objects A and B are isomorphic ($A \simeq B$) if there exist morphisms $A \xrightarrow{f} B$ and $B \xrightarrow{g} A$ such that $f \circ g = \mathrm{id}_B$ and $g \circ f = \mathrm{id}_A$. f and g are the underlying isomorphisms. Given a (class of) category, the question is then to find equations characterizing when two objects A and B are isomorphic (in all instances of the class). The focus here is on pairs of isomorphic objects rather than on the isomorphisms themselves. For example, in the class of cartesian categories, one finds the following isomorphic objects: $A \times B \simeq B \times A$, $(A \times B) \times C \simeq A \times (B \times C)$ and $A \times \top \simeq A$. Regarding type systems and logics, one can instantiate the categorical notion. For instance in typed λ-calculi: two types A and B are isomorphic if there exist two λ-terms $M : A \to B$ and $N : B \to A$ such that $\lambda x : B.(M\ (N\ x)) =_{\beta\eta} \lambda x : B.x$ and $\lambda x : A.(N\ (M\ x)) =_{\beta\eta} \lambda x : A.x$ where $=_{\beta\eta}$ is $\beta\eta$-equality. This corresponds to isomorphic objects in the syntactic category generated by terms up to $=_{\beta\eta}$. Similarly, type isomorphisms can also be considered in logic, following what happens in the λ-calculus through the Curry-Howard correspondence: simply replace λ-terms with proofs, types with formulas, β-reduction with cut-elimination and η-expansion with axiom-expansion. In this way, type isomorphisms are studied in a wide range of theories, such as category theory [16], λ-calculus [4] and proof theory [2]. They may be used to develop practical tools, such as search in a library of a functional programming language [14].

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 26; pp. 26:1–26:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Following the definition, it is usually easy to prove that the type-isomorphism relation is a congruence. It is then natural to look for an equational theory generating this congruence. Testing whether or not two types are isomorphic is then much easier. An equational theory $\mathcal{T}$ is called *sound* with respect to type isomorphisms if types equal up to $\mathcal{T}$ are isomorphic. It is called *complete* if it equates any pair of isomorphic types. Given a (class of) category, a type system or a logic, our goal is to find an associated sound and complete equational theory for type isomorphisms. This is not always possible as the induced theory may not be finitely axiomatisable (see for instance [6]).

Soundness is usually the easy direction as it is sufficient to exhibit pairs of terms corresponding to each equation. The completeness part is often harder, and there are in the literature two main approaches to solve this problem. The first is a semantic method, relying on the fact that if two types are isomorphic then they are isomorphic in all (denotational) models. One thus looks for a model in which isomorphisms can be computed (more easily than in the syntactic model) and are all included in the equational theory under consideration (this is the approach used in [16, 12] for example). Finding such a model simple enough for its isomorphisms to be computed, but still complex enough not to contain isomorphisms absent in the syntax is the difficulty. The second method is the syntactic one, which consists in studying isomorphisms directly in the syntax. The analysis of pairs of terms composing to the identity should provide information on their structure and then on their type so as to deduce the completeness of the equational theory (see for example [4, 2]). The easier the equality ($=_{\beta\eta}$ for example) between proof objects can be computed, the easier the analysis of isomorphisms will be.

We place ourselves in the framework of linear logic (LL) [7], the underlying question being "is there an equational theory corresponding to the isomorphisms between formulas in this logic?". LL is a very rich logic containing three classes of propositional connectives: multiplicative, additive and exponential ones. The multiplicative and additive families provide two copies of each classical propositional connective: two copies of conjunction ($\otimes$ and $\&$), of disjunction ($\parr$ and $\oplus$), of true (1 and $\top$) and of false ($\bot$ and 0). The exponential family is constituted of two modalities ! and ? bridging the gap between multiplicatives and additives through four isomorphisms $!(A \mathbin{\&} B) \simeq {!A} \otimes {!B}$, $?(A \oplus B) \simeq {?A} \parr {?B}$, $!\top \simeq 1$ and $?0 \simeq \bot$. In the multiplicative fragment (MLL) of LL (using only $\otimes$, $\parr$, 1 and $\bot$, and corresponding to $\star$-autonomous categories), the question of type isomorphisms was answered positively using a syntactic method based on proof-nets by Balat and Di Cosmo [2]: isomorphisms emerge from associativity and commutativity of the multiplicative connectives $\otimes$ and $\parr$, as well as neutrality of the multiplicative units 1 and $\bot$. The question was also solved for the polarized fragment of LL by one of the authors using game semantics [12]. It is conjectured that isomorphisms in full LL correspond to those in its polarized fragment (Table 1 together with the four exponential equations above). As a step towards solving this conjecture, we prove the type isomorphisms in the multiplicative-additive fragment (MALL) of LL are generated by the equational theory of Table 1 (and this applies at the same time to the class of $\star$-autonomous categories with finite products).

This situation is much richer than in the multiplicative fragment since isomorphisms include not only associativity, commutativity and neutrality, but also the distributivity of the multiplicative connective $\otimes$ (resp. $\parr$) over the additive $\oplus$ (resp. $\&$) as well as the associated annihilation laws for the additive unit 0 (resp. $\top$) over the multiplicative connective $\otimes$ (resp. $\parr$). Using a semantic approach looks difficult as most of the known models of MALL immediately come with unwanted isomorphisms not valid in the syntax: $\top \otimes A \simeq \top$ in coherent spaces for example [7]. For this reason we use a syntactic method. We follow the

Table 1 Type isomorphisms in MALL.

Commutativity	$A \otimes B = B \otimes A$	$A \parr B = B \parr A$	$A \oplus B = B \oplus A$	$A \,\&\, B = B \,\&\, A$
Associativity	$A \otimes (B \otimes C) = (A \otimes B) \otimes C$		$A \parr (B \parr C) = (A \parr B) \parr C$	
	$A \oplus (B \oplus C) = (A \oplus B) \oplus C$		$A \,\&\, (B \,\&\, C) = (A \,\&\, B) \,\&\, C$	
Distributivity	$A \otimes (B \oplus C) = (A \otimes B) \oplus (A \otimes C)$		$A \parr (B \,\&\, C) = (A \parr B) \,\&\, (A \parr C)$	
Neutrality	$A \otimes 1 = A$	$A \parr \bot = A$	$A \oplus 0 = A$	$A \,\&\, \top = A$
Annihilation	$A \otimes 0 = 0$	$A \parr \top = \top$		

approach by Balat and Di Cosmo [2] based on proof-nets. Indeed, proof-nets provide a very good syntax for linear logic where studying composition of proofs by cut, cut-elimination and identity of proofs is very natural. However, already in [2] some trick had to be used to deal with units as proof-nets are working perfectly only in the unit-free multiplicative fragment of linear logic. If one puts units aside, there is a notion of proof-nets incorporating both multiplicative and additive connectives in such a way that cut-free proofs are represented in a *canonical* way, and cut-elimination can be dealt with in a parallel manner. This is the syntax of proof-nets introduced by Hughes & Van Glabbeek in [10].

Our proof of the completeness of the equational theory of Table 1 goes in two steps. First we adapt, in Section 3, the proof of Balat & Di Cosmo [2] to the setting of Hughes & Van Glabbeek's proof-nets [10]. This requires a precise analysis of the structure of proof-nets because of the richer structure induced by the presence of the additive connectives. The situation is much more complex than in the multiplicative setting since for example subformulas can be duplicated through distributivity equations, breaking a linearity property crucial in [2]. Once this is solved, it remains to add units (Section 4). By lack of a good-enough notion of proof-nets for MALL including units, we go back to the sequent calculus to deal with units on top of the results obtained for the unit-free fragment. This goes through a characterization of the equality of proofs up to cut-elimination and axiom-expansion by means of rule commutations. A result which is not surprising, but never proved before for MALL as far as we know, and rather tedious to settle. Using it, we analyse the behaviour of units inside isomorphisms to conclude that they can be replaced with fresh atoms, once formulas are simplified appropriately. We can conclude by means of the unit-free case. Finally, seeing MALL as a category, we extend our result to conclude that Table 1 (together with $A \multimap B \simeq A^\perp \parr B$, De Morgan's laws and involutivity of negation) provides the equational theory of isomorphisms valid in all $\star$-autonomous categories with finite products (Section 5).

2 Definitions and preliminary results

2.1 Multiplicative-Additive Linear Logic

The multiplicative-additive fragment of linear logic [7], denoted by MALL, has formulas given by the following grammar, where X belongs to a given enumerable set of *atoms*:

$$A, B \;:=\; X \mid X^\perp \mid A \otimes B \mid A \parr B \mid 1 \mid \bot \mid A \,\&\, B \mid A \oplus B \mid \top \mid 0$$

Orthogonality $(\cdot)^\perp$ expands into an involution on arbitrary formulas through $X^{\perp\perp} = X$ on an atom X, $1^\perp = \bot$, $\bot^\perp = 1$, $\top^\perp = 0$, $0^\perp = \top$ and De Morgan's laws $(A \otimes B)^\perp = B^\perp \parr A^\perp$, $(A \parr B)^\perp = B^\perp \otimes A^\perp$, $(A \,\&\, B)^\perp = B^\perp \oplus A^\perp$, $(A \oplus B)^\perp = B^\perp \,\&\, A^\perp$. The *non-commutative* De Morgan's laws are the good notion of duality, as shown in the context of cyclic linear logic where this leads to planar proof-nets [1]. This choice in our setting will often result in planar graphs on our illustrations, with axiom links not crossing each others.

Sequents are lists of formulas of the form $\vdash A_1, \ldots, A_n$. Sequent calculus rules are:[1]

$$\frac{}{\vdash A^\perp, A}\,ax \qquad \frac{\vdash \Gamma}{\vdash \sigma(\Gamma)}\,ex \qquad \frac{\vdash A, \Gamma \qquad \vdash A^\perp, \Delta}{\vdash \Gamma, \Delta}\,cut$$

$$\frac{\vdash A, \Gamma \qquad \vdash B, \Delta}{\vdash A \otimes B, \Gamma, \Delta}\,\otimes \qquad \frac{\vdash A, B, \Gamma}{\vdash A \parr B, \Gamma}\,\parr \qquad \frac{}{\vdash 1}\,1 \qquad \frac{\vdash \Gamma}{\vdash \bot, \Gamma}\,\bot$$

$$\frac{\vdash A, \Gamma \qquad \vdash B, \Gamma}{\vdash A \mathbin{\&} B, \Gamma}\,\& \qquad \frac{\vdash A, \Gamma}{\vdash A \oplus B, \Gamma}\,\oplus_1 \qquad \frac{\vdash B, \Gamma}{\vdash A \oplus B, \Gamma}\,\oplus_2 \qquad \frac{}{\vdash \top, \Gamma}\,\top$$

In practice we consider exchange rules as incorporated in the conclusion of the rule above, thus dealing with rules like: $\dfrac{\vdash A, B, \Gamma, \Delta}{\vdash \Gamma, A \parr B, \Delta}\,\parr$. In this spirit, when we write $\dfrac{\vdash \Gamma, A, B, \Delta}{\vdash \Gamma, A \parr B, \Delta}\,\parr$ we mean that the appropriate permutation is also incorporated in the rule above.

The main difference with the multiplicative fragment of linear logic (MLL) is the &-rule, which introduces some sharing of the context Γ. From this comes the notion of a *slice* [7, 8] which is a partial proof missing some additive component. Slices are obtained by using the same rules as proofs except for the &-rule which is replaced by its two sliced versions:

$$\frac{\vdash A, \Gamma}{\vdash A \mathbin{\&} B, \Gamma}\,\&_1 \qquad\qquad \frac{\vdash B, \Gamma}{\vdash A \mathbin{\&} B, \Gamma}\,\&_2$$

By *unit-free* MALL, we mean the restriction of MALL to formulas not involving the units 1, $\bot$, $\top$ and 0, and as such without the 1, $\bot$ and $\top$-rules. When speaking of a *positive* formula, we mean a formula with main connective $\otimes$ or $\oplus$, a unit 1 or 0, or an atom X. A *negative* formula is one with main connective $\parr$ or &, a unit $\bot$ or $\top$, or a negated atom $X^\perp$.

2.2 Linear isomorphisms

▶ **Definition 1** (Isomorphism). *Two formulas A and B are* isomorphic*, denoted $A \simeq B$, if there exist proofs π of $\vdash A^\perp, B$ and π' of $\vdash B^\perp, A$ whose composition by cut over B (resp. A) is equal to the axiom on $\vdash A^\perp, A$ (resp. $\vdash B^\perp, B$) up to axiom-expansion and cut-elimination. (Axiom-expansion and cut-elimination for MALL are recalled in Appendix A.)*

Because of the analogy with the λ-calculus and since there will be no ambiguity, we use the notation $=_{\beta\eta}$ for equality of proofs up to cut-elimination (β) and axiom-expansion (η). Similarly, $=_\beta$ is equality up to cut-elimination only. We use the notations $\pi \overset{B}{\bowtie} \pi'$ for the proof obtained by adding a cut on B between π and π', and $A \overset{\pi,\pi'}{\simeq} B$ when π and π' define an isomorphism between A and B, that is when $\pi \overset{B}{\bowtie} \pi' =_{\beta\eta} \mathrm{id}_A$ and $\pi \overset{A}{\bowtie} \pi' =_{\beta\eta} \mathrm{id}_B$ (where id_A is the axiom-expansion of the proof of $\vdash A^\perp, A$ containing just an axiom rule).

We aim to prove that two MALL (resp. unit-free MALL) formulas are isomorphic if and only if they are equal in the equational theory $\mathcal{E}$ (resp. $\mathcal{E}^\dagger$) defined as follows.

▶ **Definition 2** (Equational theories). *We denote by $\mathcal{E}$ the equational theory given on Table 1 on Page 3, while $\mathcal{E}^\dagger$ denotes the part not involving units,* i.e. *with commutativity, associativity and distributivity only.*

Given an equational theory $\mathcal{T}$, the notation $A =_{\mathcal{T}} B$ means that formulas A and B are equal in the theory $\mathcal{T}$. As often, the soundness part is easy (but tedious) to prove.

▶ **Theorem 3** (Isomorphisms soundness, see Lemma 3 in [12]). *If $A =_{\mathcal{E}} B$ then $A \simeq B$.*

[1] With A and B arbitrary formulas, Γ and Δ contexts (*i.e.* lists of formulas) and σ a permutation.

All the difficulty lies in the proof of the other implication, completeness, on which the rest of this work focuses.

2.3 Axiom-expansion

A first simplification is that we can reduce the definition of isomorphisms to proofs with expanded axioms only, no more using the η relation. Given an MALL proof π, we denote by $\eta(\pi)$ the η-normal form of π, *i.e.* the proof obtained by expanding iteratively all *ax*-rules in π (axiom-expansion is confluent and strongly normalizing).

▶ **Proposition 4** (Reduction to axiom-expanded proofs). *Let π and ϖ be MALL proofs such that $\pi =_{\beta\eta} \varpi$. Then $\eta(\pi) =_{\beta} \eta(\varpi)$ with, in this sequence, only proofs in η-normal form.*

Thus, we will from now on consider only proofs with expanded axioms, manipulated through composition by cut and cut-elimination. To prove completeness, we start with the unit-free case by using a syntactic approach based on the proof-nets from Hughes & Van Glabbeek [10], which are a more canonical representation of proofs [11].

2.4 Proof-nets for unit-free MALL

We use the definition of unit-free MALL proof-net from [10]. Other definitions exist, see the original one from Girard [8], or others such as [5, 9]. Still, the definition we take is one of the most satisfactory, from the point of view of canonicity and cut-elimination for instance (see [10, 11], or the introduction of [9] for a comparison of alternative definitions). We recall here quickly this definition of proof-nets. Please refer to [10] for more details.

A sequent is seen as its syntactic forest with as internal vertices its connectives and as leaves the atoms of its formulas. We always identify a formula A with its syntactic tree $T(A)$. A *cut pair* is a formula $A * A^{\perp}$, for a formula A; the connective $*$ is unordered. A *cut sequent* $[\Sigma]\ \Gamma$ is composed of a list Σ of cut pairs and a sequent Γ. When $\Sigma = \emptyset$ is empty, we denote it simply by Γ. When we write a $\parr\backslash\&$-vertex, we mean a $\parr$- or $\&$-vertex (a *negative* vertex); similarly a $\otimes\backslash\oplus$-vertex is a $\otimes$- or $\oplus$-vertex (a *positive* vertex). An *additive resolution* of a cut sequent $[\Sigma]\ \Gamma$ is any result of deleting zero or more cut pairs from Σ and one argument subtree of each additive connective ($\&$ or $\oplus$) of $\Sigma \cup \Gamma$. A *$\&$-resolution* of a cut sequent $[\Sigma]\ \Gamma$ is any result of deleting one argument subtree of each $\&$-connective of $\Sigma \cup \Gamma$.

An *(axiom) link* on $[\Sigma]\ \Gamma$ is an unordered pair of complementary leaves in $\Sigma \cup \Gamma$ (labeled with X and $X^{\perp}$). A *linking* λ on $[\Sigma]\ \Gamma$ is a set of links on $[\Sigma]\ \Gamma$ such that the sets of the leaves of its links form a partition of the set of leaves of an additive resolution of $[\Sigma]\ \Gamma$, additive resolution which is denoted $[\Sigma]\ \Gamma \restriction \lambda$.

A set of linkings Λ on $[\Sigma]\ \Gamma$ *toggles* a $\&$-vertex W if both arguments (called *premises*) of W are in $[\Sigma]\ \Gamma \restriction \Lambda \coloneqq \bigcup_{\lambda \in \Lambda} [\Sigma]\ \Gamma \restriction \lambda$. We say a link a *depends* on a $\&$-vertex W in Λ if there exist $\lambda, \lambda' \in \Lambda$ such that $a \in \lambda \backslash \lambda'$ and W is the only $\&$-vertex toggled by $\{\lambda; \lambda'\}$. The graph $\mathcal{G}_{\Lambda}$ is defined as $[\Sigma]\ \Gamma \restriction \Lambda$ with the edges from $\cup\Lambda$ and enriched with jump edges $l \to W$ for each leaf l and each $\&$-vertex W such that there exists $a \in \lambda \in \Lambda$, between l and some l', with a depending on W in Λ. When $\Lambda = \{\lambda\}$ is composed of a single linking, we shall simply denote $\mathcal{G}_{\lambda} = \mathcal{G}_{\{\lambda\}}$ (which is the graph $[\Sigma]\ \Gamma \restriction \lambda$ with the edges from λ and no jump edge).

A *switch edge* of a $\parr\backslash\&$-vertex N is an in-edge of N, *i.e.* an edge between N and one of its premises or a jump to N. A *switching cycle* is a cycle with at most one switch edge of each $\parr\backslash\&$-vertex. A *$\parr$-switching* of a linking λ is any subgraph of $\mathcal{G}_{\lambda}$ obtained by deleting a switch edge of each $\parr$-vertex; denoting by ϕ this choice of edges, the subgraph it yields is $\mathcal{G}_{\phi}$.

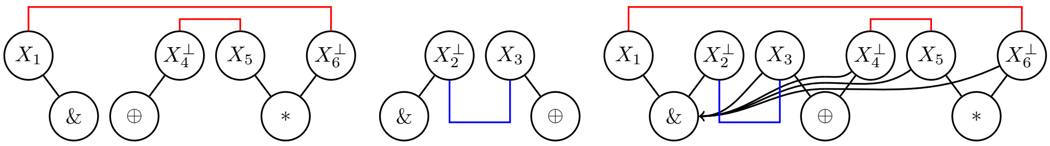


Figure 1 Graphs from an example of a proof-net: from left to right $\mathcal{G}_{\lambda_1}$, $\mathcal{G}_{\lambda_2}$ and $\mathcal{G}_{\{\lambda_1;\lambda_2\}}$.

▶ **Definition 5** (Proof-net). *A* unit-free MALL proof-net *θ on a cut sequent $[\Sigma]\ \Gamma$ is a set of linkings satisfying:*

(P0) Cut: *Every cut pair of Σ has a leaf in θ.*

(P1) Resolution: *Exactly one linking of θ is on any given &-resolution of $[\Sigma]\ \Gamma$.*

(P2) MLL: *For every ⅋-switching ϕ of every linking $\lambda \in \theta$, $\mathcal{G}_\phi$ is a tree.*

(P3) Toggling: *Every set $\Lambda \subseteq \theta$ of two or more linkings toggles a &-vertex that is in no switching cycle of $\mathcal{G}_\Lambda$.*

These conditions are called the *correctness criterion.* Condition (P0) is here to prevent unused $*$-vertices. A *cut-free* proof-net is one without $*$-vertices (it respects (P0) trivially). Condition (P1) is a correctness criterion for ALL proof-nets [10] and (P2) is the Danos-Regnier criterion for MLL proof-nets [3]. However, (P1) and (P2) together are insufficient for cut-free MALL proof-nets, hence the last condition (P3) taking into account interactions between the slices (see also [5] for a similar condition for example). Sets composed of a single linking λ are not considered in (P3), for by (P2) the graph $\mathcal{G}_\lambda$ has no switching cycle.

An example of proof-net, illustrated on Figure 1, is the following. On the cut sequent $[X_5 * X_6^\perp]\ X_1 \,\&\, X_2^\perp, X_3 \oplus X_4^\perp$ (where each X_i is an occurrence of the same atom X), set $\lambda_1 = \{(X_1, X_6^\perp), (X_4^\perp, X_5)\}$ and $\lambda_2 = \{(X_2^\perp, X_3)\}$. One can check $\{\lambda_1; \lambda_2\}$ is a proof-net.

In the particular setting of isomorphisms, we mainly consider proof-nets with two conclusions. This allows to define a notion of duality on leaves and connectives. Consider a cut sequent containing both A and $A^\perp$. For V a vertex in (the syntax tree $T(A)$ of) A, we denote by $V^\perp$ the corresponding vertex in $A^\perp$. As expected, $V^{\perp\perp} = V$. This also respects orthogonality for formulas on leaves: given a leaf l of A, labeled by a formula X, the label of $l^\perp$ is $X^\perp$. We can also define a notion of duality on premises: given a premise of a vertex $V \in T(A)$, the dual premise of $V^\perp$ is the corresponding premise in $T(A^\perp)$. In other words, if in $L - V - R$ we consider the premise L then in $R^\perp - V^\perp - L^\perp$ its dual premise is $L^\perp$.

▶ **Definition 6** (Composition). *Given proof-nets θ and ϑ of respective conclusions $[\Sigma]\ \Gamma, A$ and $[\Xi]\ \Delta, A^\perp$, the* composition *over A of θ and ϑ is the proof-net $\theta \overset{A}{\bowtie} \vartheta = \{\lambda \cup \mu \mid \lambda \in \theta, \mu \in \vartheta\}$, with conclusions $[\Sigma, \Xi, A * A^\perp]\ \Gamma, \Delta$.*

For example, see Figure 7 with a composition of the proof-nets on Figure 5.

▶ **Definition 7** (Cut-elimination). *Let θ be a set of linkings on a cut sequent $[\Sigma]\ \Gamma$, and $A * A^\perp$ a cut pair in Σ. Define the* elimination *of $A * A^\perp$ (or of the cut $*$ between A and $A^\perp$) as:*

(a) *If A is an atom, delete $A * A^\perp$ from Σ and replace any pair of links (l, A), $(A^\perp, m)$ (l and m being other occurrences of $A^\perp$ and A respectively) with the link (l, m).*

(b) *If $A = A_1 \otimes A_2$ and $A^\perp = A_2^\perp ⅋ A_1^\perp$ (or vice-versa), replace $A * A^\perp$ with two cut pairs $A_1 * A_1^\perp$ and $A_2 * A_2^\perp$. Retain all original linkings.*

(c) *If $A = A_1 \,\&\, A_2$ and $A^\perp = A_2^\perp \oplus A_1^\perp$ (or vice-versa), replace $A * A^\perp$ with two cut pairs $A_1 * A_1^\perp$ and $A_2 * A_2^\perp$. Delete all* inconsistent *linkings, namely those $\lambda \in \theta$ such that in $[\Sigma]\ \Gamma \upharpoonright \lambda$ the children & and $\oplus$ of the cut do not take dual premises. Finally, "garbage collect" by deleting any cut pair $B * B^\perp$ for which no leaf of $B * B^\perp$ is in any of the remaining linkings.*

See Figure 8 for a result on applying steps (b) and (c) to the proof-net of Figure 7.

▶ **Proposition 8** (Proposition 5.4 in [10]). *Eliminating a cut in a proof-net yields a proof-net.*

▶ **Theorem 9** (Theorem 5.5 in [10]). *Cut-elimination of proof-nets is strongly normalizing and confluent.*

A linking λ on a cut sequent $[\Sigma]\ \Gamma$ *matches* if, for every cut pair $A * A^\perp$ in Σ, any given leaf l of A is in $[\Sigma]\ \Gamma \restriction \lambda$ if and only if $l^\perp$ of $A^\perp$ is in $[\Sigma]\ \Gamma \restriction \lambda$. A linking matches if and only if, when cut-elimination is carried out, the linking never becomes inconsistent, and thus is never deleted. This allows defining *Turbo Cut-elimination* [10], eliminating a cut in a single step by removing inconsistent linkings.

3 Completeness for unit-free MALL

Our method relates closely to the one used by Balat and Di Cosmo in [2]. We work on proof-nets, as they highly simplify the problem by representing proofs up to rule commutations [11]. We start by transposing the study of unit-free MALL isomorphisms to proof-nets of a particular shape, called *bipartite full* (Sections 3.1 and 3.2). Then, we use the distributivity isomorphisms to reduce the problem to special formulas, called *distributed*, allowing to consider even more constrained proof-nets (Section 3.3). These are the key differences with the proof in MLL from [2], where some properties are given for free as there are no slice nor distributivity isomorphism. From this point the problem is similar to unit-free MLL, with commutativity and associativity only. We conclude as in [2]: restricting the problem to so-called *non-ambiguous* formulas, isomorphisms are easily characterized (Section 3.4).

3.1 Reduction to proof-nets

We desequentialize a unit-free MALL proof π (with expanded axioms) into a proof-net $\mathcal{R}(\pi)$ by induction on π using the steps detailed on Figure 2, following [10] with the notation $\theta \triangleright [\Sigma]\ \Gamma$ for "θ is a set of linkings on the cut sequent $[\Sigma]\ \Gamma$". As identified in Section 5.3.4 of [10], desequentializing with both *cut* and &-rules is complex, for cuts can be shared (or not) when translating a &-rule: $\dfrac{\theta \triangleright [\Sigma, \Xi]\ A, \Gamma \qquad \vartheta \triangleright [\Sigma, \Phi]\ B, \Gamma}{\theta \cup \vartheta \triangleright [\Sigma, \Xi, \Phi]\ A \mathbin{\&} B, \Gamma}\,\&$. We choose to never share cuts ($\Sigma = \emptyset$), thus desequentialization is a function. The cost being that the following $\& - \mathit{cut}$ commutation yields different proof-nets (contrary to the other commutations, see [11]).

$$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A, B, \Gamma} \qquad \dfrac{\pi_2}{\vdash A, C, \Gamma}}{\vdash A, B \mathbin{\&} C, \Gamma}\,\& \qquad \dfrac{\pi_3}{\vdash A^\perp, \Delta}}{\vdash B \mathbin{\&} C, \Gamma, \Delta}\,\mathit{cut} \quad \equiv \quad \dfrac{\dfrac{\dfrac{\pi_1}{\vdash A, B, \Gamma} \qquad \dfrac{\pi_3}{\vdash A^\perp, \Delta}}{\vdash B, \Gamma, \Delta}\,\mathit{cut} \qquad \dfrac{\dfrac{\pi_2}{\vdash A, C, \Gamma} \qquad \dfrac{\pi_3}{\vdash A^\perp, \Delta}}{\vdash C, \Gamma, \Delta}\,\mathit{cut}}{\vdash B \mathbin{\&} C, \Gamma, \Delta}\,\&$$

▶ **Theorem 10** (Sequentialization, Theorem 5.9 in [10]). *A set of linkings on a cut sequent is a translation of a MALL proof if and only if it is a proof-net.*

▶ **Definition 11** (Identity proof-net). *We call* identity proof-net *of a unit-free MALL formula A, the proof-net corresponding to the proof* id_A *(the axiom-expansion of* $\dfrac{}{\vdash A^\perp, A}\,\mathit{ax}$ *).*

▶ **Theorem 12** (Simulation Theorem). *Let π and ϖ be unit-free MALL proof trees (with expanded axioms). If $\pi =_\beta \varpi$, then $\mathcal{R}(\pi) =_\beta \mathcal{R}(\varpi)$.*

A notion of isomorphism $A \overset{\theta, \vartheta}{\simeq} B$ can be defined directly on proof-nets: θ and ϑ are two cut-free proof-nets of respective conclusions $A^\perp, B$ and $B^\perp, A$ such that $\theta \overset{B}{\bowtie} \vartheta$ and $\vartheta \overset{A}{\bowtie} \theta$ reduce by cut-elimination to identity proof-nets. Using the Simulation Theorem, we obtain:

$$\dfrac{}{\{\{(X,X^\perp)\}\} \rhd [\emptyset]\ X, X^\perp}\,ax \qquad \dfrac{\theta \rhd [\Sigma]\ \Gamma}{\theta \rhd [\Sigma]\ \sigma(\Gamma)}\,ex$$

$$\dfrac{\theta \rhd [\Sigma]\ A, \Gamma \qquad \vartheta \rhd [\Xi]\ A^\perp, \Delta}{\{\lambda \cup \mu \mid \lambda \in \theta, \mu \in \vartheta\} \rhd [\Sigma, \Xi, A * A^\perp]\ \Gamma, \Delta}\,cut$$

$$\dfrac{\theta \rhd [\Sigma]\ A, \Gamma \qquad \vartheta \rhd [\Xi]\ B, \Delta}{\{\lambda \cup \mu \mid \lambda \in \theta, \mu \in \vartheta\} \rhd [\Sigma, \Xi]\ A \otimes B, \Gamma, \Delta}\,\otimes \qquad \dfrac{\theta \rhd [\Sigma]\ A, B, \Gamma}{\theta \rhd [\Sigma]\ A \parr B, \Gamma}\,\parr$$

$$\dfrac{\theta \rhd [\Xi]\ A, \Gamma \qquad \vartheta \rhd [\Phi]\ B, \Gamma}{\theta \cup \vartheta \rhd [\Xi, \Phi]\ A \,\&\, B, \Gamma}\,\& \qquad \dfrac{\theta \rhd [\Sigma]\ A, \Gamma}{\theta \rhd [\Sigma]\ A \oplus B, \Gamma}\,\oplus_1 \qquad \dfrac{\theta \rhd [\Sigma]\ B, \Gamma}{\theta \rhd [\Sigma]\ A \oplus B, \Gamma}\,\oplus_2$$

We use the implicit tracking of formula occurrences downwards through the rules.

Figure 2 Inductive definition of the translation of unit-free MALL proof trees to sets of linkings.

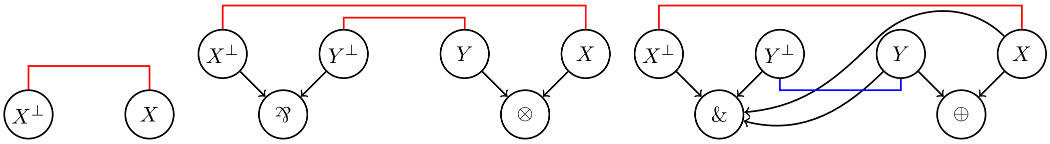


Figure 3 Identity proof-nets (from left to right: atoms, $\parr\backslash\otimes$ and $\&\backslash\oplus$).

▶ **Theorem 13** (Type isomorphisms in proof-nets). *Let A and B be two unit-free MALL formulas. If $A \simeq B$ then there exist two proof-nets θ and ϑ such that $A \overset{\theta,\vartheta}{\simeq} B$.*

3.2 Reduction to bipartite full proof-nets

▶ **Definition 14** (Full, *Ax*-unique, Bipartite proof-net). *A cut-free proof-net is called* full *if any of its leaves has (at least) one link on it. Furthermore, if for any leaf there exists a unique link on it (possibly shared among several linkings), then we call this proof-net ax-*unique.

A cut-free proof-net is bipartite *if it has two conclusions, A and B, and each of its links is between a leaf of A and a leaf of B (no link between leaves of A, or between leaves of B).*

We show identity proof-nets are bipartite ax-unique, and isomorphisms are bipartite full.

Using an induction on the formula A, we can prove the following results on the identity proof-net of A (see Figure 3 for a graphical intuition).

▶ **Proposition 15.**

(i) *An identity proof-net is bipartite ax-unique.*

(ii) *The axiom links of an identity proof-net are exactly the $(l, l^\perp)$, for any leaf l.*

(iii) *In the identity proof-net of A, exactly one linking is on any given additive resolution of the conclusion A.*

Neither fullness, ax-uniqueness nor bipartiteness is preserved by cut anti-reduction. A counter-example is given on Figure 4, with a non bipartite proof-net and a non full one whose composition reduces to the identity proof-net (bipartite ax-unique by Proposition 15(i)).[2] However, if *both* compositions yield identity proof-nets, we get bipartiteness and fullness.

[2] This example gives a retraction between $(A \parr A^\perp) \otimes B$ and $((A \parr A^\perp) \otimes B) \oplus B$ in MALL which is not an isomorphism (as is the retraction between A and $(A \multimap A) \multimap A = (A \otimes A^\perp) \parr A$ in MLL).

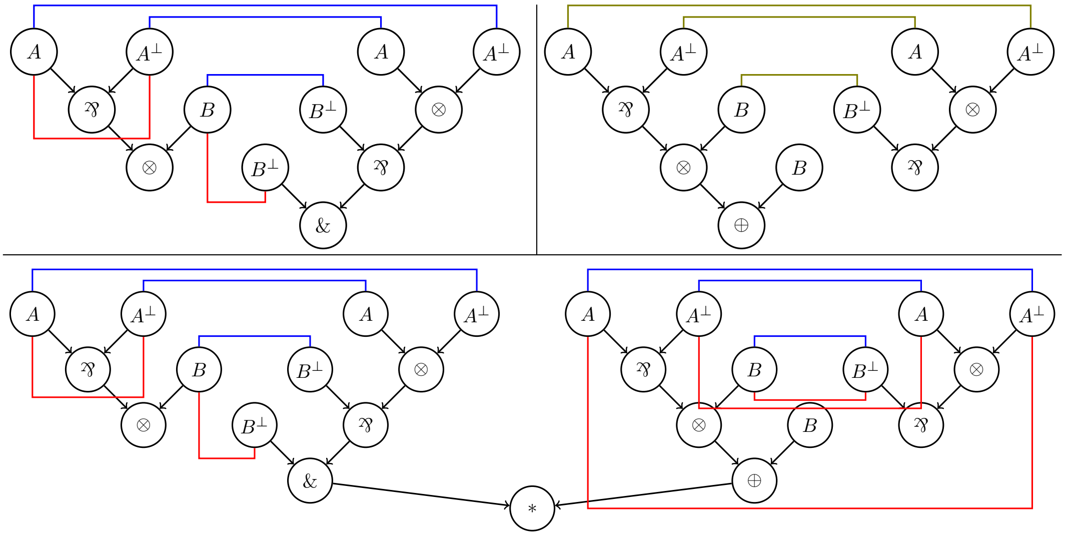


Figure 4 Non bipartite proof-net (top-left), non full proof-net (top-right) and one of their composition yielding the identity proof-net (bottom) (jump edges not represented).

▶ **Lemma 16.** *Let θ and θ' be cut-free proof-nets of respective conclusions $A^\perp, B$ and $B^\perp, A$, such that $\theta' \overset{A}{\bowtie} \theta$ reduces to the identity proof-net of B. For any linking $\lambda \in \theta$, there exists $\lambda' \in \theta'$ such that $\lambda \cup \lambda'$ matches in the composition over B of θ and θ', $\theta \overset{B}{\bowtie} \theta'$.*

Proof. Let us consider a linking $\lambda \in \theta$, and call $\mathcal{C}$ the choices of premise on additive connectives of B that λ makes. We search some $\lambda' \in \theta'$ making the dual choices of premise on additive connectives of $B^\perp$ compared to $\mathcal{C}$. Consider the composition of θ and θ' over A. It reduces to the identity proof-net of B by hypothesis. By Proposition 15(iii), there exists a unique linking in the identity proof-net of B corresponding to $\mathcal{C}$. Furthermore, the linkings of the identity proof-net are derived from the $\mu \cup \mu'$ for μ a linking of θ and μ' one of θ', with $\mu \cup \mu'$ matching for a cut over A: a linking in the identity proof-net is a linking of the form $\mu \cup \mu'$ where axiom links $(l, m) \in \mu$ and $(m^\perp, l^\perp) \in \mu'$ are replaced with $(l, l^\perp)$, with l a leaf of B and m one of $A^\perp$ (because an identity proof-net has only links of the form $(l, l^\perp)$ by Proposition 15(ii)). Therefore, there exist $\mu \in \theta$ and $\mu' \in \theta'$ such that μ makes the choices $\mathcal{C}$ on B and $\mu \cup \mu'$ matches for the composition of θ and θ' over both A and B. But λ makes the same choices $\mathcal{C}$ on B as μ: $\lambda \cup \mu'$ also matches for a cut over B. ◀

▶ **Corollary 17.** *Assuming $A \overset{\theta,\theta'}{\simeq} B$, θ and θ' are bipartite.*

Proof. We proceed by contradiction: *w.l.o.g.* there is a link a in some linking $\lambda \in \theta$ which is between leaves of $A^\perp$. By Lemma 16 there exists $\lambda' \in \theta'$ such that $\lambda \cup \lambda'$ matches for a cut over B. Whence a, which does not involve leaves of B, belongs to a linking of the composition where cuts have been eliminated (it belongs to the linking resulting from $\lambda \cup \lambda'$). But this reduction yields a bipartite proof-net by Proposition 15(i), a contradiction. ◀

▶ **Lemma 18.** *Assume θ and θ' are cut-free proof-nets of respective conclusions $A^\perp, B$ and $B^\perp, A$, and that their composition over B yields the identity proof-net of A. Then any leaf of $A^\perp$ (resp. A) has (at least) one axiom link on it in θ (resp. θ').*

▶ **Theorem 19.** *Assuming $A \overset{\theta,\theta'}{\simeq} B$, θ and θ' are bipartite full.*

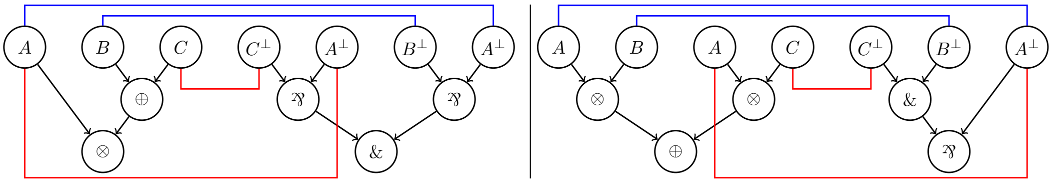


Figure 5 Proof-nets for $A \otimes (B \oplus C) \simeq (A \otimes B) \oplus (A \otimes C)$.

3.3 Distribution

In general, isomorphisms do not yield *ax*-unique proof-nets. A counter-example is distributivity: $A \otimes (B \oplus C) \simeq (A \otimes B) \oplus (A \otimes C)$, see Figure 5. Nonetheless, distributivity equations are the only ones in $\mathcal{E}^\dagger$ not giving *ax*-unique proof-nets. We will restrict our study to so-called *distributed* formulas. Once formulas are distributed, distributivity isomorphisms can be ignored, and isomorphisms between distributed formulas happen to be bipartite *ax*-unique.

▶ **Definition 20** (Distributed formula). *An MALL formula is* distributed *if it does not have any sub-formula of the form $A \otimes (B \oplus C)$, $(A \oplus B) \otimes C$, $A \otimes 1$, $1 \otimes A$, $A \oplus 0$, $0 \oplus A$, $A \otimes 0$, $0 \otimes A$ or their duals $(C \,\&\, B) \parr A$, $C \parr (B \,\&\, A)$, $\bot \parr A$, $A \parr \bot$, $\top \,\&\, A$, $A \,\&\, \top$, $\top \parr A$, $A \parr \top$ (where A, B and C are any formulas).*

▶ Remark. This notion is stable by duality: if A is distributed, so is $A^\perp$.

▶ **Proposition 21.** *If $\mathcal{E}$ is complete for isomorphisms between distributed formulas, then it is complete for isomorphisms between arbitrary formulas.*

Proof. Up to equations of Table 1, any formula can be rewritten into a distributed one. ◀

We mostly use the correctness criterion through the fact we can sequentialize, *i.e.* recover a proof tree from a proof-net by Theorem 10. However, in order to prove *ax*-uniqueness, we make a direct use of the correctness criterion to deduce geometric properties of proof-nets. This part of the proof takes benefits from the specificities of this syntax. We begin with two preliminary results. For Λ a set of linkings and W a &-vertex, Λ^W denote the set of all linkings in Λ whose additive resolution does not contain the right argument of W.

▶ **Lemma 22** (Lemma 4.32 in [10], adapted). *Let ω be a jump-free switching cycle in a proof-net θ. There exists a subset of linkings $\Lambda \subseteq \theta$ such that $\omega \subseteq \mathcal{G}_\Lambda$, $\omega \not\subseteq \mathcal{G}_{\Lambda^W}$ and for any &-vertex W toggled by Λ, there exists an axiom link $a \in \omega$ depending on W in Λ.*

For U and V vertices in a tree, their *first common descendant* is the vertex of the tree which is a descendant of both U and V and which has no descendant respecting this property (with a tree represented with its root at the bottom, which is a *descendant* of the leaves).

▶ **Lemma 23.** *Let θ be a proof-net of conclusions Γ, A. If there is a jump edge $l \xrightarrow{j} W$ with $l, W \in T(A)$ and W not a descendant of l, then their first common descendant C is a $\parr$.*

Proof. As there is a jump $l \xrightarrow{j} W$, there exist linkings $\lambda, \lambda' \in \theta$ such that W is the only & toggled by $\{\lambda; \lambda'\}$, and a link $a \in \lambda \backslash \lambda'$ using the leaf l. In particular, the jump $l \xrightarrow{j} W$ is in $\mathcal{G}_{\{\lambda;\lambda'\}}$. For l and W are both in the additive resolution of λ, both premises of C are in the additive resolution of λ, thus C cannot be an additive connective, so not a & nor a ⊕-vertex.

Assume by contradiction that C is a ⊗. Call δ the path in $T(A)$ from W to C, and μ the one from C to l (see Figure 6). Then, $(l \xrightarrow{j} W)\delta\mu$ is a switching cycle in $\mathcal{G}_{\{\lambda;\lambda'\}}$. According to (P3), there exists a & toggled by $\{\lambda; \lambda'\}$ not in any switching cycle of $\mathcal{G}_{\{\lambda;\lambda'\}}$. A contradiction, for W is the only & toggled by $\{\lambda; \lambda'\}$. Whence, C can only be a $\parr$. ◀

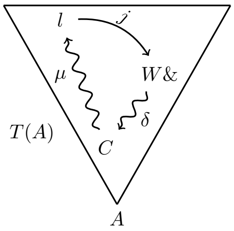


Figure 6 Illustration of the proof of Lemma 23.

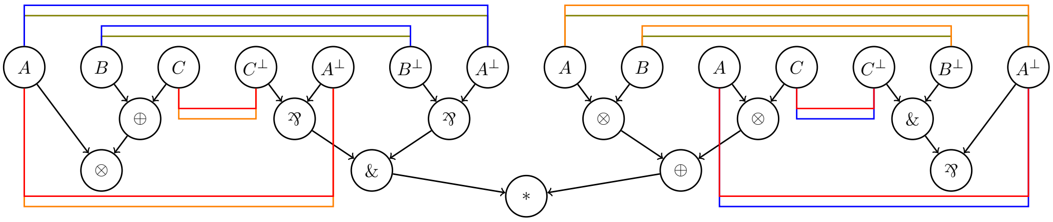


Figure 7 Proof-nets from Figure 5 composed by cut on $(A \otimes B) \oplus (A \otimes C)$.

Now, let us prove that isomorphisms of distributed formulas are bipartite *ax*-unique. We will consider proof-nets corresponding to an isomorphism that we cut and where we eliminate all cuts not involving atoms. To give some intuition, let us consider the non-*ax*-unique proof-nets of Figure 5. Composing them together by cut on $(A \otimes B) \oplus (A \otimes C)$ gives the proof-net illustrated on Figure 7. Reducing all cuts not involving atoms yields the proof-net on Figure 8, that we call an *almost reduced composition*. We stop there because of the switching cycle produced by the two links on A (dashed in blue on Figure 8), less visible in the non-reduced composition of Figure 7. However, reducing all cuts gives the identity proof-net, which has no switching cycle: during these reductions, both links on A are merged. By using almost reduced composition, we are going to prove that links preventing *ax*-uniqueness yield switching cycles, and moreover that these cycles are due to non-distributed formulas only.

▶ **Definition 24** (Almost reduced composition). *Take θ and θ' cut-free proof-nets of respective conclusions A, B and $B^\perp, C$. The* almost reduced composition *over B of θ and θ' is the proof-net resulting from the composition over B of θ and θ' where we repeatedly reduce all cuts not involving atoms (*i.e. *not applying step (a) of Definition 7).*

Let us fix A and B two unit-free MALL (not necessarily distributed yet) formulas as well as θ and θ' such that $A \overset{\theta,\theta'}{\simeq} B$. By Theorem 19, θ and θ' are bipartite full. We denote by ϑ the almost reduced composition over B of θ and θ'. Here, we can extend our duality on vertices and premises (defined in Section 2.4) to links.

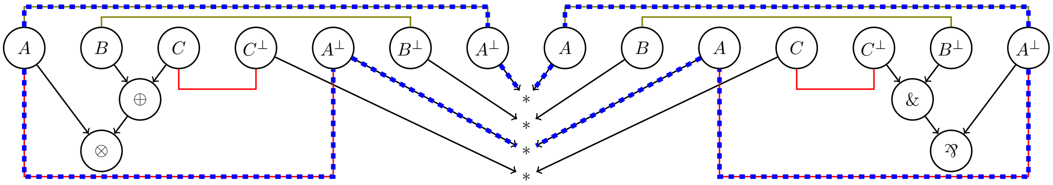


Figure 8 An almost reduced composition of the proof-nets on Figure 5.

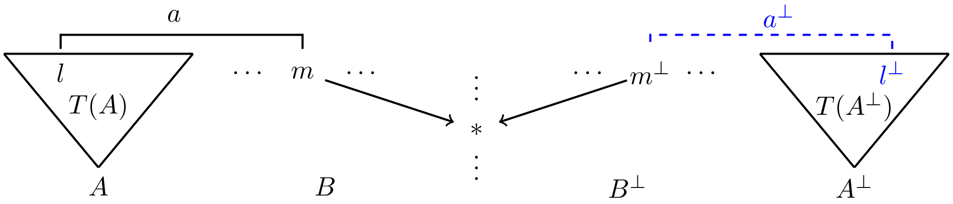


Figure 9 Illustration of Lemma 25.

▶ **Lemma 25.** *Given l a leaf of A (resp. $A^{\perp}$) and m one of $B^{\perp}$ (resp. B), there is an axiom link $a = (l, m)$ in some linking $\lambda \in \vartheta$ if and only if there is an axiom link $(l^{\perp}, m^{\perp})$ in the same linking λ, that we will denote $a^{\perp} = (l^{\perp}, m^{\perp})$ (see Figure 9).*

Proof. By symmetry, assume $(l, m) \in \lambda \in \vartheta$. As the cut $m * m^{\perp}$ belongs to the additive resolution of λ (for m is inside), $m^{\perp}$ is a leaf in this resolution. Thus, there is a link $(m^{\perp}, l') \in \lambda$ for some leaf l', which necessarily belongs to A by bipartiteness of θ'. It stays to prove $l' = l^{\perp}$. If we were to eliminate all cuts in ϑ, we would get the identity proof-net on A by hypothesis. But eliminating the cut $m * m^{\perp}$ yields a link (l, l'), which is not modified by the elimination of the other atomic cuts. By Proposition 15(ii), $l' = l^{\perp}$ follows. ◀

▶ **Lemma 26.** *Let λ be a linking of ϑ, and V an additive vertex in its additive resolution. Then $V^{\perp}$ is also inside, with as premise kept the dual premise of the one kept for V.*

▶ **Lemma 27.** *Let W and P be respectively a &-vertex and a $\oplus$-vertex in ϑ, with W an ancestor of P. Then for any axiom link a depending on W in ϑ, a also depends on $P^{\perp}$ in ϑ.*

Proof. There exist linkings $\lambda, \lambda' \in \vartheta$ such that W is the only & toggled by $\{\lambda; \lambda'\}$ and $a \in \lambda \backslash \lambda'$. We consider a linking $\lambda_{P^{\perp}}$ defined by taking an arbitrary &-resolution of λ where we choose the other premise for $P^{\perp}$ (and arbitrary premises for &-vertices introduced this way): by (P1), there exists a unique linking on it. By Lemma 26, the additive resolutions of λ and $\lambda_{P^{\perp}}$ (resp. λ and λ') differ exactly on ancestors of P and $P^{\perp}$ (resp. W and $W^{\perp}$). Thus, the additive resolutions of λ' and $\lambda_{P^{\perp}}$ also differ exactly on ancestors of P and $P^{\perp}$, for W is an ancestor of P. In particular, $\{\lambda; \lambda_{P^{\perp}}\}$, as well as $\{\lambda'; \lambda_{P^{\perp}}\}$, toggles only $P^{\perp}$. If $a \in \lambda_{P^{\perp}}$, then a depends on $P^{\perp}$ in $\{\lambda'; \lambda_{P^{\perp}}\}$. Otherwise, a depends on $P^{\perp}$ in $\{\lambda; \lambda_{P^{\perp}}\}$. ◀

The key result to use distributivity is that a positive vertex "between" a leaf l and a &-vertex W in the same tree prevents them from interacting, *i.e.* there is no jump $l \xrightarrow{j} W$.

▶ **Lemma 28.** *Let $l \xrightarrow{j} W$ be a jump edge in ϑ, with l not an ancestor of W and $l, W \in T(A^{\perp})$ (resp. $T(A)$). Denoting by N the first common descendant of l and W, there is no positive vertex in the path between N and W in $T(A^{\perp})$ (resp. $T(A)$).*

Proof. Let P be a vertex on the path between N and W in $T(A^{\perp})$. By Lemma 23, N is a $⅋$-vertex. We prove by contradiction that P can neither be a $\oplus$ nor a $\otimes$-vertex.

Suppose P is a $\oplus$-vertex. By Lemma 27, a depends on $P^{\perp}$, and so does $a^{\perp}$ through Lemma 25: there is a jump edge $l^{\perp} \xrightarrow{j} P^{\perp}$. Applying Lemma 23, the first common descendant of $l^{\perp}$ and $P^{\perp}$, which is $N^{\perp}$, is a $⅋$-vertex: a contradiction as it is a $\otimes$-vertex.

Assume now P to be a $\otimes$-vertex. As there is a jump $l \xrightarrow{j} W$, there exist linkings $\lambda, \lambda' \in \vartheta$ and a leaf m of B such that W is the only & toggled by $\{\lambda; \lambda'\}$ and $a = (l, m) \in \lambda \backslash \lambda'$. For P is a $\otimes$, there is a leaf p which is an ancestor of P in the additive resolution of λ, from a different

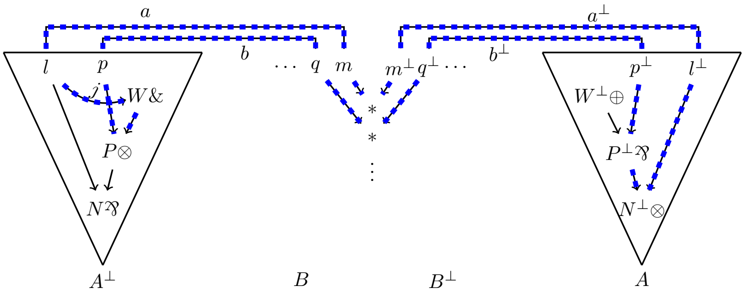


Figure 10 Switching cycle containing W if P is a ⊗-vertex in the proof of Lemma 28.

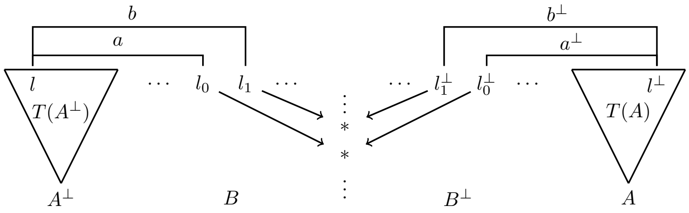


Figure 11 Almost reduced composition ϑ of θ and θ' by cut over B in the proof of Theorem 29.

premise of P than W; it is used by a link $b = (p, q) \in \lambda$[3] (see Figure 10). Then the switching cycle $l \xrightarrow{j} W \to P \leftarrow p \overset{b}{-} q \to * \leftarrow q^\perp \overset{b^\perp}{-} p^\perp \to P^\perp \to N^\perp \leftarrow l^\perp \overset{a^\perp}{-} m^\perp \to * \leftarrow m \overset{a}{-} l$ (dashed in blue on Figure 10) belongs to $\mathcal{G}_{\{\lambda;\lambda'\}}$. Contradiction: W, the only & toggled by $\{\lambda; \lambda'\}$, cannot be in any switching cycle of $\mathcal{G}_{\{\lambda;\lambda'\}}$ by (P3). ◀

▶ **Theorem 29.** *Assuming $A \overset{\theta,\theta'}{\simeq} B$ with A and B distributed, θ and θ' are bipartite ax-unique.*

Proof. We already know that θ and θ' are bipartite full thanks to Theorem 19. We reason by contradiction and assume *w.l.o.g.* that θ is not *ax*-unique: there exist a leaf l of $A^\perp$ and two distinct leaves l_0 and l_1 of B with links $a = (l, l_0)$ and $b = (l, l_1)$ in θ. We consider ϑ the almost reduced composition of θ and θ' over B, depicted on Figure 11. By Lemma 16, a and b are also links in ϑ (for the linkings they belong to in θ have matching linkings in θ', and we did not eliminate atomic cuts). Using Lemma 25, we have in $\mathcal{G}_\vartheta$ a switching cycle $\omega = l \overset{a}{-} l_0 \to * \leftarrow l_0^\perp \overset{a^\perp}{-} l^\perp \overset{b^\perp}{-} l_1^\perp \to * \leftarrow l_1 \overset{b}{-} l$.

Let Λ be a set of linkings given by Lemma 22 applied to ω. As there are two distinct links on l in $\omega \subseteq \mathcal{G}_\Lambda$, Λ contains at least two linkings. By (P3), there exists W a & toggled by Λ that is not in any switching cycle of $\mathcal{G}_\Lambda$. By Lemma 22, a, $a^\perp$, b or $b^\perp$ depends on W. So a or b depends on W by Lemma 25; *w.l.o.g.* a depends on W. The vertex W belongs to either $T(A)$ or $T(A^\perp)$: up to considering $a^\perp$ instead of a, W is in $T(A^\perp)$. Remark l is not an ancestor of W: if it were, by symmetry assume it is a left-ancestor. Whence a and b belong to Λ^W, so $a^\perp$ and $b^\perp$ too (Lemma 25); thus $\omega \subseteq \mathcal{G}_{\Lambda^W}$, contradicting Lemma 22. By Lemma 23, the first common descendant N of l and W in $T(A^\perp)$ is a ⅋. There is a ⊗\⊕ on the path between the ⅋ N and its ancestor the & W in $T(A^\perp)$, for there is no sub-formula of the shape $-$ ⅋ ($-$ & $-$) in the distributed $A^\perp$. This contradicts Lemma 28. ◀

[3] With $q \neq m$, as a and b are two distinct links in the same linking λ.

3.4 Non-ambiguous formulas & Completeness for unit-free MALL

Once our study is restricted to bipartite *ax-unique* proof-nets, we can also restrict formulas.

▶ **Definition 30** (Non-ambiguous formula). *A formula A is said* non-ambiguous *if each atom in A occurs at most once positive and once negative.*

▶ Remark. This means all leaves in A are distinct. If A is non-ambiguous, so is $A^{\perp}$.

For instance, $X \mathbin{\&} X^{\perp}$ is non-ambiguous, whereas $(A \otimes B) \oplus (A \otimes C)$ is ambiguous. The reduction to non-ambiguous formulas requires to restrict to distributed formulas first: in $(A \otimes B) \oplus (A \otimes C) \simeq A \otimes (B \oplus C)$ we need the two occurrences of A to factorize. The two following results are a direct adaptation of Section 3 in [2].

▶ **Corollary 31** (Reduction to distributed non-ambiguous formulas). *The set of couples of distributed formulas A and B such that $A \overset{\theta,\vartheta}{\simeq} B$ is the set of instances (by a substitution on atoms) of couples of distributed* non-ambiguous *formulas A' and B' such that $A' \overset{\theta',\vartheta'}{\simeq} B'$.*

▶ **Corollary 32.** *Let A and B be non-ambiguous formulas. If there exist bipartite proof-nets θ and ϑ of respective conclusions $A^{\perp}, B$ and $B^{\perp}, A$, then $A \overset{\theta,\vartheta}{\simeq} B$.*

We then prove the completeness of $\mathcal{E}^{\dagger}$ for unit-free MALL by reasoning as in Section 4 of [2] (with some more technicalities for we reorder not only ⅋-vertices but also &-vertices).

▶ **Theorem 33** (Isomorphisms completeness for unit-free MALL). *Given A and B two unit-free MALL formulas, if $A \simeq B$, then $A =_{\mathcal{E}^{\dagger}} B$.*

4 Completeness for MALL with units

We now consider full MALL, with units, and show how to reduce it to the unit-free case. We solve this addition purely in sequent calculus showing that, for distributed formulas, multiplicative and additive units can be replaced by fresh atoms.

A key property of proof-nets is to define a quotient of sequent calculus proofs up to rule commutations [11] (see Appendix A for rule commutations in MALL). Because no such notion of proof-nets exist with units, we are forced to stay in the sequent calculus, meaning that we have to deal with possible rule commutations. As a key example, cut-elimination in proof-nets is confluent and leads to a unique normal form. This is not true in the sequent calculus and we need to relate the different possible cut-free proofs obtained by cut-elimination.

▶ **Theorem 34** (Confluence up to rule commutations). *If π_1 and π_2 are cut-free proofs obtained by cut-elimination from the same proof π, then π_1 and π_2 are equal up to rule commutations.*

This result is not surprising but has not already been proved as far as we know for it is rather tedious to establish. It is an important general result about sequent calculus which we are convinced should hold for full linear logic. It can be lifted to $\beta\eta$-equality of proofs.

▶ **Theorem 35.** *Let π and ϖ be $\beta\eta$-equal MALL proofs. Then, letting π' (resp. ϖ') be a result of expanding all axioms and then eliminating all cuts in π (resp. ϖ), π' is equal to ϖ' up to rule commutations.*

After these general properties, let us move to the question of type isomorphisms. We need to analyse the behaviour of units in proofs equal to id_A up to rule commutations. We only do so for a *distributed* formula A as we have already seen it is enough in Section 3.3.

▶ **Proposition 36.** *Let π be a proof equal, up to rule commutations, to* id_A *with* A distributed. *The* $\top$*-rules of* π *are of the shape* $\dfrac{}{\vdash \top, 0}\,\top$ *(with* $\top$ *in* A *being the dual of* 0 *in* $A^\perp$*, or vice-versa) and* $\bot$*-rules and* 1*-rules come by pairs separated with* $\oplus_i$*-rules only, called a* $1/\oplus/\bot$-pattern*:* $\dfrac{\dfrac{\dfrac{}{\vdash 1}\,1}{\vdash F}\,\rho}{\vdash \bot, F}\,\bot$ *where* ρ *is a sequence of* $\oplus_i$*-rules (with* $\bot$ *in* A *being the dual of* 1 *in* $A^\perp$*, or vice-versa).*

Proof. The key idea is to find properties of id_A preserved by all rule commutations and ensuring the properties described in the statement. For any sequent S in the proof:

(1) the formulas of S are distributed;

(2) if $\top$ is a formula of S, then $S = \vdash \top, 0$;

(3) if $\bot$ is a formula of S, then $S = \vdash \bot, F$ with F given by the following grammar $F \;:=\; 1 \mid F \oplus D \mid D \oplus F$, where the distinguished 1 is the dual of $\bot$ in $A^\perp$ if $\bot$ a sub-formula of A (or vice-versa), D is any formula, and the sub-proof of π above S is a sequence of $\oplus_i$ rules leading to the distinguished 1;

(4) if $B \,\&\, C$ is a formula of S, then $S = \vdash B \,\&\, C, F$ with F given by the following grammar $F \;:=\; C^\perp \oplus B^\perp \mid F \oplus D \mid D \oplus F$, where the distinguished $C^\perp \oplus B^\perp$ is the dual of $B \,\&\, C$ in $A^\perp$ if $B \,\&\, C$ a sub-formula of A (or vice-versa), D is any formula, and in the sub-proof of π above S the $\oplus$-rules of the distinguished $C^\perp \oplus B^\perp$ are a $\oplus_2$-rule in the left-branch of the &-rule of $B \,\&\, C$, and a $\oplus_1$-rule in its right branch;

(5) if S contains several negative formulas or several positive formulas, then its negative formulas are ⅋-formulas. ◀

These properties are preserved by cut anti-reduction.

▶ **Lemma 37.** *If* $A \overset{\pi;\pi'}{\simeq} B$ *with* π *and* π' *cut-free then all* $\top$*-rules in* π *and* π' *are of the form* $\dfrac{}{\vdash \top, 0}\,\top$ *and all* $\bot$*-rules and* 1*-rules belong to* $1/\oplus/\bot$*-patterns.*

Moving each $\bot$-rule up to the associated 1-rule (which can be done up to $\beta\eta$-equality) allows us to consider units as fresh atoms introduced by ax-rules and to apply Theorem 33.

▶ **Theorem 38** (Isomorphisms completeness with units)**.** *If* $A \simeq B$ *then* $A =_{\mathcal{E}} B$*.*

5 Star-autonomous categories with finite products

We prove here that the equational theory $\mathcal{E}$ (along $A \multimap B \simeq A^\perp ⅋ B$, De Morgan's laws and involutivity of negation) also corresponds to the isomorphisms present in all $\star$-autonomous categories with finite products. For the historical result of how linear logic can be seen as a category, see [15].

We establish this result from the one on MALL, first proving that MALL (with proofs considered up to $\beta\eta$-equality) defines a $\star$-autonomous category with finite products (Section 5.1). Then, we conclude using a semantic method (Section 5.2).

5.1 MALL as a star-autonomous category with finite products

The logic MALL, with proofs taken up to $\beta\eta$-equality, defines a $\star$-autonomous category with finite products, that we will call $\mathbb{MALL}$. Indeed, we can define it as follows.

Objects of $\mathbb{MALL}$ are formulas of MALL, while its morphisms from A to B are proofs of $\vdash A^\perp, B$, considered up to $\beta\eta$-equality.[4] One can check that a proof of MALL is an isomorphism if and only if, when seen as a morphism, it is an isomorphism in $\mathbb{MALL}$.

We define a bifunctor $\otimes$ on $\mathbb{MALL}$, associating to formulas (*i.e.* objects) A and B the formula $A \otimes B$ and to proofs (*i.e.* morphisms) π_0 and π_1 respectively of $\vdash A_0^\perp, B_0$ and $\vdash A_1^\perp, B_1$ the following proof of $\vdash (A_0 \otimes A_1)^\perp, B_0 \otimes B_1$:

$$\dfrac{\dfrac{\dfrac{\pi_0}{\vdash A_0^\perp, B_0} \qquad \dfrac{\pi_1}{\vdash A_1^\perp, B_1}}{\vdash A_1^\perp, A_0^\perp, B_0 \otimes B_1}\,\otimes}{\vdash A_1^\perp \parr A_0^\perp, B_0 \otimes B_1}\,\parr$$

One can check that $(\mathbb{MALL}, \otimes, 1, \alpha, \lambda, \rho, \gamma)$ forms a symmetric monoidal category, where 1 is the 1-formula, α are isomorphisms of MALL associated to $(A \otimes B) \otimes C \simeq A \otimes (B \otimes C)$ seen as a natural isomorphism of $\mathbb{MALL}$, and similarly for λ with $1 \otimes A \simeq A$, ρ with $A \otimes 1 \simeq A$, and γ with $A \otimes B \simeq B \otimes A$.

Furthermore, define $A \multimap B \coloneqq A^\perp \parr B$ and $ev_{A,B}$ as the following morphism from $(A \multimap B) \otimes A$ to B (*i.e.* a proof of $\vdash A^\perp \parr (B^\perp \otimes A), B$):

$$\dfrac{\dfrac{\dfrac{}{\vdash B^\perp, B}\,ax \qquad \dfrac{}{\vdash A^\perp, A}\,ax}{\vdash A^\perp, B^\perp \otimes A, B}\,\otimes}{\vdash A^\perp \parr (B^\perp \otimes A), B}\,\parr$$

It can be checked that $\mathbb{MALL}$ is a symmetric monoidal closed category with as exponential object $(A \multimap B, ev_{A,B})$ for objects A and B.

Moreover, one can also check that $\bot$ is a dualizing object for this category, making $\mathbb{MALL}$ a $\star$-autonomous category. This relies on the following morphism from $(A \multimap \bot) \multimap \bot$ to A (which is an inverse of the curryfication of $ev_{A,\bot}$):

$$\dfrac{\dfrac{}{\vdash 1}\,1 \qquad \dfrac{\dfrac{\dfrac{}{\vdash A^\perp, A}\,ax}{\vdash A^\perp, \bot, A}\,\bot}{\vdash A^\perp \parr \bot, A}\,\parr}{\vdash 1 \otimes (A^\perp \parr \bot), A}\,\otimes$$

Finally, $\top$ is a terminal object of $\mathbb{MALL}$, and $A \,\&\, B$ is the product of objects A and B, with as projections π_A and π_B the following morphisms respectively from $A \,\&\, B$ to A and from $A \,\&\, B$ to B:

$$\dfrac{\dfrac{}{\vdash A^\perp, A}\,ax}{\vdash B^\perp \oplus A^\perp, A}\,\oplus_2 \qquad \text{and} \qquad \dfrac{\dfrac{}{\vdash B^\perp, B}\,ax}{\vdash B^\perp \oplus A^\perp, B}\,\oplus_1$$

Therefore, $\mathbb{MALL}$ is a $\star$-autonomous category with finite products [15].

5.2 Isomorphisms of star-autonomous categories with finite products

We take the same notations as in the previous section (& for product, ...). One can easily check that isomorphisms in a $\star$-autonomous category with finite products form a congruence (as all binary connectives define bifunctors), and that $\mathcal{E}$ is sound (*i.e.* that equations defining

[4] We recall that $(\cdot)^\perp$ is defined by induction, making it an involution.

Table 2 De Morgan's isomorphisms.

$$\begin{array}{rcl@{\qquad\qquad}rcl}
A \multimap B & \simeq & A^{\perp} \parr B & X^{\perp\perp} & \simeq & X \\
(A \otimes B)^{\perp} & \simeq & B^{\perp} \parr A^{\perp} & (A \parr B)^{\perp} & \simeq & B^{\perp} \otimes A^{\perp} \\
1^{\perp} & \simeq & \perp & \perp^{\perp} & \simeq & 1 \\
(A \,\&\, B)^{\perp} & \simeq & B^{\perp} \oplus A^{\perp} & (A \oplus B)^{\perp} & \simeq & B^{\perp} \,\&\, A^{\perp} \\
\top^{\perp} & \simeq & 0 & 0^{\perp} & \simeq & \top
\end{array}$$

$\mathcal{E}$ in Table 1 on Page 3 are isomorphisms in any $\star$-autonomous category with finite products). Moreover the isomorphisms of Table 2 (which are equalities in $\mathbb{MALL}$) also hold in any $\star$-autonomous category with finite products.

Completeness follows by Theorem 38 (isomorphisms in $\mathbb{MALL}$ are exactly those given by $\mathcal{E}$) and from the fact that two objects definable in the language of $\star$-autonomous categories with finite products are equal in $\mathbb{MALL}$ if and only if they are related by the equational theory generated by Table 2. For example, one can deduce $(A \multimap \perp) \multimap \perp \simeq (A^{\perp} \parr \perp)^{\perp} \parr \perp \simeq (A^{\perp} \parr \perp)^{\perp} \simeq 1 \otimes A^{\perp\perp} \simeq A^{\perp\perp} \simeq A$ (the last equation being derivable by induction on A). Henceforth, isomorphisms valid in all $\star$-autonomous categories with finite products are included in $\mathcal{E}$ enriched with Table 2.

▶ **Theorem 39** (Isomorphisms in $\star$-autonomous categories with finite products). *$\mathcal{E}$ enriched with Table 2 is a sound and complete equational theory for isomorphisms in $\star$-autonomous categories with finite products.*

6 Conclusion

Extending the result of Balat and Di Cosmo in [2], we give an equational theory characterising type isomorphisms in multiplicative-additive linear logic with units as well as in $\star$-autonomous categories with finite products: the one described on Table 1 on Page 3 (together with Table 2 for $\star$-autonomous categories). Looking at the proof, we get as a sub-result that isomorphisms for ALL (resp. unit-free ALL) are given by the equational theory $\mathcal{E}$ (resp. $\mathcal{E}^{\dagger}$) restricted to ALL formulas (and more generally this applies to any fragment of MALL, thanks to the sub-formula property). Proof-nets were a major tool to prove completeness, as notions like fullness and ax-uniqueness are much harder to define and manipulate in sequent calculus. However, we could not use them for taking care of the (additive) units, because there is no known appropriate notion of proof-nets. We have thus been forced to develop (some parts of) the theory of cut-elimination, axiom-expansion and rule commutations for the sequent calculus of MALL with units.

The immediate question to address is the extension of our results to the characterization of type isomorphisms for full propositional linear logic, thus including the exponential connectives. This is clearly not immediate since the interaction between additive and exponential connectives is not well described in proof-nets.

A more general problem is the study of type retractions (where only one of the two compositions yields an identity) which is also much more difficult (see for example [13]). The question is mostly open in the case of linear logic. Even in multiplicative linear logic (where there is for example a retraction between A and $(A \multimap A) \multimap A = (A \otimes A^{\perp}) \parr A$ which is not an isomorphism, and where the associated proof-nets are not bipartite), no characterization is known. In the multiplicative-additive fragment, the problem looks even harder, with more retractions; for instance the one depicted on Figure 4, but there also is a retraction between A and $A \oplus A$.

References

1 Michele Abrusci and Elena Maringelli. A new correctness criterion for cyclic proof nets. *Journal of Logic, Language and Information*, 7:449–459, 1998. doi:10.1023/A:1008354130493.

2 Vincent Balat and Roberto Di Cosmo. A linear logical view of linear type isomorphisms. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Computer Science Logic*, volume 1683 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 1999.

3 Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 28:181–203, 1989.

4 Roberto Di Cosmo. *Isomorphisms of Types*. Progress in Theoretical Computer Science. Birkhäuser, 1995.

5 Paolo Di Giamberardino. Jump from parallel to sequential proofs: Additives, 2011. URL: https://hal.science/hal-00616386.

6 Marcelo Fiore, Roberto Di Cosmo, and Vincent Balat. Remarks on isomorphisms in typed lambda calculi with empty and sum types. In *Proceedings of the seventeenth annual symposium on Logic In Computer Science*, pages 147–156, Copenhagen, July 2002. IEEE, IEEE Computer Society Press.

7 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.

8 Jean-Yves Girard. Proof-nets: the parallel syntax for proof-theory. In Aldo Ursini and Paolo Agliano, editors, *Logic and Algebra*, volume 180 of *Lecture Notes In Pure and Applied Mathematics*, pages 97–124, New York, 1996. Marcel Dekker. doi:10.1201/9780203748671-4.

9 Willem Heijltjes and Dominic Hughes. Conflict nets: Efficient locally canonical MALL proof nets. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 437–446. ACM Press, July 2016. doi:10.1145/2933575.2934559.

10 Dominic Hughes and Rob van Glabbeek. Proof nets for unit-free multiplicative-additive linear logic. *ACM Transactions on Computational Logic*, 6(4):784–842, 2005. doi:10.1145/1094622.1094629.

11 Dominic Hughes and Rob van Glabbeek. MALL proof nets identify proofs modulo rule commutation, 2016. URL: https://arxiv.org/abs/1609.04693.

12 Olivier Laurent. Classical isomorphisms of types. *Mathematical Structures in Computer Science*, 15(5):969–1004, October 2005.

13 Laurent Regnier and Pawel Urzyczyn. Retractions of types with many atoms, 2002. URL: http://arxiv.org/abs/cs/0212005.

14 Mikael Rittri. Using types as search keys in function libraries. *Journal of Functional Programming*, 1(1):71–89, 1991.

15 Robert Seely. Linear logic, $\star$-autonomous categories and cofree coalgebras. *Contemporary mathematics*, 92, 1989.

16 Sergei Soloviev. The category of finite sets and cartesian closed categories. *Journal of Soviet Mathematics*, 22(3):1387–1400, 1983.

A Transformations of sequent calculus proofs in MALL

▶ **Definition 40.** *In the sequent calculus of MALL, we call* axiom-expansion *the rewriting system* $\xrightarrow{\eta}$ *described on Table 3.*

▶ **Definition 41.** *In the sequent calculus of MALL, we call* cut-elimination *the rewriting system* $\xrightarrow{\beta}$ *described on Tables 4 and 5 (up to commuting the two branches of a cut-rule).*

▶ **Definition 42.** *In the sequent calculus of MALL, we call* rule commutation *the equational theory* $=_c$ *described on Tables 6 and 7. This corresponds to rule commutations in cut-free MALL; in particular, in a* $\top - \otimes$ *permutation we assume the created or erased sub-proof to be cut-free.*

Table 3 Axiom-expansion in the sequent calculus of MALL.

$\parr - \otimes$	$\dfrac{}{\vdash A\otimes B, B^\perp \parr A^\perp}\,ax$	$\xrightarrow{\eta}$	$\dfrac{\dfrac{\dfrac{}{\vdash A^\perp, A}\,ax \quad \dfrac{}{\vdash B^\perp, B}\,ax}{\vdash A\otimes B, A^\perp, B^\perp}\,\otimes}{\vdash A\otimes B, B^\perp\parr A^\perp}\,\parr$
$\with - \oplus$	$\dfrac{}{\vdash A\oplus B, B^\perp \with A^\perp}\,ax$	$\xrightarrow{\eta}$	$\dfrac{\dfrac{\dfrac{}{\vdash B, B^\perp}\,ax}{\vdash A\oplus B, B^\perp}\,\oplus_2 \quad \dfrac{\dfrac{}{\vdash A, A^\perp}\,ax}{\vdash A\oplus B, A^\perp}\,\oplus_1}{\vdash A\oplus B, B^\perp\with A^\perp}\,\with$
$\bot - 1$	$\dfrac{}{\vdash 1,\bot}\,ax$	$\xrightarrow{\eta}$	$\dfrac{\dfrac{}{\vdash 1}\,1}{\vdash 1,\bot}\,\bot$
$\top - 0$	$\dfrac{}{\vdash 0,\top}\,ax$	$\xrightarrow{\eta}$	$\dfrac{}{\vdash 0,\top}\,\top$

Table 4 Cut-elimination in sequent calculus (key cases).

ax	$\dfrac{\dfrac{}{\vdash A^\perp, A}\,ax \quad \dfrac{\pi}{\vdash A,\Gamma}}{\vdash A,\Gamma}\,cut$	$\xrightarrow{\beta}$	$\dfrac{\pi}{\vdash A,\Gamma}$
$\parr - \otimes$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A,\Gamma}\quad\dfrac{\pi_2}{\vdash B,\Delta}}{\vdash A\otimes B,\Gamma,\Delta}\,\otimes \quad \dfrac{\dfrac{\pi_3}{\vdash B^\perp,A^\perp,\Sigma}}{\vdash B^\perp\parr A^\perp,\Sigma}\,\parr}{\vdash\Gamma,\Delta,\Sigma}\,cut$	$\xrightarrow{\beta}$	$\dfrac{\dfrac{\pi_1}{\vdash A,\Gamma}\quad \dfrac{\dfrac{\pi_2}{\vdash B,\Delta}\quad\dfrac{\pi_3}{\vdash B^\perp,A^\perp,\Sigma}}{\vdash A^\perp,\Delta,\Sigma}\,cut}{\vdash \Gamma,\Delta,\Sigma}\,cut$
$\with - \oplus_1$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1,\Gamma}\quad\dfrac{\pi_2}{\vdash A_2,\Gamma}}{\vdash A_1\with A_2,\Gamma}\,\with\quad\dfrac{\dfrac{\pi_3}{\vdash A_2^\perp,\Delta}}{\vdash A_2^\perp\oplus A_1^\perp,\Delta}\,\oplus_1}{\vdash\Gamma,\Delta}\,cut$	$\xrightarrow{\beta}$	$\dfrac{\dfrac{\pi_2}{\vdash A_2,\Gamma}\quad\dfrac{\pi_3}{\vdash A_2^\perp,\Delta}}{\vdash\Gamma,\Delta}\,cut$
$\with - \oplus_2$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1,\Gamma}\quad\dfrac{\pi_2}{\vdash A_2,\Gamma}}{\vdash A_1\with A_2,\Gamma}\,\with\quad\dfrac{\dfrac{\pi_3}{\vdash A_1^\perp,\Delta}}{\vdash A_2^\perp\oplus A_1^\perp,\Delta}\,\oplus_2}{\vdash\Gamma,\Delta}\,cut$	$\xrightarrow{\beta}$	$\dfrac{\dfrac{\pi_1}{\vdash A_1,\Gamma}\quad\dfrac{\pi_3}{\vdash A_1^\perp,\Delta}}{\vdash\Gamma,\Delta}\,cut$
$\bot - 1$	$\dfrac{\dfrac{}{\vdash 1}\,1\quad\dfrac{\dfrac{\pi}{\vdash\Gamma}}{\vdash\Gamma,\bot}\,\bot}{\vdash\Gamma}\,cut$	$\xrightarrow{\beta}$	$\dfrac{\pi}{\vdash\Gamma}$
(No $\top - 0$ key case as there are no rule for 0.)			

Table 5 Cut-elimination in sequent calculus (commutative cases).

$\parr - cut$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A,B,C,\Gamma}}{\vdash A,B\parr C,\Gamma}\parr\quad\dfrac{\pi_2}{\vdash A^\perp,\Delta}}{\vdash B\parr C,\Gamma,\Delta}\,\mathit{cut}$	$\xrightarrow{\beta}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A,B,C,\Gamma}\quad\dfrac{\pi_2}{\vdash A^\perp,\Delta}}{\vdash B,C,\Gamma,\Delta}\,\mathit{cut}}{\vdash B\parr C,\Gamma,\Delta}\parr$
$\otimes - cut - 1$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A,B,\Gamma}\quad\dfrac{\pi_2}{\vdash C,\Delta}}{\vdash A,B\otimes C,\Gamma,\Delta}\otimes\quad\dfrac{\pi_3}{\vdash A^\perp,\Sigma}}{\vdash B\otimes C,\Gamma,\Delta,\Sigma}\,\mathit{cut}$	$\xrightarrow{\beta}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A,B,\Gamma}\quad\dfrac{\pi_3}{\vdash A^\perp,\Sigma}}{\vdash B,\Gamma,\Sigma}\,\mathit{cut}\quad\dfrac{\pi_2}{\vdash C,\Delta}}{\vdash B\otimes C,\Gamma,\Delta,\Sigma}\otimes$
$\otimes - cut - 2$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash B,\Gamma}\quad\dfrac{\pi_2}{\vdash A,C,\Delta}}{\vdash A,B\otimes C,\Gamma,\Delta}\otimes\quad\dfrac{\pi_3}{\vdash A^\perp,\Sigma}}{\vdash B\otimes C,\Gamma,\Delta,\Sigma}\,\mathit{cut}$	$\xrightarrow{\beta}$	$\dfrac{\dfrac{\pi_1}{\vdash B,\Gamma}\quad\dfrac{\dfrac{\pi_2}{\vdash A,C,\Delta}\quad\dfrac{\pi_3}{\vdash A^\perp,\Sigma}}{\vdash C,\Delta,\Sigma}\,\mathit{cut}}{\vdash B\otimes C,\Gamma,\Delta,\Sigma}\otimes$
$\& - cut$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A,B,\Gamma}\quad\dfrac{\pi_2}{\vdash A,C,\Gamma}}{\vdash A,B\,\&\,C,\Gamma}\,\&\quad\dfrac{\pi_3}{\vdash A^\perp,\Delta}}{\vdash B\,\&\,C,\Gamma,\Delta}\,\mathit{cut}$	$\xrightarrow{\beta}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A,B,\Gamma}\quad\dfrac{\pi_3}{\vdash A^\perp,\Delta}}{\vdash B,\Gamma,\Delta}\,\mathit{cut}\quad\dfrac{\dfrac{\pi_2}{\vdash A,C,\Gamma}\quad\dfrac{\pi_3}{\vdash A^\perp,\Delta}}{\vdash C,\Gamma,\Delta}\,\mathit{cut}}{\vdash B\,\&\,C,\Gamma,\Delta}\,\&$
$\oplus_i - cut$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A,B_i,\Gamma}}{\vdash A,B_1\oplus B_2,\Gamma}\oplus_i\quad\dfrac{\pi_2}{\vdash A^\perp,\Delta}}{\vdash B_1\oplus B_2,\Gamma,\Delta}\,\mathit{cut}$	$\xrightarrow{\beta}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A,B_i,\Gamma}\quad\dfrac{\pi_2}{\vdash A^\perp,\Delta}}{\vdash B_i,\Gamma,\Delta}\,\mathit{cut}}{\vdash B_1\oplus B_2,\Gamma,\Delta}\oplus_i$
$\perp - cut$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A,\Gamma}}{\vdash A,\perp,\Gamma}\perp\quad\dfrac{\pi_2}{\vdash A^\perp,\Delta}}{\vdash \perp,\Gamma,\Delta}\,\mathit{cut}$	$\xrightarrow{\beta}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A,\Gamma}\quad\dfrac{\pi_2}{\vdash A^\perp,\Delta}}{\vdash \Gamma,\Delta}\,\mathit{cut}}{\vdash \perp,\Gamma,\Delta}\perp$
$\top - cut$	$\dfrac{\dfrac{}{\vdash A,\top,\Gamma}\top\quad\dfrac{\pi}{\vdash A^\perp,\Delta}}{\vdash \top,\Gamma,\Delta}\,\mathit{cut}$	$\xrightarrow{\beta}$	$\dfrac{}{\vdash \top,\Gamma,\Delta}\top$
$cut - cut$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A,B,\Gamma}\quad\dfrac{\pi_2}{\vdash B^\perp,\Delta}}{\vdash A,\Gamma,\Delta}\,\mathit{cut}\quad\dfrac{\pi_3}{\vdash A^\perp,\Sigma}}{\vdash \Gamma,\Delta,\Sigma}\,\mathit{cut}$	$\xrightarrow{\beta}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A,B,\Gamma}\quad\dfrac{\pi_3}{\vdash A^\perp,\Sigma}}{\vdash B,\Gamma,\Sigma}\,\mathit{cut}\quad\dfrac{\pi_2}{\vdash B^\perp,\Delta}}{\vdash \Gamma,\Delta,\Sigma}\,\mathit{cut}$
(No $ax - cut$ nor $1 - cut$ nor $0 - cut$ commutative cases as the ax and 1-rules have no context and there are no rule for 0.)			

Table 6 Rule commutations involving a unit rule.

$\dfrac{}{\vdash A_1\parr A_2,\top,\Gamma}\top$	$\xrightarrow{c^{\parr}_{\top}}$ $\xleftarrow[c^{\top}_{\parr}]{}$	$\dfrac{\dfrac{}{\vdash A_1,A_2,\top,\Gamma}\top}{\vdash A_1\parr A_2,\top,\Gamma}\parr$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1,A_2,\Gamma}}{\vdash A_1\parr A_2,\Gamma}\parr}{\vdash A_1\parr A_2,\perp,\Gamma}\perp$	$\xrightarrow{c^{\parr}_{\perp}}$ $\xleftarrow[c^{\perp}_{\parr}]{}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1,A_2,\Gamma}}{\vdash A_1,A_2,\perp,\Gamma}\perp}{\vdash A_1\parr A_2,\perp,\Gamma}\parr$
$\dfrac{}{\vdash A_1\otimes A_2,\top,\Gamma,\Delta}\top$	$\xrightarrow{c^{\otimes}_{\top}}$ $\xleftarrow[c^{\top}_{\otimes}]{}$	$\dfrac{\dfrac{}{\vdash A_1,\top,\Gamma}\top\quad\dfrac{\pi}{\vdash A_2,\Delta}}{\vdash A_1\otimes A_2,\top,\Gamma,\Delta}\otimes$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1,\Gamma}\quad\dfrac{\pi_2}{\vdash A_2,\Delta}}{\vdash A_1\otimes A_2,\Gamma,\Delta}\otimes}{\vdash A_1\otimes A_2,\perp,\Gamma,\Delta}\perp$	$\xrightarrow{c^{\otimes}_{\perp}}$ $\xleftarrow[c^{\perp}_{\otimes}]{}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1,\Gamma}}{\vdash A_1,\perp,\Gamma}\perp\quad\dfrac{\pi_2}{\vdash A_2,\Delta}}{\vdash A_1\otimes A_2,\perp,\Gamma,\Delta}\otimes$
$\dfrac{}{\vdash A_1\otimes A_2,\top,\Gamma,\Delta}\top$	$\xrightarrow{c^{\otimes}_{\top}}$ $\xleftarrow[c^{\top}_{\otimes}]{}$	$\dfrac{\dfrac{\pi}{\vdash A_1,\Gamma}\quad\dfrac{}{\vdash A_2,\top,\Delta}\top}{\vdash A_1\otimes A_2,\top,\Gamma,\Delta}\otimes$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1,\Gamma}\quad\dfrac{\pi_2}{\vdash A_2,\Delta}}{\vdash A_1\otimes A_2,\Gamma,\Delta}\otimes}{\vdash A_1\otimes A_2,\perp,\Gamma,\Delta}\perp$	$\xrightarrow{c^{\otimes}_{\perp}}$ $\xleftarrow[c^{\perp}_{\otimes}]{}$	$\dfrac{\dfrac{\pi_1}{\vdash A_1,\Gamma}\quad\dfrac{\dfrac{\pi_2}{\vdash A_2,\Delta}}{\vdash A_2,\perp,\Delta}\perp}{\vdash A_1\otimes A_2,\perp,\Gamma,\Delta}\otimes$
$\dfrac{}{\vdash A_1\,\&\,A_2,\top,\Gamma}\top$	$\xrightarrow{c^{\&}_{\top}}$ $\xleftarrow[c^{\top}_{\&}]{}$	$\dfrac{\dfrac{}{\vdash A_1,\top,\Gamma}\top\quad\dfrac{}{\vdash A_2,\top,\Gamma}\top}{\vdash A_1\,\&\,A_2,\top,\Gamma}\,\&$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1,\Gamma}\quad\dfrac{\pi_2}{\vdash A_2,\Gamma}}{\vdash A_1\,\&\,A_2,\Gamma}\,\&}{\vdash A_1\,\&\,A_2,\perp,\Gamma}\perp$	$\xrightarrow{c^{\&}_{\perp}}$ $\xleftarrow[c^{\perp}_{\&}]{}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1,\Gamma}}{\vdash A_1,\perp,\Gamma}\perp\quad\dfrac{\dfrac{\pi_2}{\vdash A_2,\Gamma}}{\vdash A_2,\perp,\Gamma}\perp}{\vdash A_1\,\&\,A_2,\perp,\Gamma}\,\&$
$\dfrac{}{\vdash A_1\oplus A_2,\top,\Gamma}\top$	$\xrightarrow{c^{\oplus_i}_{\top}}$ $\xleftarrow[c^{\top}_{\oplus_i}]{}$	$\dfrac{\dfrac{}{\vdash A_i,\top,\Gamma}\top}{\vdash A_1\oplus A_2,\top,\Gamma}\oplus_i$	$\dfrac{\dfrac{\dfrac{\pi}{\vdash A_i,\Gamma}}{\vdash A_1\oplus A_2,\Gamma}\oplus_i}{\vdash A_1\oplus A_2,\perp,\Gamma}\perp$	$\xrightarrow{c^{\oplus_i}_{\perp}}$ $\xleftarrow[c^{\perp}_{\oplus_i}]{}$	$\dfrac{\dfrac{\dfrac{\pi}{\vdash A_i,\Gamma}}{\vdash A_i,\perp,\Gamma}\perp}{\vdash A_1\oplus A_2,\perp,\Gamma}\oplus_i$
$\dfrac{}{\vdash \top_0,\top_1,\Gamma}\top_0$	$\xrightarrow{c^{\top}_{\top}}$	$\dfrac{}{\vdash \top_0,\top_1,\Gamma}\top_1$	$\dfrac{\dfrac{\dfrac{\pi}{\vdash \Gamma}}{\vdash \perp_0,\Gamma}\perp_0}{\vdash \perp_0,\perp_1,\Gamma}\perp_1$	$\xrightarrow{c^{\perp}_{\perp}}$	$\dfrac{\dfrac{\dfrac{\pi}{\vdash \Gamma}}{\vdash \perp_1,\Gamma}\perp_1}{\vdash \perp_0,\perp_1,\Gamma}\perp_0$
$\dfrac{}{\vdash \top,\perp,\Gamma}\top$	$\xrightarrow{c^{\perp}_{\top}}$ $\xleftarrow[c^{\top}_{\perp}]{}$	$\dfrac{\dfrac{}{\vdash \top,\Gamma}\top}{\vdash \top,\perp,\Gamma}\perp$			
(No commutation with ax, 1 nor 0 as the ax and 1-rules have no context and there are no rule for 0.)					

Table 7 Rule commutations not involving a unit rule.

$\dfrac{\dfrac{\dfrac{\pi}{\vdash A_1, A_2, B_1, B_2, \Gamma}}{\vdash A_1 \parr A_2, B_1, B_2, \Gamma}\,\parr}{\vdash A_1 \parr A_2, B_1 \parr B_2, \Gamma}\,\parr$	$\xrightarrow{c^{\parr}_{\parr}}$	$\dfrac{\dfrac{\dfrac{\pi}{\vdash A_1, A_2, B_1, B_2, \Gamma}}{\vdash A_1, A_2, B_1 \parr B_2, \Gamma}\,\parr}{\vdash A_1 \parr A_2, B_1 \parr B_2, \Gamma}\,\parr$
$\dfrac{\dfrac{\pi_1}{\vdash A_1, \Gamma} \quad \dfrac{\dfrac{\pi_2}{\vdash A_2, B_1, \Delta} \quad \dfrac{\pi_3}{\vdash B_2, \Sigma}}{\vdash A_2, B_1 \otimes B_2, \Delta, \Sigma}\,\otimes}{\vdash A_1 \otimes A_2, B_1 \otimes B_2, \Gamma, \Delta, \Sigma}\,\otimes$	$\xrightarrow{c^{\otimes}_{\otimes}}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, \Gamma} \quad \dfrac{\pi_2}{\vdash A_2, B_1, \Delta}}{\vdash A_1 \otimes A_2, B_1, \Gamma, \Delta}\,\otimes \quad \dfrac{\pi_3}{\vdash B_2, \Sigma}}{\vdash A_1 \otimes A_2, B_1 \otimes B_2, \Gamma, \Delta, \Sigma}\,\otimes$
$\dfrac{\dfrac{\pi_1}{\vdash A_1, \Gamma} \quad \dfrac{\dfrac{\pi_2}{\vdash B_1, \Delta} \quad \dfrac{\pi_3}{\vdash A_2, B_2, \Sigma}}{\vdash A_2, B_1 \otimes B_2, \Delta, \Sigma}\,\otimes}{\vdash A_1 \otimes A_2, B_1 \otimes B_2, \Gamma, \Delta, \Sigma}\,\otimes$	$\xrightarrow{c^{\otimes}_{\otimes}}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, \Gamma} \quad \dfrac{\pi_3}{\vdash A_2, B_2, \Sigma}}{\vdash A_1 \otimes A_2, B_2, \Gamma, \Sigma}\,\otimes \quad \dfrac{\pi_2}{\vdash B_1, \Delta}}{\vdash A_1 \otimes A_2, B_1 \otimes B_2, \Gamma, \Delta, \Sigma}\,\otimes$
$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, B_1, \Gamma} \quad \dfrac{\pi_2}{\vdash B_2, \Delta}}{\vdash A_1, B_1 \otimes B_2, \Gamma, \Delta}\,\otimes \quad \dfrac{\pi_3}{\vdash A_2, \Sigma}}{\vdash A_1 \otimes A_2, B_1 \otimes B_2, \Gamma, \Delta, \Sigma}\,\otimes$	$\xrightarrow{c^{\otimes}_{\otimes}}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, B_1, \Gamma} \quad \dfrac{\pi_3}{\vdash A_2, \Sigma}}{\vdash A_1 \otimes A_2, B_1, \Gamma, \Sigma}\,\otimes \quad \dfrac{\pi_2}{\vdash B_2, \Delta}}{\vdash A_1 \otimes A_2, B_1 \otimes B_2, \Gamma, \Delta, \Sigma}\,\otimes$
$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash B_1, \Gamma} \quad \dfrac{\pi_2}{\vdash A_1, B_2, \Delta}}{\vdash A_1, B_1 \otimes B_2, \Gamma, \Delta}\,\otimes \quad \dfrac{\pi_3}{\vdash A_2, \Sigma}}{\vdash A_1 \otimes A_2, B_1 \otimes B_2, \Gamma, \Delta, \Sigma}\,\otimes$	$\xrightarrow{c^{\otimes}_{\otimes}}$	$\dfrac{\dfrac{\pi_1}{\vdash B_1, \Gamma} \quad \dfrac{\dfrac{\pi_2}{\vdash A_1, B_2, \Delta} \quad \dfrac{\pi_3}{\vdash A_2, \Sigma}}{\vdash A_1 \otimes A_2, B_2, \Delta, \Sigma}\,\otimes}{\vdash A_1 \otimes A_2, B_1 \otimes B_2, \Gamma, \Delta, \Sigma}\,\otimes$
$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, B_1, \Gamma} \quad \dfrac{\pi_2}{\vdash A_2, B_1, \Gamma}}{\vdash A_1 \mathbin{\&} A_2, B_1, \Gamma}\,\& \quad \dfrac{\dfrac{\pi_3}{\vdash A_1, B_2, \Gamma} \quad \dfrac{\pi_4}{\vdash A_2, B_2, \Gamma}}{\vdash A_1 \mathbin{\&} A_2, B_2, \Gamma}\,\&}{\vdash A_1 \mathbin{\&} A_2, B_1 \mathbin{\&} B_2, \Gamma}\,\&$	$\xrightarrow{c^{\&}_{\&}}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, B_1, \Gamma} \quad \dfrac{\pi_3}{\vdash A_1, B_2, \Gamma}}{\vdash A_1, B_1 \mathbin{\&} B_2, \Gamma}\,\& \quad \dfrac{\dfrac{\pi_2}{\vdash A_2, B_1, \Gamma} \quad \dfrac{\pi_4}{\vdash A_2, B_2, \Gamma}}{\vdash A_2, B_1 \mathbin{\&} B_2, \Gamma}\,\&}{\vdash A_1 \mathbin{\&} A_2, B_1 \mathbin{\&} B_2, \Gamma}\,\&$
$\dfrac{\dfrac{\dfrac{\pi}{\vdash A_i, B_j, \Gamma}}{\vdash A_1 \oplus A_2, B_j, \Gamma}\,\oplus_i}{\vdash A_1 \oplus A_2, B_1 \oplus B_2, \Gamma}\,\oplus_j$	$\xrightarrow{c^{\oplus_i}_{\oplus_j}}$	$\dfrac{\dfrac{\dfrac{\pi}{\vdash A_i, B_j, \Gamma}}{\vdash A_i, B_1 \oplus B_2, \Gamma}\,\oplus_j}{\vdash A_1 \oplus A_2, B_1 \oplus B_2, \Gamma}\,\oplus_i$
$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, A_2, B_1, \Gamma}}{\vdash A_1 \parr A_2, B_1, \Gamma}\,\parr \quad \dfrac{\pi_2}{\vdash B_2, \Delta}}{\vdash A_1 \parr A_2, B_1 \otimes B_2, \Gamma, \Delta}\,\otimes$	$\underset{c^{\otimes}_{\parr}}{\overset{c^{\parr}_{\otimes}}{\rightleftarrows}}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, A_2, B_1, \Gamma} \quad \dfrac{\pi_2}{\vdash B_2, \Delta}}{\vdash A_1, A_2, B_1 \otimes B_2, \Gamma, \Delta}\,\otimes}{\vdash A_1 \parr A_2, B_1 \otimes B_2, \Gamma, \Delta}\,\parr$
$\dfrac{\dfrac{\pi_1}{\vdash B_1, \Gamma} \quad \dfrac{\dfrac{\pi_2}{\vdash A_1, A_2, B_2, \Delta}}{\vdash A_1 \parr A_2, B_2, \Delta}\,\parr}{\vdash A_1 \parr A_2, B_1 \otimes B_2, \Gamma, \Delta}\,\otimes$	$\underset{c^{\otimes}_{\parr}}{\overset{c^{\parr}_{\otimes}}{\rightleftarrows}}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash B_1, \Gamma} \quad \dfrac{\pi_2}{\vdash A_1, A_2, B_2, \Delta}}{\vdash A_1, A_2, B_1 \otimes B_2, \Gamma, \Delta}\,\otimes}{\vdash A_1 \parr A_2, B_1 \otimes B_2, \Gamma, \Delta}\,\parr$
$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, A_2, B_1, \Gamma}}{\vdash A_1 \parr A_2, B_1, \Gamma}\,\parr \quad \dfrac{\dfrac{\pi_1}{\vdash A_1, A_2, B_2, \Gamma}}{\vdash A_1 \parr A_2, B_2, \Gamma}\,\parr}{\vdash A_1 \parr A_2, B_1 \mathbin{\&} B_2, \Gamma}\,\&$	$\underset{c^{\&}_{\parr}}{\overset{c^{\parr}_{\&}}{\rightleftarrows}}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, A_2, B_1, \Gamma} \quad \dfrac{\pi_1}{\vdash A_1, A_2, B_2, \Gamma}}{\vdash A_1, A_2, B_1 \mathbin{\&} B_2, \Gamma}\,\&}{\vdash A_1 \parr A_2, B_1 \mathbin{\&} B_2, \Gamma}\,\parr$
$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, A_2, B_i, \Gamma}}{\vdash A_1 \parr A_2, B_i, \Gamma}\,\parr}{\vdash A_1 \parr A_2, B_1 \oplus B_2, \Gamma}\,\oplus_i$	$\underset{c^{\oplus_i}_{\parr}}{\overset{c^{\parr}_{\oplus_i}}{\rightleftarrows}}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, A_2, B_i, \Gamma}}{\vdash A_1, A_2, B_1 \oplus B_2, \Gamma}\,\oplus_i}{\vdash A_1 \parr A_2, B_1 \oplus B_2, \Gamma}\,\parr$
$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, \Gamma} \quad \dfrac{\pi_2}{\vdash A_2, B_1, \Delta}}{\vdash A_1 \otimes A_2, B_1, \Gamma, \Delta}\,\otimes \quad \dfrac{\dfrac{\pi_1}{\vdash A_1, \Gamma} \quad \dfrac{\pi_3}{\vdash A_2, B_2, \Delta}}{\vdash A_1 \otimes A_2, B_2, \Gamma, \Delta}\,\otimes}{\vdash A_1 \otimes A_2, B_1 \mathbin{\&} B_2, \Gamma, \Delta}\,\&$	$\underset{c^{\&}_{\otimes}}{\overset{c^{\otimes}_{\&}}{\rightleftarrows}}$	$\dfrac{\dfrac{\pi_1}{\vdash A_1, \Gamma} \quad \dfrac{\dfrac{\pi_2}{\vdash A_2, B_1, \Delta} \quad \dfrac{\pi_3}{\vdash A_2, B_2, \Delta}}{\vdash A_2, B_1 \mathbin{\&} B_2, \Delta}\,\&}{\vdash A_1 \otimes A_2, B_1 \mathbin{\&} B_2, \Gamma, \Delta}\,\otimes$
$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, B_1, \Gamma} \quad \dfrac{\pi_2}{\vdash A_2, \Delta}}{\vdash A_1 \otimes A_2, B_1, \Gamma, \Delta}\,\otimes \quad \dfrac{\dfrac{\pi_3}{\vdash A_1, B_2, \Gamma} \quad \dfrac{\pi_2}{\vdash A_2, \Delta}}{\vdash A_1 \otimes A_2, B_2, \Gamma, \Delta}\,\otimes}{\vdash A_1 \otimes A_2, B_1 \mathbin{\&} B_2, \Gamma, \Delta}\,\&$	$\underset{c^{\&}_{\otimes}}{\overset{c^{\otimes}_{\&}}{\rightleftarrows}}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, B_1, \Gamma} \quad \dfrac{\pi_3}{\vdash A_1, B_2, \Gamma}}{\vdash A_1, B_1 \mathbin{\&} B_2, \Gamma}\,\& \quad \dfrac{\pi_2}{\vdash A_2, \Delta}}{\vdash A_1 \otimes A_2, B_1 \mathbin{\&} B_2, \Gamma, \Delta}\,\otimes$
$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, B_i, \Gamma} \quad \dfrac{\pi_2}{\vdash A_2, \Delta}}{\vdash A_1 \otimes A_2, B_i, \Gamma, \Delta}\,\otimes}{\vdash A_1 \otimes A_2, B_1 \oplus B_2, \Gamma, \Delta}\,\oplus_i$	$\underset{c^{\oplus_i}_{\otimes}}{\overset{c^{\otimes}_{\oplus_i}}{\rightleftarrows}}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, B_i, \Gamma}}{\vdash A_1, B_1 \oplus B_2, \Gamma}\,\oplus_i \quad \dfrac{\pi_2}{\vdash A_2, \Delta}}{\vdash A_1 \otimes A_2, B_1 \oplus B_2, \Gamma, \Delta}\,\otimes$
$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, \Gamma} \quad \dfrac{\pi_2}{\vdash A_2, B_i, \Delta}}{\vdash A_1 \otimes A_2, B_i, \Gamma, \Delta}\,\otimes}{\vdash A_1 \otimes A_2, B_1 \oplus B_2, \Gamma, \Delta}\,\oplus_i$	$\underset{c^{\oplus_i}_{\otimes}}{\overset{c^{\otimes}_{\oplus_i}}{\rightleftarrows}}$	$\dfrac{\dfrac{\pi_1}{\vdash A_1, B_i, \Gamma} \quad \dfrac{\dfrac{\pi_2}{\vdash A_2, \Delta}}{\vdash A_2, B_1 \oplus B_2, \Delta}\,\oplus_i}{\vdash A_1 \otimes A_2, B_1 \oplus B_2, \Gamma, \Delta}\,\otimes$
$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, B_i, \Gamma} \quad \dfrac{\pi_2}{\vdash A_2, B_i, \Gamma}}{\vdash A_1 \mathbin{\&} A_2, B_i, \Gamma}\,\&}{\vdash A_1 \mathbin{\&} A_2, B_1 \oplus B_2, \Gamma}\,\oplus_i$	$\underset{c^{\oplus_i}_{\&}}{\overset{c^{\&}_{\oplus_i}}{\rightleftarrows}}$	$\dfrac{\dfrac{\dfrac{\pi_1}{\vdash A_1, B_i, \Gamma}}{\vdash A_1, B_1 \oplus B_2, \Gamma}\,\oplus_i \quad \dfrac{\dfrac{\pi_2}{\vdash A_2, B_i, \Gamma}}{\vdash A_2, B_1 \oplus B_2, \Gamma}\,\oplus_i}{\vdash A_1 \mathbin{\&} A_2, B_1 \oplus B_2, \Gamma}\,\&$
(No commutation with ax, 1 nor 0 as the ax and 1-rule have no context and there are no rule for 0.)		

Cyclic Proofs for Arithmetical Inductive Definitions

Anupam Das
University of Birmingham, UK

Lukas Melgaard
University of Birmingham, UK

Abstract

We investigate the cyclic proof theory of extensions of Peano Arithmetic by (finitely iterated) inductive definitions. Such theories are essential to proof theoretic analyses of certain "impredicative" theories; moreover, our cyclic systems naturally subsume Simpson's Cyclic Arithmetic.

Our main result is that cyclic and inductive systems for arithmetical inductive definitions are equally powerful. We conduct a metamathematical argument, formalising the soundness of cyclic proofs within second-order arithmetic by a form of induction on closure ordinals, thence appealing to conservativity results. This approach is inspired by those of Simpson and Das for Cyclic Arithmetic, however we must further address a difficulty: the closure ordinals of our inductive definitions (around Church-Kleene) far exceed the proof theoretic ordinal of the appropriate metatheory (around Bachmann-Howard), so explicit induction on their notations is not possible. For this reason, we rather rely on formalisation of the theory of (recursive) ordinals within second-order arithmetic.

2012 ACM Subject Classification Theory of computation → Proof theory

Keywords and phrases cyclic proofs, inductive definitions, arithmetic, fixed points, proof theory

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.27

Related Version *Full Version*: https://arxiv.org/abs/2306.08535

Funding This work was supported by a UKRI Future Leaders Fellowship, "Structure vs. Invariants in Proofs", project reference MR/S035540/1.

Acknowledgements The authors would like to thank Graham Leigh and Colin Riba for several interesting conversations about (arithmetical) inductive definitions.

1 Introduction

Cyclic proof theory studies "proofs" whose underlying dependency graph may not be well-founded, but are nonetheless regular. Soundness for such systems is controlled by an appropriate "correctness criterion", usually an ω-regular property on infinite branches, defined at the level of formula ancestry. Cyclic proofs are a relatively recent development in proof theory (and related areas), with origins in seminal work of Niwiński and Walukiewicz for the modal μ-calculus [18]. Inspired by that work, Brotherston and Simpson studied the extension of first-order logic by (ordinary) inductive definitions [7, 9, 10]. More recently, Simpson has proposed *Cyclic Arithmetic* (CA), an adaptation of usual Peano Arithmetic (PA) to the cyclic setting [21].

One of the recurring themes of cyclic proof theory is the capacity for non-wellfounded reasoning to simulate inductive arguments with apparently simpler (and often *analytic*) invariants. Indeed this difference in expressivity has been made formal in various settings [3, 6] and has been exploited in implementations [8, 20, 26, 27]. Within the setting of arithmetic, we have a more nuanced picture: while Simpson showed that CA and PA are equivalent as theories [21], Das has shown that indeed the logical complexity of invariants required in CA is indeed strictly simpler than in PA [11]. These arguments typically follow a

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 27; pp. 27:1–27:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

metamathematical approach, formalising the soundness argument of cyclic proofs themselves within arithmetic and relying on a *reflection* principle (though there are alternative approaches too, cf. [4, 5]). Due to the infinitary nature of non-wellfounded proofs and the complexity of correctness, such arguments require a further detour through the reverse mathematics of ω-automata theory, cf. [14, 15].

In this work we somewhat bridge the aforementioned traditions in the μ-calculus, first-order logic with inductive definitions, and arithmetic. In particular we present a cyclic proof system $\mathsf{CID}_{<\omega}$ over the language of (finitely iterated) arithmetical inductive definitions: the closure of the language of arithmetic under formation of (non-parametrised) fixed points. Such languages form the basis of important systems in proof theory, in particular $\mathsf{ID}_{<\omega}$, which allows for an ordinal analysis of impredicative second-order theories such as Π^1_1-CA_0. Our cyclic system $\mathsf{CID}_{<\omega}$ over this language is essentially recovered by directly importing analogous definitions from the μ-calculus and first-order inductive definitions.

Our main result is the equivalence between $\mathsf{CID}_{<\omega}$ and its inductive counterpart $\mathsf{ID}_{<\omega}$. While subsuming inductive proofs by cyclic proofs is a routine construction, the converse direction constitutes a generalisation of ideas from the setting of arithmetic, cf. [21, 11]. One particular nuance here is that the soundness of cyclic proofs with forms of inductive definitions typically reduces to a form of induction on the corresponding *closure ordinals*. For the setting of even unnested inductive definitions, ID_1, closure ordinals already exhaust all the recursive ordinals (up to Church-Kleene, ω_1^{CK}). On the other hand the proof theoretical ordinal of ID_1 is only the Bachmann-Howard ordinal, so we cannot formalise the required induction principle on explicit ordinal notations. Instead we rely on a (known) formalisation of (recursive) ordinal theory *within* appropriate fragments of second-order arithmetic.

This paper is structured as follows. In Section 2 we recall the syntax and semantics of first-order logic with inductive definitions, as well as the Knaster-Tarski fixed point theorem specialised to $\mathcal{P}(\mathbb{N})$. In Section 3 we recall PA and $\mathsf{ID}_{<\omega}$, recast in the sequent calculus to facilitate the definition of $\mathsf{CID}_{<\omega}$. The latter is presented in Section 4 where we also show its simulation of $\mathsf{ID}_{<\omega}$. In Section 5 we show that the system $\mathsf{CID}_{<\omega}$ is indeed sound for the standard model. In Sections 6 and 7 we formalise aspects of inductive definitions, truth, order theory and fixed point theory within suitable fragments of second-order arithmetic. Finally in Section 8 we present the converse simulation, from $\mathsf{CID}_{<\omega}$ to $\mathsf{ID}_{<\omega}$, by essentially arithmetising the soundness argument of Section 5.

Due to space constraints, most proofs and auxiliary material are omitted.

2 Syntax and semantics of arithmetical inductive definitions

2.1 First-order logic (with equality)

In this work we shall work in predicate logic over various languages, written $\mathcal{L}, \mathcal{L}'$ etc. We write x, y etc. for (first-order) variables and s, t etc. for terms, and φ, ψ etc. for formulas (including equality). For later convenience, we shall write formulas in *De Morgan normal form*, with negations only in front of atomic formulas. I.e. formulas are generated from "atomic" formulas $P(\vec{t}), \neg P(\vec{t}), s = t, \neg s = t$ under $\vee, \wedge, \exists, \forall$. From here we use standard abbreviations for negation and other connectives.

In order to interpret "inductive definitions" in the next section, it will be useful to consider a variation of usual Henkin semantics that interprets (relativised) formulas as operators on a structure. Given a language $\mathcal{L}$, We write $\mathcal{L}(X)$ for the extension of $\mathcal{L}$ by the fresh predicate symbol X. For instance formulas of $\mathcal{L}(X)$, where X is unary, include all those of $\mathcal{L}$, new "atomic" formulas of the form $X(t)$ and $\neg X(t)$, and are closed under usual logical operations.

Fix a language $\mathcal{L}$ and $\mathcal{L}$-structure $\mathfrak{M}$ with domain M. Let X be a fresh k-ary predicate symbol and let $\vec{x} = x_1, \dots, x_l$ be distinguished variables. Temporarily expand $\mathcal{L}$ to include each $a \in M$ as a constant symbol and each $A \subseteq M^k$ as a predicate symbol and fix $a^{\mathfrak{M}} := a$ and $A^{\mathfrak{M}} := A$. We interpret formulas $\varphi(X, \vec{x})$ of $\mathcal{L}(X)$ as functions $\varphi^{\mathfrak{M}} : \mathcal{P}(M^k) \to \mathcal{P}(M^l)$ by setting $\vec{a} \in \varphi^{\mathfrak{M}}(A)$ just if $\mathfrak{M} \models \varphi[A/X][\vec{a}/\vec{x}]$.

Let us call a formula $\varphi(X)$ *positive* in X if it has no subformula of the form $\neg X(\vec{t})$. The following result motivates the "positive inductive definitions" we consider in the next section:

▶ **Proposition 2.1** (Positivity implies monotonicity). *Let $\mathcal{L}$, $\mathfrak{M}$, X, $\vec{x}$ be as above. If φ, a formula of $\mathcal{L}(X)$, is positive in X then $\varphi^{\mathfrak{M}}$ is monotone: $A \subseteq B \implies \varphi^{\mathfrak{M}}(A) \subseteq \varphi^{\mathfrak{M}}(B)$.*

Proof idea. By straightforward induction on the structure of φ. ◀

2.2 Languages of arithmetic and (finitely iterated) inductive definitions

The *language of arithmetic (with inequality)* is $\mathcal{L}_A := \{0, \mathsf{s}, +, \times, <\}$. Here, as usual, 0 is a constant symbol (i.e. a 0-ary function symbol), s is a unary function symbol, $+$ and $\times$ are binary function symbols, and $<$ is a binary relation symbol.

Throughout this paper we shall work with (certain extensions of) $\mathcal{L}_A$:

▶ **Definition 2.2** ((Finitely iterated) inductive definitions). *$\mathcal{L}_{<\omega}$ is the smallest language containing $\mathcal{L}_A$ and closed under:*

- *if φ is a formula of $\mathcal{L}_{<\omega}(X)$ positive in X, for X a fresh unary predicate symbol, and x is a distinguished variable, then $I_{\varphi,X,x}$ is a unary predicate symbol of $\mathcal{L}_{<\omega}$.*

Note that we only take the case that X is unary above since we can always code k-ary predicates using unary ones within arithmetic. When X, x are clear from context, we shall simply write I_φ instead of $I_{\varphi,X,x}$. We shall also frequently suppress free variables and parameters (i.e. predicate symbols), e.g. writing interchangably $\varphi(X, x)$ and φ, when it is convenient and unambiguous.

Let us introduce some running examples for this work.

▶ **Example 2.3** (Naturals, evens and odds). We define the following formulas of $\mathcal{L}_A(X)$:

- $n(X, x) := x = 0 \vee \exists y(X(y) \wedge x = \mathsf{s}y)$.
- $e(X, x) := x = 0 \vee \exists y(X(y) \wedge x = \mathsf{ss}y)$.
- $o(X, x) := x = 1 \wedge \exists y(X(y) \wedge x = \mathsf{ss}y)$ (where $1 := \mathsf{s}0$).

By definition $\mathcal{L}_{<\omega}$ contains the symbols $N := I_n$, $E := I_e$ and $O := I_o$. Now, writing,

- $m(X, x) := e(X, x) \vee (\forall y(E(y) \to X(y)) \wedge x = 1)$

we also have that $M := I_m$ is a symbol of $\mathcal{L}_{<\omega}$, by the closure property of the language.

All our theories are interpreted by the "standard model" of arithmetic $\mathfrak{N} = (0, \mathsf{s}, +, \times, <)$, which we extend to a $\mathcal{L}_{<\omega}$-structure by:

- $I^{\mathfrak{N}}_{\varphi,X,x} := \bigcap\{A \subseteq \mathbb{N} : \varphi^{\mathfrak{N}}(A) \subseteq A\}$

2.3 On Knaster-Tarski: inductive definitions as fixed points

We conclude this section by making some comments about the interpretation of inductive definitions as fixed points. Let us first state a version of the well-known Knaster-Tarski theorem specialised to the setting at hand:

▶ **Proposition 2.4** (Knaster-Tarski on $\mathcal{P}(\mathbb{N})$). *Let $F : \mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$ be monotone, i.e. $A \subseteq B \subseteq \mathbb{N} \implies F(A) \subseteq F(B)$. Then F has a least fixed point μF and a greatest fixed point νF. Moreover, we have: $\mu F = \bigcap\{A \subseteq \mathbb{N} : F(A) \subseteq A\}$ and $\nu F = \bigcup\{A \subseteq \mathbb{N} : A \subseteq F(A)\}$.*

We shall henceforth adopt the notation of the theorem above, writing μF and νF for the least and greatest fixed point of an operator F, when they exist.

In light of Proposition 2.1 we immediately have:

▶ **Corollary 2.5.** *$I_\varphi^{\mathfrak{N}} = \mu\,\varphi^{\mathfrak{N}}$, i.e. $I_\varphi^{\mathfrak{N}}$ is the least fixed point of $\varphi^{\mathfrak{N}} : \mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$.*

▶ **Example 2.6** (Naturals, evens and odds: interpretation)**.** Revisiting Example 2.3 we have:

- $N^{\mathfrak{N}} = \mathbb{N}$
- $E^{\mathfrak{N}} = \mathbb{E} := \{2n : n \in \mathbb{N}\}$
- $O^{\mathfrak{N}} = \mathbb{O} := \{2n+1 : n \in \mathbb{N}\}$

It turns out that also $M^{\mathfrak{N}} = \mathbb{N}$. While this is readily verifiable with the current definitions, we shall delay a justification of this until we have built up some more technology.

Let us point out that the syntax of $\mathcal{L}_{<\omega}$ also allows the formation of *greatest* fixed points, by appealing to duality via negation, but we omit such considerations here.

It is well known that least (and greatest) fixed points can be approximated "from below" (and "from above", respectively) via the notion of *(ordinal) approximant.* For any $F : \mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$, let us define by transfinite induction,

$$\begin{aligned} F^0(A) &:= A \\ F^{\alpha+1}(A) &:= F(F^\alpha(A)) \\ F^\lambda(A) &:= \bigcup_{\alpha<\lambda} F^\alpha(A) \quad \text{if } \lambda \text{ is a limit ordinal} \end{aligned} \tag{1}$$

By appealing to the transfinite pigeonhole principle we have:

▶ **Proposition 2.7.** *For $F : \mathcal{P}(\mathbb{N}) \to \mathcal{P}(\mathbb{N})$ monotone, there is an ordinal α s.t. $\mu F = F^\alpha(\varnothing)$.*

Indeed we may assume that such α is *countable* and, by the well-ordering principle, there is indeed a *least* such α satisfying the proposition above.

▶ **Example 2.8** (Naturals, evens and odds: closure ordinals)**.** Revisiting Example 2.3 again, it is not hard to see that the approximants of $n^{\mathfrak{N}}, e^{\mathfrak{N}}, o^{\mathfrak{N}}$ are respectively:

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl} (n^{\mathfrak{N}})^0(\varnothing) &=& \varnothing & (e^{\mathfrak{N}})^0(\varnothing) &=& \varnothing & (o^{\mathfrak{N}})^0(\varnothing) &=& \varnothing \\ (n^{\mathfrak{N}})^1(\varnothing) &=& \{0\} & (e^{\mathfrak{N}})^1(\varnothing) &=& \{0\} & (o^{\mathfrak{N}})^1(\varnothing) &=& \{1\} \\ (n^{\mathfrak{N}})^2(\varnothing) &=& \{0,1\} & (e^{\mathfrak{N}})^2(\varnothing) &=& \{0,2\} & (o^{\mathfrak{N}})^2(\varnothing) &=& \{1,3\} \\ && \vdots &&& \vdots &&& \vdots \\ (n^{\mathfrak{N}})^\omega(\varnothing) &=& \mathbb{N} & (e^{\mathfrak{N}})^\omega(\varnothing) &=& \mathbb{E} & (o^{\mathfrak{N}})^\omega(\varnothing) &=& \mathbb{O} \end{array}$$

Note that for each of these operators we reached the (least) fixed point for the first time at stage ω. We say that ω is the *closure ordinal* of these operators.

Now, returning to the formula $m(X, x)$, let us finally compute its least fixed point in $\mathfrak{N}$ by the method of approximants:

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl} (m^{\mathfrak{N}})^0(\varnothing) &=& \varnothing & (m^{\mathfrak{N}})^\omega(\varnothing) &=& \mathbb{E} & (m^{\mathfrak{N}})^{\omega 2}(\varnothing) &=& \mathbb{E} \cup \mathbb{O} = \mathbb{N} \\ (m^{\mathfrak{N}})^1(\varnothing) &=& \{0\} & (m^{\mathfrak{N}})^{\omega+1}(\varnothing) &=& \mathbb{E} \cup \{1\} &&& \\ (m^{\mathfrak{N}})^2(\varnothing) &=& \{0,2\} & (m^{\mathfrak{N}})^{\omega+2}(\varnothing) &=& \mathbb{E} \cup \{1,3\} &&& \\ && \vdots &&& \vdots &&& \end{array}$$

Thus indeed $I_m^{\mathfrak{N}} = \mathbb{N}$, but this time with closure ordinal $\omega 2$.

$$\mathsf{id}\,\frac{}{\Gamma, \varphi \Rightarrow \Delta, \varphi} \qquad \mathsf{w}\,\frac{\Gamma \Rightarrow \Delta}{\Gamma, \Gamma' \Rightarrow \Delta, \Delta'} \qquad \theta\,\frac{\Gamma \Rightarrow \Delta}{\theta(\Gamma) \Rightarrow \theta(\Delta)} \qquad \mathsf{cut}\,\frac{\Gamma \Rightarrow \Delta, \varphi \quad \Gamma, \varphi \Rightarrow \Delta}{\Gamma \Rightarrow \Delta}$$

$$\neg\text{-}l\,\frac{\Gamma \Rightarrow \Delta, \chi}{\Gamma, \neg\chi \Rightarrow \Delta} \qquad \vee\text{-}l\,\frac{\Gamma, \varphi \Rightarrow \Delta \quad \Gamma, \psi \Rightarrow \Delta}{\Gamma, \varphi \vee \psi \Rightarrow \Delta} \qquad \vee\text{-}r\,\frac{\Gamma \Rightarrow \Delta, \varphi, \psi}{\Gamma \Rightarrow \Delta, \varphi \vee \psi}$$

$$\neg\text{-}r\,\frac{\Gamma, \chi \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg\chi} \qquad \wedge\text{-}r\,\frac{\Gamma \Rightarrow \Delta, \varphi \quad \Gamma \Rightarrow \Delta, \psi}{\Gamma \Rightarrow \Delta, \varphi \wedge \psi} \qquad \wedge\text{-}l\,\frac{\Gamma, \varphi, \psi \Rightarrow \Delta}{\Gamma, \varphi \wedge \psi \Rightarrow \Delta}$$

$$\forall\text{-}r\,\frac{\Gamma \Rightarrow \Delta, \varphi[y/x]}{\Gamma \Rightarrow \Delta, \forall x \varphi}\, y \text{ fresh} \qquad \forall\text{-}l\,\frac{\Gamma, \varphi(t) \Rightarrow \Delta}{\Gamma, \forall x \varphi(x) \Rightarrow \Delta}$$

$$\exists\text{-}l\,\frac{\Gamma, \varphi[y/x] \Rightarrow \Delta}{\Gamma, \exists x \varphi \Rightarrow \Delta}\, y \text{ fresh} \qquad \exists\text{-}r\,\frac{\Gamma \Rightarrow \Delta, \varphi(t)}{\Gamma \Rightarrow \Delta, \exists x \varphi(x)}$$

$$=\text{-}l\,\frac{\Gamma(s,t) \Rightarrow \Delta(s,t)}{\Gamma(t,s), s = t \Rightarrow \Delta(t,s)} \qquad =\text{-}r\,\frac{}{\Gamma \Rightarrow \Delta, t = t}$$

Figure 1 The sequent calculus $\mathsf{LK}_=$ for first-order logic with equality. θ is always a substitution, i.e. a map from variables to terms, extended to formulas and cedents in the expected way. χ is always an atomic formula $P(\vec{t})$ or $s = t$.

3 Arithmetical theories of inductive definitions

Thusfar we have only considered the language of arithmetic and inductive definitions ("syntax") and structures over these languages ("semantics"). We shall now introduce *theories* over these languages, in particular setting them up within a *sequent calculus* system, in order to facilitate the definition of the non-wellfounded and cyclic systems we introduce later.

▶ **Definition 3.1** (Sequent calculus for PA). *A* sequent *is an expression* $\Gamma \Rightarrow \Delta$ *where* Γ *and* Δ *are sets of formulas (sometimes called* cedents*).*[1] *The calculus* $\mathsf{LK}_=$ *for first-order logic with equality and substitution is given in Figure 1.*

The sequent calculus for PA *extends* $\mathsf{LK}_=$ *by initial sequents for all axioms of Robinson Arithmetic* Q*, as well as the induction rule:*

$$\mathsf{ind}\,\frac{\Gamma \Rightarrow \Delta, \varphi(0) \quad \Gamma, \varphi(y) \Rightarrow \Delta, \varphi(\mathsf{s}y)}{\Gamma \Rightarrow \Delta, \varphi(t)}\, y \textit{ fresh}$$

We will present some examples of proofs shortly, but first let us develop the implementation of the first-order theories we consider within the sequent calculus.

3.1 Theory of (finitely iterated) inductive definitions

$\mathsf{ID}_{<\omega}$ is a $\mathcal{L}_{<\omega}$-theory that extends PA by (the universal closures of):[2]

- (Pre-fixed) $\forall x(\varphi(I_\varphi, x) \to I_\varphi(x))$
- (Least) $\forall x(\varphi(\psi, x) \to \psi(x)) \to \forall x(I_\varphi(x) \to \psi(x))$

for all formulas $\varphi(X, x)$ positive in X.

[1] The symbol $\Rightarrow$ is just a syntactic delimiter, but is suggestive of the semantic interpretation of sequents.
[2] Formally, we include instances of the induction schema for all formulas φ in the extended language too.

Note that, while the first axiom states that I_φ is a *pre-fixed point* of $\varphi(-)$, the second axiom (schema) states that I_φ is least among the (arithmetically definable) pre-fixed points. As before, we implement this theory within the sequent calculus:

▶ **Definition 3.2** (Sequent calculus for $\mathsf{ID}_{<\omega}$). *The sequent caclulus for* $\mathsf{ID}_{<\omega}$ *extends that for* PA *by the rules:*

$$
{\scriptstyle I_\varphi\text{-}l}\ \dfrac{\Gamma, \varphi(I_\varphi, t) \Rightarrow \Delta}{\Gamma, I_\varphi(t) \Rightarrow \Delta}
\qquad\qquad
{\scriptstyle I_\varphi\text{-}r}\ \dfrac{\Gamma \Rightarrow \Delta, \varphi(I_\varphi, t)}{\Gamma \Rightarrow \Delta, I_\varphi(t)}
\tag{2}
$$

$$
{\scriptstyle \mathsf{ind}(\varphi)}\ \dfrac{\Gamma, \varphi(\psi, y) \Rightarrow \Delta, \psi(y) \quad \Gamma, \psi(t) \Rightarrow \Delta}{\Gamma, I_\varphi(t) \Rightarrow \Delta}\ \textit{y fresh}
\tag{3}
$$

3.2 Examples

In this subsection we consider some examples of sequent proofs for $\mathsf{ID}_{<\omega}$.

Note that the I_φ-r and $\mathsf{ind}(\varphi)$ rules correspond respectively to the axioms we gave for $\mathsf{ID}_{<\omega}$. The I_φ-l rule, morally stating that I_φ is a *post-fixed point* of $\varphi(-)$, does not correspond to any of the axioms. In fact we may consider it a form of "syntactic sugar" that will be useful for defining our cyclic systems later:

▶ **Example 3.3** (Post-fixed point). We can derive the I_φ-l rule from the other two as follows:

$$
{\scriptstyle \mathsf{ind}(\varphi)}\ \dfrac{
{\scriptstyle \varphi}\ \dfrac{
{\scriptstyle I_\varphi\text{-}r}\ \dfrac{
{\scriptstyle \mathsf{id}}\ \dfrac{}{\varphi(I_\varphi, y) \Rightarrow \varphi(I_\varphi, y)}
}{\varphi(I_\varphi, y) \Rightarrow I_\varphi(y)}
}{\varphi(\varphi(I_\varphi), y) \Rightarrow \varphi(I_\varphi, y)}
\quad \Gamma, \varphi(I_\varphi, t) \Rightarrow \Delta
}{\Gamma, I_\varphi(t) \Rightarrow \Delta}
$$

where the derivation marked φ ("functoriality") is obtained by structural induction on φ.

▶ **Example 3.4** (Subsuming numerical induction). Recalling the inductive predicate N from Example 2.3, the usual induction rule of PA is an immediate consequence of $\forall x N(x)$:

$$
{\scriptstyle \forall x N(x)}\ \dfrac{
{\scriptstyle \mathsf{ind}(n)}\ \dfrac{
{\scriptstyle \exists,\vee}\ \dfrac{
{\scriptstyle =}\ \dfrac{\Gamma \Rightarrow \Delta, \varphi(0)}{z = 0, \Gamma \Rightarrow \Delta, \varphi(z)}
\quad
{\scriptstyle =}\ \dfrac{\Gamma, \varphi(y) \Rightarrow \Delta, \varphi(\mathsf{s}y)}{z = \mathsf{s}y, \Gamma, \varphi(y) \Rightarrow \Delta, \varphi(z)}
}{\Gamma, n(\varphi, z) \Rightarrow \Delta, \varphi(z)}
\quad
{\scriptstyle \mathsf{id}}\ \dfrac{}{\varphi(t) \Rightarrow \varphi(t)}
}{\Gamma, N(t) \Rightarrow \Delta, \varphi(t)}
}{\Gamma \Rightarrow \Delta, \varphi(t)}
$$

4 Cyclic proofs for the theory of (finitely iterated) inductive definitions

In this section we introduce our "cyclic" version of the theory $\mathsf{ID}_{<\omega}$, based on ideas from the modal μ-calculus [18, 25, 2] and calculi of first-order logic with inductive definitions [7, 9, 10].

4.1 Non-wellfounded and cyclic proofs

The "non-wellfounded derivations" we consider will be potentially infinite proofs (of height $\leq \omega$) generated coinductively from the rules of the calculus. More formally:

▶ **Definition 4.1** (Preproofs). *A* (infinite, binary) tree *is a prefix-closed (potentially infinite) subset of* $\{0,1\}^*$. *A preproof* π *in a system* L *is a map from a tree* T_π *(the* support *of* π*) to inference steps of* L *such that, whenever* $\pi(v)$ *has premisses* $S_1, \dots, S_n$*,* v *has precisely* n *children*[3] $v_1, \dots, v_n \in T_\pi$ *where* $\pi(v_1), \dots, \pi(v_n)$ *have conclusions* $S_1, \dots, S_n$ *respectively.*

Given some $u \in T_\pi$*, we write* π_u *for the preproof with support* $T_{\pi_u} = \{v : uv \in T_\pi\}$ *given by* $\pi_u(v) := \pi(uv)$*. We call such* π_u *a* sub-preproof *of* π*. If* π *has only finitely many sub-preproofs, we call it* regular *or* cyclic*.*

Regular preproofs π can be represented as a finite (possibly cyclic) graph in the expected way, by simply quotienting T_π by the relation $\sim \subseteq T_\pi \times T_\pi$ given by $u \sim v$ if $\pi_u = \pi_v$. Let us now set up our principal system of interest:

▶ **Definition 4.2** (Rules for preproofs). *The system* $\mathsf{LID}^-_{<\omega}$ *extends* $\mathsf{LK}_=$ *by:*

- *initial sequents* $\Rightarrow \varphi$ *for each axiom* φ *of* Q*; and,*
- *the rules* I_φ*-l and* I_φ*-r from* (2)*; and,*
- *the following additional rule:* $N\,\dfrac{}{\Gamma \Rightarrow \Delta, N(t)}$

The "$-$" superscript in $\mathsf{LID}^-_{<\omega}$ indicates that we do not include the $\mathsf{ind}(\varphi)$ rules in this system. Note in the definition above that, in light of Example 3.4, we have chosen to simplify our system by omitting an explicit rule for numerical induction and instead simply including a rule that insists that our domain consists only of natural numbers. This streamlines the resulting definition of "progressing trace":

▶ **Definition 4.3** (Traces and progress). *Fix a* $\mathsf{LID}^-_{<\omega}$*-preproof* π *and* $(v_i)_{i\in\omega}$ *an infinite branch along* T_π*. A* trace *along* $(v_i)_{i\in\omega}$ *is a sequence of formulas* $(\varphi_i)_{i\geq k}$*, with each* φ_i *occurring on the LHS of* $\pi(v_i)$*, such that for all* $i \geq k$*:*

- $\pi(v_i)$ *is not a substitution step and* $\varphi_{i+1} = \varphi_i$*; or,*
- $\pi(v_i)$ *is a* θ*-substitution step and* $\theta(\varphi_{i+1}) = \varphi_i$*; or,*
- $\pi(v_i)$ *is a* $=$*-l step with respect to* $s = t$ *and, for some* $\psi(x,y)$*, we have* $\varphi_{i+1} = \psi(s,t)$ *and* $\varphi_i = \psi(t,s)$*; or,*
- φ_i *is the principal formula of* $\pi(v_i)$ *and* φ_{i+1} *is auxiliary.*

We say that φ_{k+1} *is an* immediate ancestor *of* φ_k *if they extend to some trace* $(\varphi_i)_{i\geq k}$*. A trace* $(\varphi_i)_{i\geq k}$ *is* progressing *if it is principal infinitely often.*

▶ **Definition 4.4** (Non-wellfounded proofs). *A (non-wellfounded)* $\mathsf{LID}^-_{<\omega}$-proof *is a* $\mathsf{LID}^-_{<\omega}$*-preproof* π *for which each infinite branch has a progressing trace. We also say that* π *is progressing in this case. If* π *is regular, we call it a* cyclic proof*.*

We write $\mathsf{LID}^-_{<\omega} \vdash_{\mathrm{nwf}} \varphi$ *or* $\mathsf{LID}^-_{<\omega} \vdash_{\mathrm{cyc}} \varphi$ *if there is a non-wellfounded or cyclic, respectively,* $\mathsf{LID}^-_{<\omega}$*-proof of* φ*. We write* $\mathsf{CID}_{<\omega}$ *for the class of cyclic* $\mathsf{LID}^-_{<\omega}$*-proofs.*

Many of the basic results and features of non-wellfounded and cyclic proofs for arithmetic from [21, 11] are present also in our setting, and we point the reader to those works for several examples further to those we give here.

[3] Implicit here is the assumption that all rules of L have at most two premisses, so $n \leq 2$. This assumption covers all systems in this work.

▶ **Example 4.5** (Naturals, evens and odds: proving relationships)**.** Let us revisit once more Example 2.3. Several examples about the relationships between N, E, O for a similar framework of first-order logic with inductive definitions are given in [7, 9, 10], in particular including ones with complex cycle structure. Here we shall instead revisit the relationship between the inductive predicates M and N.

Recall that we showed in Example 2.8 that N and M compute the same set, namely $\mathbb{N}$, in the standard model. We can show this formally within $\mathsf{CID}_{<\omega}$ by means of cyclic proofs. For the direction $M \subseteq N$:

$$
{\scriptstyle I_m\text{-}l}\,\dfrac{{\scriptstyle \vee\text{-}l}\,\dfrac{{\scriptstyle \vee\text{-}l}\,\dfrac{{\scriptstyle N(0)}\,\dfrac{}{x = 0 \Rightarrow N(x)} \qquad {\scriptstyle =,\wedge,\exists}\,\dfrac{{\scriptstyle N(\mathsf{s}\mathsf{s})}\,\dfrac{{\scriptstyle [y/x]}\,\dfrac{{\scriptstyle I_m\text{-}l}\,\dfrac{\vdots}{M(x) \Rightarrow N(x)}\,\bullet}{M(y) \Rightarrow N(y)}}{M(y) \Rightarrow N(\mathsf{s}\mathsf{s}y)}}{\exists y(x = \mathsf{s}\mathsf{s}y \wedge M(y)) \Rightarrow N(x)}}{e(M,x) \Rightarrow N(x)} \qquad {\scriptstyle \mathsf{w},\wedge}\,\dfrac{{\scriptstyle N(1)}\,\dfrac{}{x = 1 \Rightarrow N(1)}}{\forall y(E(y) \to M(y)) \wedge x = 1 \Rightarrow N(x)}}{m(M,x) \Rightarrow N(x)}}{M(x) \Rightarrow N(x)}\,\bullet
$$

where the derivations marked $N(0), N(\mathsf{ss}), N(1)$ all have simple finite proofs by unfolding N on the RHS. Again we indicate by $\bullet$ roots of identical subproofs, and the only infinite branch, looping on $\bullet$, has progressing trace in blue.

4.2 Simulating inductive proofs

Our cyclic system $\mathsf{CID}_{<\omega}$ subsumes $\mathsf{ID}_{<\omega}$ by a standard construction:

▶ **Theorem 4.6** (Induction to cycles)**.** *If* $\mathsf{ID}_{<\omega} \vdash \varphi$ *then* $\mathsf{CID}_{<\omega} \vdash \varphi$.

Proof sketch. We proceed by induction on the structure of a $\mathsf{ID}_{<\omega}$ proof. The critical step is $\mathsf{ind}(\varphi)$, for which we do not have a corresponding rule in $\mathsf{LID}^-_{<\omega}$. We simulate this rule by,

$$
{\scriptstyle \mathsf{cut}}\,\dfrac{{\scriptstyle I_\varphi\text{-}l}\,\dfrac{{\scriptstyle \mathsf{cut}}\,\dfrac{{\scriptstyle \varphi}\,\dfrac{{\scriptstyle I_\varphi\text{-}l}\,\dfrac{\vdots}{\Gamma, I_\varphi(t) \Rightarrow \Delta, \psi(t)}\,\bullet}{\Gamma, \varphi(I_\varphi, t) \Rightarrow \Delta, \varphi(\psi, t)} \qquad {\scriptstyle [t/y]}\,\dfrac{\Gamma, \varphi(\psi, y) \Rightarrow \Delta, \psi(y)}{\Gamma, \varphi(\psi, t) \Rightarrow \Delta, \psi(t)}}{\Gamma, \varphi(I_\varphi, t) \Rightarrow \Delta, \psi(t)}}{\Gamma, I_\varphi(t) \Rightarrow \Delta, \psi(t)}\,\bullet \qquad \Gamma, \psi(t) \Rightarrow \Delta}{\Gamma, I_\varphi(t) \Rightarrow \Delta}
$$

where $\bullet$ marks roots of identical subproofs and the derivation marked φ is obtained by induction on the structure of φ. Any infinite branch is either progressing by the induction hypothesis, or loops infinitely on $\bullet$ and has the progressing trace coloured in blue. ◀

Of course, the converse result is much harder (and, indeed, implies soundness of cyclic proofs).

4.3 About traces

Our notion of (progressing) trace may seem surprisingly simple to the seasoned cyclic proof theorist, when comparing to analogous conditions in similar logics such as the μ-calculus requiring complex "signatures", e.g. [18, 25, 2]. However this simplicity arises naturally from the way we have formulated our syntax. Let us take some time to detail some of properties of (progressing) traces that will facilitate our soundness argument later.

Write $\mathcal{I}$ for the set of inductive predicates of $\mathcal{L}_{<\omega}$ (i.e. the set of symbols I_φ). Write $<$ for the smallest transitive relation on $\mathcal{I}$ satisfying:

- if I_φ occurs in $\psi(X, x)$ then $I_\varphi < I_\psi$.

By the inductive definition of the language $\mathcal{L}_{<\omega}$, it is immediate that $<$ is a well-founded relation on $\mathcal{I}$. In what follows, we shall extend $<$ arbitrarily to a (total) well-order on $\mathcal{I}$, so as to freely use of terminology peculiar to linear orders.

▶ **Proposition 4.7** (Properties of progressing traces). *Let $(\tau_i)_{i \geq k}$ be a progressing trace. There is a (unique) inductive predicate symbol I_ψ and some $k' \geq k$ such that:*

1. *τ_i is of the form $I_\psi(t)$ and principal for infinitely many $i \geq k$;*
2. *I_ψ occurs positively in each τ_i, for $i \geq k'$;*
3. *for any $j \geq k'$ and I_χ occurring in τ_j, we have $I_\chi \leq I_\psi$.*

5 Soundness of non-wellfounded proofs

The main goal of this section is to prove the following result:

▶ **Theorem 5.1** (Soundness). *If $\mathsf{LID}^-_{<\omega} \vdash_{\mathrm{nwf}} \varphi$ then $\mathfrak{N} \vDash \varphi$.*

Before proving this, it is convenient to omit consideration of substitutions in preproofs:

▶ **Proposition 5.2** (Admissibility of substitution). *If there is a (non-wellfounded) $\mathsf{LID}^-_{<\omega}$-proof of a sequent $\Gamma \Rightarrow \Delta$, then there is one not using the substitution rule.*

5.1 Satisfaction with respect to approximants

Before proceeding, let us build up a little more theory about approximants of (least) fixed points. Let us temporarily expand the language $\mathcal{L}_{<\omega}$ to include, for each inductive predicate symbol I_φ and each ordinal α a symbol I^α_φ. We do not consider these symbols “inductive predicates”, but rather refer to them as *approximant symbols*. In the standard model, using the notation of Section 2, we set $(I^\alpha_\varphi)^{\mathfrak{N}} := (\varphi^{\mathfrak{N}})^\alpha(\varnothing)$.

For a formula φ of $\mathcal{L}_{<\omega}$ whose $<$-greatest inductive predicate in positive position is I_ψ, we write φ^α for the formula obtained from φ by replacing each positive occurrence of I_ψ by I^α_ψ. As an immediate consequence of the characterisation of least fixed points by unions of approximants, Proposition 2.7, we have:

▶ **Corollary 5.3** (of Proposition 2.7). *If $\mathfrak{N} \vDash \varphi$ then there is an ordinal α such that $\mathfrak{N} \vDash \varphi^\alpha$.*

Note that, as a consequence of positivity implying monotonicity, we also have:

▶ **Corollary 5.4** (of Proposition 2.1). *If $\alpha \leq \beta$ then $\mathfrak{N} \vDash \varphi^\alpha \to \varphi^\beta$.*

Finally, let us point out that, by the definition of the inflationary construction in Equation (1), if $t^{\mathfrak{N}} \in \mu\varphi^{\mathfrak{N}}$, then the least ordinal α with $t^{\mathfrak{N}} \in (\varphi^{\mathfrak{N}})^\alpha$ must be a successor ordinal. Albeit rather immediate, we better state the following consequence of this reasoning:

▶ **Observation 5.5.** *If α, β are least s.t. $\mathfrak{N} \vDash I^\alpha_\varphi(t)$ and $\mathfrak{N} \vDash \varphi(I^\beta_\varphi, t)$ respectively, then $\beta < \alpha$.*

5.2 Building countermodels

An *assignment* is a (partial) map ρ from variables to natural numbers. If φ is a formula and $\rho : \mathsf{FV}(\varphi) \to \mathbb{N}$, we define $\mathfrak{N}, \rho \vDash \varphi$ (or simply $\rho \vDash \varphi$) by simply interpreting free variables under ρ in $\mathfrak{N}$. Formally, $\mathfrak{N}, \rho \vDash \varphi$ if $\mathfrak{N} \vDash \varphi\,[\rho(x)/x]_{x \in \mathsf{FV}(\varphi)}$.[4]

[4] Note here we are implicitly identifying natural numbers with their corresponding numerals.

As a consequence of local soundness of the rules, as well as preserving truth we have that rules "reflect" falsity. In fact we can say more:

▶ **Lemma 5.6** (Reflecting falsity). *Fix an inference step:*

$$\mathsf{r}\,\frac{\Gamma_1 \Rightarrow \Delta_1 \quad \cdots \quad \Gamma_n \Rightarrow \Delta_n}{\Gamma \Rightarrow \Delta} \tag{4}$$

If $\rho \vDash \bigwedge \Gamma$ and $\rho \nvDash \bigvee \Delta$ then there is an assignment ρ' and premiss $\Gamma' \Rightarrow \Delta'$ with:

1. *ρ' extends ρ, i.e. $\rho'(x) = \rho(x)$ for any x in the domain of ρ;*
2. *$\rho' \vDash \bigwedge \Gamma'$ and $\rho' \nvDash \bigvee \Delta'$;*
3. *if $\psi \in \Gamma'$ is an immediate ancestor of $\varphi \in \Gamma$ then either:*
   a. *I, I' are the greatest inductive predicates occurring in φ, ψ resp. and $I' < I$; or,*
   b. *For any ordinal α, we have $\rho \vDash \varphi^\alpha \implies \rho' \vDash \psi^\alpha$.*

The proof is similar to analogous results in [21, 11], however we must also take care to maintain the invariant Item 3 during the construction. An important distinction here is that, for Item 3b, we must find the least ordinal approximating the principal formula of, say a $\vee$-left step, and evaluate auxiliary formulas with respect to this ordinal in order to appropriately choose the correct premiss. The required property then follows by monotonicity, Proposition 2.1, and the fact that approximants form an increasing chain, cf. Equation (1). The necessity of this consideration is similar to (but somewhat simpler than) analogous issues arising in the cyclic proof theory of the modal μ-calculus, cf. [18, 25, 2].

5.3 Putting it all together

We are now ready to prove the main result of this section.

Proof of Theorem 5.1. Let π be a (non-wellfounded) $\mathsf{LID}^-_{<\omega}$ proof of the sequent $\Rightarrow \varphi$ and suppose, for contradiction, that $\mathfrak{N} \nvDash \varphi$. We define a branch $(v_i)_{i<\omega}$ and assignments $(\rho_i)_{i<\omega}$ by setting:

- $\rho_0 := \varnothing$ and $v_0 := \varepsilon$ (the root of π);[5]
- appealing to Lemma 5.6, if $\pi(v_i)$ has form (4), we set v_{i+1} s.t. $\pi(v_{i+1})$ has conclusion $\Gamma' \Rightarrow \Delta'$ and $\rho_{i+1} := \rho'_i$.

By assumption that π is progressing, let $(\tau_i)_{i\geq k}$ be a progressing trace along $(v_i)_{i<\omega}$, and let α_i be the least ordinals such that $\mathfrak{N} \vDash \tau_i^{\alpha_i}$ for $i \geq k$.

Now, let $k' \geq k$ and I_ψ be obtained from $(\tau_i)_{i\geq k}$ by Proposition 4.7. By Items 2 and 3 of Proposition 4.7 we have that I_ψ is the greatest inductive predicate occurring (positively) in each τ_i, for $i \geq k'$, and so Item 3a of Lemma 5.6 never applies (for $i \geq k'$). Thus, by Proposition 2.1, we have $\alpha_{i+1} \leq \alpha_i$ for $i \geq k'$.

On the other hand, at any I_ψ-l step where τ_i is principal, for $i \geq k'$, we must have that $\alpha_{i+1} < \alpha_i$ by Observation 5.5. Since this happens infinitely often, by Item 1 of Proposition 4.7, we conclude that $(\alpha_i)_{i\geq k'}$ is a monotone non-increasing sequence of ordinals that does not converge, contradicting the well-foundedness of ordinals. ◀

[5] We assume here that φ is closed, i.e. has no free variables.

6 Inductive definitions and truth in second-order arithmetic

The remainder of this paper is devoted to proving the converse of Theorem 4.6. For this, we are inspired by the ideas of previous work [21, 11], using "second-order" theories to formalise the metatheorems of cyclic systems (namely soundness), and then appealing to conservativity results. However the exposition here is far more involved than the analogous ones for arithmetic. For this reason, we rather rely on a formalisation of the "theory of recursive ordinals" (with parameters) in Π^1_1-CA_0, and formalise the soundness argument abstractly in this way.

6.1 Subsystems of second-order arithmetic and inductive definitions

We shall work with common subsystems of second-order arithmetic, as found in textbooks such as [22], and assume basic facts about them.

In particular, recall that ACA_0 is a two-sorted extension of basic arithmetic by:

- *Arithmetical comprehension.* $\exists X \forall x (X(x) \leftrightarrow \varphi(x))$ for each arithmetical formula $\varphi(x)$.
- *Set induction.* $\forall X (X(0) \to \forall x (X(x) \to X(\mathsf{s}x)) \to \forall x X(x))$

From here Π^1_1-CA_0 is the extension of ACA_0 by the comprehension schema for all Π^1_1 formulas. It is well-known that Π^1_1-CA_0 proves also the Σ^1_1-comprehension scheme, a fact that we shall freely use, along with other established principles, e.g. from [22].

We can interpret $\mathcal{L}_{<\omega}$ into the language of second-order arithmetic by:

$$I_\varphi(t) \quad := \quad \forall X ((\forall x \varphi(X, x) \to X(x)) \to X(t)) \tag{5}$$

This interpretation induces a bona fide (and well-known) encoding of $\mathsf{ID}_{<\omega}$ within Π^1_1-CA_0, and we shall henceforth freely use (arithmetical) inductive predicates when working within Π^1_1-CA_0, always understanding them as abbreviations under (5). In fact, we can make a stronger statement. Not only does Π^1_1-CA_0 extend $\mathsf{ID}_{<\omega}$ arithmetically, it does so conservatively:

▶ **Theorem 6.1** (E.g., [12]). *Π^1_1-CA_0 is arithmetically conservative over $\mathsf{ID}_{<\omega}$.*

This is a nontrivial but now well-known result in proof theory whose details we shall not recount. We will use this result as a "black box" henceforth.

6.2 Satisfaction as an inductive definition

As usual, there is no universal (first-order) truth predicate for a predicate language, for Tarskian reasons. However we may define *partial* truth predicates for fragments of the language. In a language closed under inductive definitions, this is particularly straightforward since satisfaction itself is inductively defined (at the meta level). In what follows we will employ standard metamathematical notations and conventions for coding, e.g. we write $\ulcorner E \urcorner$ for the Gödel code of an expression E. Also, when it is not ambiguous, we shall typically use the same notation for meta-level objects/operations and their object-level (manipulations on) codes, as a convenient abuse of notation.

▶ **Proposition 6.2** (Formalised relative satisfaction). *Let $\vec{X} = X_1, \dots, X_k$ be a sequence of set symbols. There is a Π^1_1 formula $\mathrm{Sat}_{\vec{X}}(\rho, m, \vec{A})$ such that Π^1_1-CA_0 proves the characterisation in Figure 2 for φ, ψ ranging over arithmetical formulas over $\vec{X}$.*

▶ **Corollary 6.3** (Reflection, Π^1_1-CA_0). *For any arithmetical formula $\varphi(\vec{X}, \vec{x})$ with all free first-order variables displayed, we have $\mathrm{Sat}_{\vec{X}}(\rho, \ulcorner \varphi(\vec{X}, \vec{x}) \urcorner, \vec{A}) \leftrightarrow \varphi(\vec{A}, \rho(\vec{x}))$.*

$$\forall \rho, m, \vec{A} \left(\mathrm{Sat}_{\vec{X}}(\rho, m, \vec{A}) \quad \leftrightarrow \quad \begin{array}{rl} & m = \ulcorner s = t \urcorner \wedge \rho(s) = \rho(t) \\ \vee & m = \ulcorner s < t \urcorner \wedge \rho(s) < \rho(t) \\ \vee & m = \ulcorner \varphi \vee \psi \urcorner \wedge \big(\mathrm{Sat}_{\vec{X}}(\rho, \ulcorner \varphi \urcorner, \vec{A}) \vee \mathrm{Sat}_{\vec{X}}(\rho, \ulcorner \psi \urcorner, \vec{A})\big) \\ \vee & m = \ulcorner \varphi \wedge \psi \urcorner \wedge \big(\mathrm{Sat}_{\vec{X}}(\rho, \ulcorner \varphi \urcorner, \vec{A}) \wedge \mathrm{Sat}_{\vec{X}}(\rho, \ulcorner \psi \urcorner, \vec{A})\big) \\ \vee & m = \ulcorner \exists x \varphi \urcorner \wedge \exists n \, \mathrm{Sat}_{\vec{X}}(\rho\{x \mapsto n\}, \ulcorner \varphi \urcorner, \vec{A}) \\ \vee & m = \ulcorner \forall x \varphi \urcorner \wedge \forall n \, \mathrm{Sat}_{\vec{X}}(\rho\{x \mapsto n\}, \ulcorner \varphi \urcorner, \vec{A}) \\ \vee & \bigvee_{i=1}^{k} (m = \ulcorner X_i(t) \urcorner \wedge \rho(t) \in A_i) \\ \vee & \bigvee_{i=1}^{k} (m = \ulcorner \neg X_i(t) \urcorner \wedge \rho(t) \notin A_i) \end{array} \right)$$

Figure 2 Inductive characterisation of the satisfaction predicate.

7 Approximants and transfinite recursion in second-order arithmetic

Throughout this section we shall fix a list $\vec{X}$ of set variables that may occur as parameters in all formulas. We shall almost always suppress them. We work within Π^1_1-CA_0 throughout this section, unless otherwise stated.

7.1 Order theory and transfinite recursion in second-order arithmetic

We assume some basic notions for speaking about (partial) (well-founded) orders in second-order arithmetic, and some well-known facts about them. Definitions and propositions in this section have appeared previously in the literature, e.g., [22].

A *(binary) relation* is a set symbol R, construed as a set of pairs, with *domain* $|R| := \{x : R(x,x)\}$. We write simply $x \leq_R y$ for $x \in |R| \wedge y \in |R| \wedge R(x,y)$ and $x <_R y := x \leq_R y \wedge \neg x = y$. We write:

- $\mathrm{LO}(R)$ for an arithmetical formula stating that $<_R$ is a linear order on $|R|$.
- $\mathrm{WF}(R)$ for a Π^1_1-formula stating that $<_R$ is well-founded on $|R|$.
- $\mathrm{WO}(R) := \mathrm{LO}(R) \wedge \mathrm{WF}(R)$. ("$R$ is a well-order")
- $R <_{\mathrm{WO}} R'$ if $\mathrm{WO}(R), \mathrm{WO}(R')$ and there is an order preserving bijection from R onto a proper initial segment of R'. ($<_{\mathrm{WO}}$ is provably Δ^1_1 within Π^1_1-CA_0).

We have, already in ACA_0, transfinite induction (for sets) over any well-order:

▶ **Proposition 7.1.** $\forall X, R(\mathrm{WO}(R) \to \forall a \in |R| \, (\forall b <_R a \, X(b) \to X(a)) \to \forall a \in |R| \, X(a))$

More importantly we have that the class of well-orders itself is well-founded under comparison:

▶ **Proposition 7.2** (Well-orders are well-ordered, ATR_0). *If $F : \mathbb{N} \to \mathrm{WO}$ then there is $n \in \mathbb{N}$ with $F(n+1) \not<_{\mathrm{WO}} F(n)$*

An important principle within Π^1_1-CA_0 is *arithmetical transfinite recursion* (ATR). Since we shall need to later bind the well-order over which recursion takes place, we better develop the principle explicitly.

▶ **Definition 7.3** (Approximants). *Let $\varphi(X,x)$ be arithmetical and R a relation. We define:*

$$I^R_\varphi(a,x) := \exists F \subseteq |R| \times \mathbb{N} \, (\forall b \in |R| \, \forall y \, (F(b,y) \to \exists c <_R b \, \varphi(F(c), y)) \wedge F(a,x))$$

Intuitively we may see $I^R_\varphi(a)$ as the union of a family of sets $F(b)$, indexed by $b <_R a$, satisfying $F(b) = \bigcup\limits_{c <_R b} \varphi(F(c))$, here construing $\varphi(-)$ as an operation on sets. The notation we have used is suggestive: the point of this section is to characterise inductive definitions in terms of approximants given by transfinite recursion.

Note that I^R_φ is a Σ^1_1-formula. The following is well-known:

▶ **Proposition 7.4** (Bounded recursion). *Let $\varphi(X, x)$ be an arithmetical formula and suppose* $\mathrm{WO}(R)$. *I^R_φ is a set (uniquely) satisfying:*

$$\forall a \in |R|\, \forall x\, (I^R_\varphi(a, x) \leftrightarrow \exists b <_R a\, \varphi(I^R_\varphi(b), x)) \qquad \left(\textit{i.e.}\ \ I^R_\varphi(a) = \bigcup_{b <_R a} \varphi(I^R_\varphi(b)) \right) \tag{6}$$

As a consequence of transfinite induction, Proposition 7.1 we have:

▶ **Corollary 7.5.** *Let $\varphi(X, x)$ be arithmetical and positive in X, and suppose* $\mathrm{WO}(R)$. *We have $\forall a <_R b\, \forall x (I^R_\varphi(a, x) \to I^R_\varphi(b, x))$.*

Intuitively the above statement tells us that $I^R_\varphi(-)$ forms an increasing chain along R.

Henceforth we write $I^R_\varphi(x) := \exists a \in |R|\, I^R_\varphi(a, x)$ which, with R occurring as a parameter, is again a Σ^1_1 formula.

7.2 Formalising recursive ordinals and approximants

Π^1_1-CA_0 is not strong enough a theory to characterise inductive definitions by limits of approximants, in general. However, when the closure ordinals of inductive definitions are recursive, they may be specified by finite data and duly admit such a characterisation within Π^1_1-CA_0. This subsection is devoted to a development of this characterisation; the definitions and propositions have appeared previously in the literature, e.g., [12, 13].

Let us fix a recursive enumeration of Σ^0_1-formulas with free (first-order) variables among x, y, and write α, β etc. to range over their Gödel codes. Thanks to a (relativised) universal Σ^0_1-formula, we can readily evaluate (the codes of) Σ^0_1 formulas already within RCA_0. In this way we may treat α, β etc. as binary relations, and duly extend the notations of the previous subsections approriately, e.g. freely writing $|\alpha|, \leq_\alpha, <_\alpha, \mathrm{LO}(\alpha), \mathrm{WF}(\alpha), \mathrm{WO}(\alpha), \alpha <_{\mathrm{WO}} \beta, I^\alpha_\varphi$.

▶ **Definition 7.6** (Recursive ordinals). *Write $\mathcal{O} := \{\alpha : \mathrm{WO}(\alpha)\}$, obtained by Π^1_1-comprehension, and $\alpha <_\mathcal{O} \beta$ for $\mathcal{O}(\alpha) \wedge \mathcal{O}(\beta) \wedge \alpha <_{\mathrm{WO}} \beta$.*

We also write $I^\mathcal{O}_\varphi(x) := \exists \alpha \in \mathcal{O}\, I^\alpha_\varphi(x)$.

Of course, well-foundedness of $\mathcal{O}$ under $<_\mathcal{O}$ is directly inherited from well-foundedness of WO under $<_{\mathrm{WO}}$, Proposition 7.2. Note that $I^\mathcal{O}_\varphi(x)$ is again a Σ^1_1-formula, and so we have access to $I^\mathcal{O}_\varphi$ as a set within Π^1_1-CA_0. In fact we even have access to the restriction $I^-_\varphi(-) \subseteq \mathcal{O} \times \mathbb{N}$ again by Σ^1_1-comprehension. As a result we can give a recursive characterisation of $I^\mathcal{O}_\varphi$ similar to Proposition 7.4 but at the level of $\mathcal{O}$:

▶ **Proposition 7.7** (Recursion). *Let $\varphi(X, x)$ be arithmetical and positive in X. We have:*

$$\forall \alpha \in \mathcal{O}\, \forall x\, (I^\alpha_\varphi(x) \leftrightarrow \exists \beta <_\mathcal{O} \alpha\, \varphi(I^\beta_\varphi, x)) \qquad \left(\textit{i.e.}\ \ I^\alpha_\varphi = \bigcup_{\beta <_\mathcal{O} \alpha} \varphi(I^\beta_\varphi) \right) \tag{7}$$

The following are well-known properties about $\mathcal{O}$:

▶ **Proposition 7.8** (Properties of $\mathcal{O}$). *We have the following:*

1. *(Increase)* $\forall \alpha \in \mathcal{O}\, \exists \beta \in \mathcal{O}\, \alpha <_{\mathcal{O}} \beta$.
2. *(Collection)* $\forall x \exists \alpha \in \mathcal{O}\, \varphi \to \exists \beta\, \forall x\, \exists \alpha <_{\mathcal{O}} \beta\ \varphi$.

Turning back to positive formulas again, we have the following useful consequence:

▶ **Corollary 7.9.** *Let $\varphi(X,x)$ and $\psi(X)$ be arithmetical and positive in X. $\psi(I_\varphi^{\mathcal{O}}) \to \exists \alpha \psi(I_\varphi^\alpha)$.*

7.3 Characterising inductive definitions as limits of approximants

The main result of this section is:

▶ **Theorem 7.10** (Characterisation). *$\forall x(I_\varphi(x) \leftrightarrow I_\varphi^{\mathcal{O}}(x))$ (i.e. $I_\varphi = I_\varphi^{\mathcal{O}}$).*

Proof sketch. For $(\to)$, it suffices to show that $I_\varphi^{\mathcal{O}}$ is a prefixed point of $\varphi(-)$:[6]

$$\begin{array}{rll}
\varphi(I_\varphi^{\mathcal{O}}) \subseteq \varphi(I_\varphi^\alpha) & & \text{for some } \alpha\text{, by Corollary 7.9} \\
\varphi(I_\varphi^{\mathcal{O}}) \subseteq I_\varphi^\beta & & \text{for some } \beta >_{\mathsf{WO}} \alpha \text{ by Propositions 7.7 and 7.8} \\
\varphi(I_\varphi^{\mathcal{O}}) \subseteq I_\varphi^{\mathcal{O}} & & \text{by definition of } I_\varphi^{\mathcal{O}} \\
I_\varphi \subseteq I_\varphi^{\mathcal{O}} & & \because \Pi^1_1\text{-}\mathsf{CA}_0 \text{ proves } I_\varphi \text{ is least among pre-fixed points}
\end{array}$$

Note here it is crucial that we have access to $I_\varphi^{\mathcal{O}}$ as a set, thanks to Σ^1_1-comprehension.

For $(\leftarrow)$, we show $I_\varphi^\alpha(a) \subseteq I_\varphi^{\mathcal{O}}$ (i.e. $\forall x(I_\varphi^\alpha(a,x) \to I_\varphi(x))$) by α-induction on $a \in |\alpha|$:

$$\begin{array}{rll}
I_\varphi^\alpha(b) \subseteq I_\varphi & & \forall b <_\alpha a \text{ by inductive hypothesis} \\
\bigcup\limits_{b <_\alpha a} I_\varphi^\alpha(b) \subseteq I_\varphi & & \text{by } \exists\text{-left-introduction} \\
\varphi\left(\bigcup\limits_{b <_\alpha a} I_\varphi^\alpha(b)\right) \subseteq \varphi(I_\varphi) & & \text{by positivity of } \varphi(-) \\
I_\varphi^\alpha(a) \subseteq \varphi(I_\varphi) & & \text{by (6)} \\
I_\varphi^\alpha(a) \subseteq I_\varphi & & \because \Pi^1_1\text{-}\mathsf{CA}_0 \text{ proves } I_\varphi \text{ is a pre-fixed point}
\end{array}$$

Note here that it is crucial that we have access to I_φ as set, thanks to Π^1_1-comprehension. ◀

8 Simulating cyclic proofs within $\mathsf{ID}_{<\omega}$

The goal of this section is to finally establish the converse to Theorem 4.6:

▶ **Theorem 8.1.** *Let φ be arithmetical. If $\mathsf{CID}_{<\omega} \vdash \varphi$ then $\mathsf{ID}_{<\omega} \vdash \varphi$.*

The argument proceeds essentially by formalising the soundness argument of Section 5 *within* Π^1_1-CA_0, with respect to the partial satisfaction predicate Sat. We spend most of this section explaining this formalisation.

We henceforth work within Π^1_1-CA_0, unless otherwise stated.

Necessity of non-uniformity. In light of Theorem 6.1 and Theorem 4.6, we obviously cannot formalise soundness of $\mathsf{CID}_{<\omega}$ *uniformly* within Π^1_1-CA_0, for Gödelian reasons. Instead we take a non-uniform approach. Let us henceforth fix a $\mathsf{CID}_{<\omega}$ proof π of a sequent $\Gamma \Rightarrow \Delta$. We assume π uses only inductive predicates among $\vec{I} = I_{\varphi_1}, \ldots, I_{\varphi_n}$. All notions about (recursive) ordinals from Section 7 are now relativised to $\vec{I}$ (recall that we allowed free set variables to occur as parameters throughout).

[6] Here we are using expressions, say, $\varphi(A) \subseteq B$ as an abbreviation for $\forall x(\varphi(A,x) \to B(x))$.

Formalising properties of traces. The results of Section 4.3, in particular Proposition 4.7, involve only finitary combinatorics and are readily formalisable already within RCA_0, essentially following the given (meta-level) proofs.

"Knowing that" a regular proof is progressing. At some point during the soundness argument, namely after constructing the "countermodel branch", we shall need to extract a progressing thread from an infinite branch of π. However, this requires our ambient theory *knowing* that π is progressing, hitherto a meta-level assumption. Let us point out that in our non-uniform exposition, for fixed π, known progressiveness has been shown to be available in even the weakest of the "big five":

▶ **Proposition 8.2** (RCA_0, [11])**.** *π is progressing.*

Formalised admissibility of substitution. The admissibility of substitution, Proposition 5.2, is available already in weak theories by a simple inductive construction: from π define π' a substitution-free $\mathsf{LID}^-_{<\omega}$ non-wellfounded proof node-wise by simply composing the (finitely many) substitutions up to a node. The progressing criterion means that there are, in particular, infinitely many non-substitution steps along any infinite branch, and so by (weak) König's lemma have that the resulting binary tree is well-defined.

We henceforth work with π' a substitution-free $\mathsf{LID}^-_{<\omega}$ non-wellfounded proof of $\Gamma \Rightarrow \Delta$ using only inductive predicates among $\vec{I} = I_{\varphi_1}, \ldots, I_{\varphi_n}$, that we "know" is progressing.

Formalising satisfaction with respect to approximants. We already defined recursive approximants in Π^1_1-CA_0 in Section 7.2. The formalised version of Corollary 5.3 is given by Corollary 7.9, and the formalised version of Corollary 5.4 is available already in pure logic. The existence of least ordinals satisfying a property is given by well-foundedness of WO under $<_{\mathrm{WO}}$, Proposition 7.2, and thus Observation 5.5 follows from Equation (6).

Formalised building countermodels. To speak about satisfaction and truth of formulas in π', we use the formalised notion $\mathrm{Sat}_{\vec{I}}$ in place of the meta-level "$\models$". Note that the inductive predicates occurring in π' parametrise the satisfiability predicate. From here Lemma 5.6 is formalised by proving soundness of the rules of $\mathsf{LID}^-_{<\omega}$ with respect to $\mathrm{Sat}_{\vec{I}}$, keeping track of immediate ancestry and using the results of the previous subsection. We use the (formalised) notions I^α_φ as inputs to $\mathrm{Sat}_{\vec{I}}$ in order to evaluate formulas like φ^α, and we rely on well-foundedness of the class of well-orders, Proposition 7.2, to make the correct decisions cf. Item 3b. Let us point out that, for a fixed step r, the description of (ρ', S') from (ρ, S) is arithmetical in $\vec{I}, \mathrm{Sat}_{\vec{I}}$, $<_{\mathrm{WO}}$, $\mathcal{O}$ and $\vec{I}^-$, by essentially following the specification in the Lemma statement, relativising "ordinals" to $\mathcal{O}$.

Putting it all together, formally. Finally let us discuss how the proof of Theorem 5.1 (for π') is formalised. Recall that the infinite "countermodel branch" $(v_i)_{i<\omega}$ is recursive in the construction from (formalised) Lemma 5.6. Since that construction was arithmetical (in certain set symbols), we indeed have access to the countermodel branch $(v_i)_{i<\omega}$ as a set by comprehension. Now, since we know that π' is progressing, we can duly take a progressing trace $(\tau_i)_{i\geq k}$ along it. From here the obtention of the sequence of (now recursive) ordinals $(\alpha_i)_{i\geq k}$ is obtained by a simple comprehension instance arithmetical in $\mathrm{Sat}_{\vec{I}}$, $\vec{I}^-$ and $<_{\mathrm{WO}}$. The remainder of the argument goes through as written, appealing to formalised versions of auxiliary statements.

From here we may conclude the main result of this section as promised:

Proof sketch of Theorem 8.1. From $\mathsf{CID}_{<\omega} \vdash \varphi$, for φ arithmetical, the explanations in this section give us $\Pi^1_1\text{-}\mathsf{CA}_0 \vdash \mathrm{Sat}_\varnothing(\varnothing, \ulcorner\varphi\urcorner, \varnothing)$. By reflection, Corollary 6.3, we thus have $\Pi^1_1\text{-}\mathsf{CA}_0 \vdash \varphi$, and so by conservativity, Theorem 6.1, we have $\mathsf{ID}_{<\omega} \vdash \varphi$, as required. ◀

9 Conclusions

We presented a new cyclic system $\mathsf{CID}_{<\omega}$ formulated over the language $\mathcal{L}_{<\omega}$ of finitely iterated arithmetical inductive definitions. We showed the arithmetical equivalence of $\mathsf{CID}_{<\omega}$ and its inductive counterpart $\mathsf{ID}_{<\omega}$ by nontrivially extending techniques that have recently appeared in the setting of cyclic arithmetic [21, 11]. Among other things, this work serves to further test the metamathematical techniques and methodology now available in cyclic proof theory.

Extensions of predicate logic by "ordinary" inductive definitions, which are essentially quantifier-free but allow for a form of simultaneous induction, were extensively studied by Brotherston and Simpson, in particular in the setting of cyclic proofs [7, 9, 10]. Indeed recently Berardi and Tatsuta have shown that cyclic systems for extensions of Peano and Heyting arithmetic by such inductive definitions prove the same theorems as the corresponding inductive systems [4, 5]. As noted by Das in [11] the result of [4] (for Peano arithmetic) is, in a sense, equivalent to Simpson's in [21] since ordinary inductive definitions can be encoded by Σ_1-formulas: closure ordinals of ordinary inductive definitions are always bounded above by ω. Comparing to the current work, recall that the closure ordinals of even a single arithmetical inductive definition exhaust all recursive ordinals.

There are many other possible extensions of the language of arithmetic $\mathcal{L}_A$ by fixed points. One natural avenue for further work would be to consider $\mathcal{L}_\alpha$ for both $\alpha < \omega$ and $\alpha \geq \omega$. Again the corresponding finitary systems ID_α play a crucial role in the ordinal analysis of stronger impredicative subsystems of second-order arithmetic (see, e.g., [19]). However what may be more interesting in the context of cyclic proof theory is the extension of $\mathcal{L}_A$ (and $\mathcal{L}_{<\omega}$) by so-called "general" inductive definitions, as in [16, 17]. These essentially extend the syntax of $\mathcal{L}_A$ in the same way that fixed points of the modal μ-calculus extend the language of modal logic, in particular allowing set parameters within inductive definitions. Such a setting necessarily exhibits more complicated metatheory, but is a natural target in light of the origins of cyclic proof theory based in the μ-calculus and first-order logic with inductive definitions. To this end, let us point out that cyclic systems for the "first-order μ-calculus" have already appeared [24, 23, 1], and so could form the basis of such investigation.

References

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

2 Bahareh Afshari and Graham E. Leigh. Cut-free completeness for modal mu-calculus. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005088.

3 Stefano Berardi and Makoto Tatsuta. Classical system of martin-löf's inductive definitions is not equivalent to cyclic proof system. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 301–317, 2017. doi:10.1007/978-3-662-54458-7_18.

4 Stefano Berardi and Makoto Tatsuta. Equivalence of inductive definitions and cyclic proofs under arithmetic. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005114.

5 Stefano Berardi and Makoto Tatsuta. Intuitionistic podelski-rybalchenko theorem and equivalence between inductive definitions and cyclic proofs. In Corina Cîrstea, 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 13–33. Springer, 2018. doi:10.1007/978-3-030-00389-0_3.

6 Stefano Berardi and Makoto Tatsuta. Classical system of martin-lof's inductive definitions is not equivalent to cyclic proofs. *Log. Methods Comput. Sci.*, 15(3), 2019. doi:10.23638/LMCS-15(3:10)2019.

7 James Brotherston. Cyclic proofs for first-order logic with inductive definitions. In Bernhard Beckert, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2005, Koblenz, Germany, September 14-17, 2005, Proceedings*, volume 3702 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2005. doi:10.1007/11554554_8.

8 James Brotherston, Dino Distefano, and Rasmus Lerchedahl Petersen. Automated cyclic entailment proofs in separation logic. In Nikolaj S. Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2011. doi:10.1007/978-3-642-22438-6_12.

9 James Brotherston and Alex Simpson. Complete sequent calculi for induction and infinite descent. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pages 51–62. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.16.

10 James Brotherston and Alex Simpson. Sequent calculi for induction and infinite descent. *J. Log. Comput.*, 21(6):1177–1216, 2011. doi:10.1093/logcom/exq052.

11 Anupam Das. On the logical complexity of cyclic arithmetic. *Log. Methods Comput. Sci.*, 16(1), 2020. doi:10.23638/LMCS-16(1:1)2020.

12 Solomon Feferman, W Sieg, and W Buchholz. *Iterated inductive definitions and subsystems of analysis: Recent proof-theoretical studies*. Springer, 1981.

13 Gerhard Jäger. Fixed points in peano arithmetic with ordinals. *Ann. Pure Appl. Log.*, 60(2):119–132, 1993. doi:10.1016/0168-0072(93)90039-G.

14 Leszek Aleksander Kolodziejczyk, Henryk Michalewski, Pierre Pradic, and Michal Skrzypczak. The logical strength of büchi's decidability theorem. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France*, volume 62 of *LIPIcs*, pages 36:1–36:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CSL.2016.36.

15 Leszek Aleksander Kolodziejczyk, Henryk Michalewski, Pierre Pradic, and Michal Skrzypczak. The logical strength of büchi's decidability theorem. *Log. Methods Comput. Sci.*, 15(2), 2019. doi:10.23638/LMCS-15(2:16)2019.

16 Robert S. Lubarsky. μ-definable sets of integers. *Journal of Symbolic Logic*, 58:291–313, 1993.

17 Michael Möllerfeld. *Generalized inductive definitions*. PhD thesis, WWU Münster, 2002. URL: https://nbn-resolving.de/urn:nbn:de:hbz:6-85659549572.

18 Damian Niwinski and Igor Walukiewicz. Games for the mu-calculus. *Theor. Comput. Sci.*, 163(1&2):99–116, 1996. doi:10.1016/0304-3975(95)00136-0.

19 Michael Rathjen and Wilfried Sieg. Proof Theory. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2022 edition, 2022.

20 Reuben N. S. Rowe and James Brotherston. Automatic cyclic termination proofs for recursive procedures in separation logic. In Yves Bertot and Viktor Vafeiadis, editors, *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, pages 53–65. ACM, 2017. `doi:10.1145/3018610.3018623`.

21 Alex Simpson. Cyclic arithmetic is equivalent to peano arithmetic. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 283–300, 2017. `doi:10.1007/978-3-662-54458-7_17`.

22 Stephen G. Simpson. *Subsystems of second order arithmetic*. Perspectives in mathematical logic. Springer, 1999.

23 Christoph Sprenger and Mads Dam. On global induction mechanisms in a μ-calculus with explicit approximations. *RAIRO Theor. Informatics Appl.*, 37(4):365–391, 2003. `doi:10.1051/ita:2003024`.

24 Christoph Sprenger and Mads Dam. On the structure of inductive reasoning: Circular and tree-shaped proofs in the μ-calculus. In Andrew D. Gordon, editor, *Foundations of Software Science and Computational Structures, 6th International Conference, FOSSACS 2003 Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings*, volume 2620 of *Lecture Notes in Computer Science*, pages 425–440. Springer, 2003. `doi:10.1007/3-540-36576-1_27`.

25 Thomas Studer. On the proof theory of the modal mu-calculus. *Stud Logica*, 89(3):343–363, 2008. `doi:10.1007/s11225-008-9133-6`.

26 Gadi Tellez and James Brotherston. Automatically verifying temporal properties of pointer programs with cyclic proof. In Leonardo de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 491–508. Springer, 2017. `doi:10.1007/978-3-319-63046-5_30`.

27 Gadi Tellez and James Brotherston. Automatically verifying temporal properties of pointer programs with cyclic proof. *J. Autom. Reason.*, 64(3):555–578, 2020. `doi:10.1007/s10817-019-09532-0`.

Concurrent Realizability on Conjunctive Structures

Emmanuel Beffara
Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, 38000 Grenoble, France

Félix Castro
IRIF, Université Paris Cité, France
IMERL, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay

Mauricio Guillermo
IMERL, Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay

Étienne Miquey
Aix-Marseille Université, CNRS, I2M, Marseille, France

Abstract

This work aims at exploring the algebraic structure of concurrent processes and their behavior independently of a particular formalism used to define them. We propose a new algebraic structure called conjunctive involutive monoidal algebra (CIMA) as a basis for an algebraic presentation of concurrent realizability, following ideas of the algebrization program already developed in the realm of classical and intuitionistic realizability. In particular, we show how any CIMA provides a sound interpretation of multiplicative linear logic. This new structure involves, in addition to the tensor and the orthogonal map, a parallel composition. We define a reference model of this structure as induced by a standard process calculus and we use this model to prove that parallel composition cannot be defined from the conjunctive structure alone.

2012 ACM Subject Classification Theory of computation → Process calculi; Theory of computation → Proof theory; Theory of computation → Linear logic

Keywords and phrases Realizability, Process Algebras, Concurrent Processes, Linear Logic

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.28

Related Version *Extended Version*: https://hal.science/hal-04083002

Funding This work was partially supported by projects STIC-AmSud *Qapla'* and *APCoRe*.

1 Introduction

1.1 Realizability and its algebrization

Realizability provides a well-established and general set of techniques for studying the relationships between programs and proofs. In the traditional presentation of intuitionistic realizability, one starts from a set A of realizers, which are objects with computational meaning, programs in some formalism (codes of recursive functions, λ-terms, etc). Logic is then interpreted in the powerset of A, in such a way that the meaning of a formula is essentially a set of programs sharing a computational behavior dictated by the formula. From an algebraic viewpoint, the set A induces in its powerset a *Heyting algebra*, which in turn induces a *topos* [12, 20].

Classical realizability adapts these principles to classical logic, building on different foundations [13]. The computational part is based on the duality between programs (potential proofs) and environments (counter-proofs). The proper categorical structure underlying classical realizability was discovered by Streicher [22] and involves an *ordered combinatory algebra* (OCA) induced by terms and stacks. This construction was later generalized in a different setting, namely Krivine ordered combinatory algebras [6], with the feature that

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 28; pp. 28:1–28:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Streicher's construction can be carried out in a purely axiomatic context. In particular, both truth values and realizers belong to the underlying set of these structures and the partial order subsumes subtyping, term reduction and the realizability relation. A similar approach was later followed within Miquel's implicative algebras [15] (and some variants by the last author's [16]), a slightly different structure which allows to encompass intuitionistic and classical models of forcing and realizability into a general framework, keeping the feature that truth values and realizers (and also forcing conditions) both belong to the underlying set of the implicative algebra. All these works therefore advocate for foundations of realizability seeking to transform a rather complex and operational definition into a much simpler and algebraic one.

1.2 Logic for concurrency

Process calculi form a wide range of formalisms designed to model concurrent systems and reason about them by means of term rewriting. Their applications are diverse, from the semantics of proof systems to the conception of concrete programming languages. Type systems for such calculi are therefore a wide domain, with systems of different kinds designed to capture different behaviors and ensure different properties of processes: basic interfacing, input-output discipline, linearity, lock-freeness, termination, respect of communication protocols, etc. To better understand the diversity of calculi and uncover basic structures and general patterns, many authors have searched for languages with simpler or more general theory in which the most features could be expressed by means of restrictions or encodings. Similar unification has been searched for in the realm of types, but no system can yet claim to be as basic and universal as, *e.g.*, simple types for the λ-calculus, which perfectly capture the abstraction and application mechanism in the logical world.

Concurrent realizability, developed by Beffara [2], transposes the ideas of classical realizability in a context of interacting concurrent processes. Following the constructions of phase semantics from linear logic [8] and ludics [9], the combinatory algebra is replaced with a variant of the π-calculus, endowed with a structure of commutative monoid under parallel composition. The pole, which defines orthogonality as in classical realizability, is seen as a testing protocol between processes, and the sets of processes that are closed by biorthogonality act both as truth values and behaviors. However, in the original construction, processes are subject to polarization and interfacing constraints which restrict the use of names, in order to avoid ambiguity when composing and to ensure consistency. As a consequence, operations over behaviors inherit such restrictions, which makes them partial in general.

The aim of the present work is to set the basis for an algebraic presentation of concurrent realizability, as a way to study processes and their types in a well-structured algebraic framework. We propose that, as with sequential models, the algebraic presentation is based on an ordered algebraic structure which must allow interpreting truth values and concurrent programs in its underlying set, subsuming the reduction semantics in the order relation. Furthermore, we want to avoid imposing a priori restrictions on processes.

1.3 Principles of the construction

We aim for an algebraic structure in which we can represent processes, types and operators in a uniform setting and we will refer to elements of such a structure as *behaviors*. A guiding intuition is to see behaviors as particular sets of terms in a process algebra, with some notion of closure by observational equivalence (a formal implementation is presented in Section 4). We postulate three fundamental structures over behaviors:

- a complete lattice structure, taking a comparison $a \leq b$ to mean that b exhibits more possible behaviors than a;
- a binary operator $\otimes$, continuous with respect to the lattice structure, that represents an operation of parallel composition without interaction.
- an anti-monotone unary operator $(\cdot)^{\perp}$ s.t. $a^{\perp}$ represents the tests that a passes.

This yields a variant of the last author's conjunctive structure [16], including notions of arrow and application and allowing for a sound interpretation of multiplicative linear logic. Within this framework, we study parallel composition as an additional continuous binary operation in the conjunctive structure.

Note that the notion of *name*, though pervasive in process algebra, is not taken to be primitive because different calculi and type systems make different choices on the use of names and actions (monadicity vs polyadicity, polarization, synchrony, etc). Instead, we consider that the multiplicative structure includes some way of managing connections between processes so that the axioms are satisfied.

1.4 Outline of the paper

We start by introducing in Section 2 the conjunctive structures that we identified to be conducive to the analysis of concurrent computation from an algebraic viewpoint. In Section 3, we develop a construction over process algebras to formally extend a language of processes with generalized fusions (PGF), on top of which we build a realizability model in Section 4 with the expected algebraic structure. We then explain in Section 5 how to equip conjunctive structures with parallelism and we illustrate it in the case of PGF. In Section 6, we prove that parallel composition cannot have an internal representation in the axioms of purely conjunctive structures, using a model that validates these axioms but does not have an operator for general parallel compisition.

2 Logic in conjunctive structures

Following the program of algebrization of realizability models that was mostly undertaken in the realm of Krivine realizability [22, 6, 15, 16] and previous work by the first author on concurrent computation [2], we introduce a particular class of *conjunctive algebras* which we identify as the key algebraic structures underlying realizability models induced by process calculi. We first define the notion of *conjunctive structure*, which reflects the algebraic structure of these models (*how are the truth values defined?*), and then define a notion of *separator* that allows to capture the logical content (*which processes define valid realizers?*).

2.1 Conjunctive structures

▶ **Definition 1.** *A conjunctive structure (CS) is a tuple* $(\mathbb{C}, \preccurlyeq, \otimes, (\cdot)^{\perp})$ *such that*

1. $(\mathbb{C}, \preccurlyeq)$ *is a complete lattice;*

2. $\otimes$ *is a binary monotone operation of* $\mathbb{C}$ *and* $(\cdot)^{\perp}$ *is a unary antimonotone function on* $\mathbb{C}$*;*

3. $\otimes$ *distributes over the join operation* $\curlyvee$*, i.e. for any* $a \in \mathbb{C}$ *and* $\mathfrak{B} \subseteq \mathbb{C}$ *we have* $\curlyvee_{b \in \mathfrak{B}}(a \otimes b) = a \otimes \left(\curlyvee_{b \in \mathfrak{B}} b\right)$ *and* $\curlyvee_{b \in \mathfrak{B}}(b \otimes a) = \left(\curlyvee_{b \in \mathfrak{B}} b\right) \otimes a$*;*

4. *the orthogonal map* $(\cdot)^{\perp}$ *satisfies De Morgan's law* $\left(\curlyvee_{b \in \mathfrak{B}} b\right)^{\perp} = \curlywedge_{b \in \mathfrak{B}} b^{\perp}$*.*

We say that the structure is involutive *(CIS) if* $(\cdot)^{\perp}$ *is involutive:* $a^{\perp\perp} = a$ *for all* $a \in \mathbb{C}$*. Also we call* unitary *every CS (resp. CIS) with a distinguished element* $1 \in \mathbb{C}$*.*

To draw the comparison with the conjunctive structures defined in the last author's work [16], the only difference lies in the use of an orthogonal map instead of a negation $\neg$ that was meant to convey a computational content. In particular, even in the classical case, a and $\neg\neg a$ are logically equivalent but not necessarily equal.

▶ **Example 2.** Any complete Boolean algebra $(\mathbb{B}, \preccurlyeq, \wedge, \vee, \neg)$ defines a CIS using the conjunction as tensor ($a \otimes b \triangleq a \wedge b$) and the negation for orthogonal map ($a^\perp \triangleq \neg a$).

▶ **Example 3.** A phase space [8] is defined by a commutative monoid M and a subset $\perp \subseteq M$. For $a \subseteq M$, define the dual $a^\perp \triangleq \{x : \forall y \in a, xy \in \perp\}$; define $a \otimes b \triangleq \{xy : x \in a, y \in b\}^{\perp\perp}$ for $a, b \subseteq M$. Then the set of subsets $a \subseteq M$ such that $a^{\perp\perp} = a$ forms a CIS.

Given a conjunctive structure $(\mathbb{C}, \preccurlyeq, \otimes, (\cdot)^\perp)$, we define the usual connectives and quantifiers of multiplicative linear logic (MLL) on $\mathbb{C}$ as follows, where F is a function over $\mathbb{C}$:

$$a \parr b \triangleq (a^\perp \otimes b^\perp)^\perp \qquad a \multimap b \triangleq (a \otimes b^\perp)^\perp \qquad \exists F \triangleq \bigcurlyvee_{a \in \mathbb{C}} F(a) \qquad \forall F \triangleq \bigcurlywedge_{a \in \mathbb{C}} F(a)$$

These definitions induce a canonical interpretation of MLL formulas within any CIS.

▶ **Definition 4.** *Consider a unitary CS $(\mathbb{C}, \preccurlyeq, \otimes, (\cdot)^\perp, 1)$. The interpretation of closed MLL formulas with parameters $P \in \mathbb{C}$ is defined as follows:*

$$\begin{array}{lll} [\![\mathbb{1}]\!] \triangleq 1 & [\![A \otimes B]\!] \triangleq [\![A]\!] \otimes [\![B]\!] & [\![\exists X.A]\!] \triangleq \exists\big(P \mapsto [\![A]\!]\{X := P\}\big) \\ [\![\perp]\!] \triangleq 1^\perp & [\![A \parr B]\!] \triangleq [\![A]\!] \parr [\![B]\!] & [\![\forall X.A]\!] \triangleq \forall\big(P \mapsto [\![A]\!]\{X := P\}\big) \\ [\![P]\!] \triangleq P & & \end{array}$$

Given a sequent $\vdash A_1, \dots, A_k$, the interpretation $[\![A_1, \dots, A_k]\!]$ is defined as $[\![A_1]\!]$ if $k = 1$ and $[\![A_1]\!] \parr [\![A_2, \dots, A_k]\!]$ otherwise.

▶ Remark 5. As usual, the interpretation $[\![A]\!]$ of a formula A depends only upon the assignment of the free variables of A. The same is valid for sequents.

▶ Remark 6. If $\mathbb{C}$ is a CIS, the involutivity of $(\cdot)^\perp$ actually implies that $(\mathbb{C}, \preccurlyeq, \multimap)$ is an implicative structure in the sense of Miquel [15] since $\multimap$ satisfies the expected variance and continuity properties: in particular $\big(\bigcurlyvee_{b \in B} b\big) \multimap a = \bigcurlywedge_{b \in B}(b \multimap a)$ for any $a \in \mathbb{C}$ and $B \subseteq \mathbb{C}$.

Note that this applies to *structures*, which only define operators and their relationships. The *algebras* defined below, which include a notion of logical validity, will actually diverge since implicative algebras model intuitionistic logic, while our algebras apply to linear logic.

2.2 Separators and internal logic

Conjunctive structures are suited to interpret formulas of MLL. To account for a validity notion, we need to introduce the notion of separator [15, 16], a subset of the structure which intuitively distinguishes valid formulas, similarly to a filter in a Boolean algebra. Separators are built upon a set of combinators that one can understand as axioms for the internal logic.

▶ **Definition 7.** *Consider a unitary CS $(\mathbb{C}, \preccurlyeq, \curlyvee, \otimes, (\cdot)^\perp, 1)$. The MLL combinators are:*

- $S_3 \triangleq \bigcurlywedge_{a,b \in \mathbb{C}} (a \otimes b) \multimap (b \otimes a)$
- $S_4 \triangleq \bigcurlywedge_{a,b,c \in \mathbb{C}} (a \multimap b) \multimap (a \otimes c) \multimap (b \otimes c)$
- $S_5 \triangleq \bigcurlywedge_{a,b,c \in \mathbb{C}} ((a \otimes b) \otimes c) \multimap (a \otimes (b \otimes c))$
- $S_6 \triangleq \bigcurlywedge_{a \in \mathbb{C}} a \multimap (1 \otimes a)$
- $S_7 \triangleq \bigcurlywedge_{a \in \mathbb{C}} (1 \otimes a) \multimap a$
- $S_8 \triangleq \bigcurlywedge_{a,b \in \mathbb{C}} (a \multimap b) \multimap (b^\perp \multimap a^\perp)$

The definitions (and names) for S_3, S_4 and S_5 come from the notion of separator in conjunctive algebras [16]. The original S_1 and S_2 are replaced here by S_6, S_7 and S_8 to account for linearity. The set of combinators $\{S_3, \dots, S_8\}$ is known to be complete for MLL [1].

$$\frac{}{\vdash 1 : \mathbb{1}}\,(\mathbb{1}) \qquad \frac{}{\vdash \mathbf{I} : A^{\perp}, A}\,(\textsc{Ax}) \qquad \frac{\vdash a : A_1, \dots, A_k}{\vdash \mathbf{ex}(\sigma) * a : A_{\sigma(1)}, \dots, A_{\sigma(k)}}\,(\textsc{Ex}) \qquad \frac{\vdash a : \Gamma, A \quad \vdash b : B, \Delta}{\vdash \mathbf{t} * a * b : \Gamma, A \otimes B, \Delta}\,(\otimes)$$

$$\frac{\vdash a : \Gamma, A \quad \vdash b : A^{\perp}, \Delta}{\vdash \mathbf{c} * a * b : \Gamma, \Delta}\,(\textsc{Cut}) \qquad \frac{\vdash a : \Gamma, A\{X := B\}}{\vdash a : \Gamma, \exists X.A}\,(\exists) \qquad \frac{\vdash a : \Gamma, A \quad X \text{ not free in } \Gamma}{\vdash a : \Gamma, \forall X.A}\,(\forall)$$

Figure 1 Semantic typing rules for MLL.

▶ **Definition 8.** *Let us consider* $(\mathbb{C}, \preccurlyeq, \curlyvee, \otimes, (\cdot)^{\perp}, 1)$ *a unitary CIS. A* monoidal separator *on* $\mathbb{C}$ *is an upwards closed set* $\mathcal{S} \subseteq \mathbb{C}$ *such that:*

- $\mathcal{S}$ *contains the MLL-combinators* $S_3, \dots, S_8$*.*
- *For any* $a, b \in \mathbb{C}$*, if* $a \multimap b \in \mathcal{S}$ *and* $a \in \mathcal{S}$ *then* $b \in \mathcal{S}$*.*

A conjunctive involutive monoidal algebra (CIMA) is a unitary CIS $\mathbb{C}$ *together with a monoidal separator* $\mathcal{S}$*.*

▶ **Example 9.** In any CIS induced by a complete Boolean algebra $\mathbb{B}$ (see Example 2), all the combinators are tautologies trivially interpreted by the maximal element $\top$. Therefore, the singleton $\{\top\}$ (or alternatively any filter on $\mathbb{B}$) defines a separator for this CIS.

▶ **Definition 10.** *Let* $\mathbb{C}$ *be a CS. A* semantic judgement *is a statement* $\vdash a : \Gamma$ *where* $a \in \mathbb{C}$ *and* Γ *is a sequent of formulas with parameters in* $\mathbb{C}$*. Such a judgement is* sound *if* $a \preccurlyeq [\![\Gamma]\!]$*. A* semantic typing rule *is an inference rule where the premises and conclusion are semantic judgements, possibly involving free variables. Such a rule is* sound *if the soundness of its premises entails that of its conclusion for any instantiation of the variables in* $\mathbb{C}$*.*

We provide semantic typing rules for MLL in Figure 1, Using the following terms, where $\sigma : [1;k] \to [1;k]$ is a permutation and $a * b \triangleq \curlywedge\{c \in \mathbb{C} \mid a \preccurlyeq b \multimap c\}$ is the usual application from implicative algebras:

$$\begin{aligned}
\mathbf{I} &\triangleq \curlywedge_{a \in \mathbb{C}} (a \multimap a) \\
\mathbf{t} &\triangleq \curlywedge_{a,b,g,d \in \mathbb{C}} ((g \parr a) \multimap (b \parr d) \multimap g \parr ((a \otimes b) \parr d)) \\
\mathbf{c} &\triangleq \curlywedge_{a,g,d \in \mathbb{C}} ((g \parr a) \multimap (a^{\perp} \parr d) \multimap (g \parr d)) \\
\mathbf{ex}(\sigma) &\triangleq \curlywedge_{a_1,\dots,a_k \in \mathbb{C}} \big((a_1 \parr \cdots \parr a_k) \multimap (a_{\sigma(1)} \parr \cdots \parr a_{\sigma(k)})\big)
\end{aligned}$$

▶ **Proposition 11.** *For any CIMA* $(\mathbb{C}, \mathcal{S})$ *and any permutation* σ*, the combinators* $\mathbf{I}$*,* $\mathbf{t}$*,* $\mathbf{c}$ *and* $\mathbf{ex}(\sigma)$ *belong to the separator* $\mathcal{S}$

Proof. These proofs are essentially simple combinatorial manipulations within conjunctive involutive structures. We detail here the proof that $\mathbf{I} \in \mathcal{S}$ as an example. Let us consider a CIMA $(\mathbb{C}, \mathcal{S})$ and define:

$$S_4' \triangleq \underset{a,b,c \in \mathbb{C}}{\curlywedge} (a \multimap b) \multimap (b \multimap c) \multimap a \multimap c \qquad\qquad t \circ s \triangleq S_4' * s * t$$

It is an easy exercise to check that $S_4' \in \mathcal{S}$ (this follows from combinatorial manipulations using the closure under modus ponens and the fact that $S_4, S_8 \in \mathcal{S}$). Besides, similarly to what happens for implicative algebras, separators of CIMAs are closed under application $*$, and as such they are also closed under composition $\circ$. To prove that $\mathbf{I} \in \mathcal{S}$, observe that for any $a \in \mathbb{C}$ we have $S_6 \preccurlyeq a \multimap a \otimes \mathbf{1}$ and $S_7 \preccurlyeq a \otimes \mathbf{1} \multimap a$. Then $S_7 \circ S_6 \preccurlyeq a \multimap a$ (uniformly on $a \in \mathbb{C}$) and thus we get $S_7 \circ S_6 \in \mathcal{S}$ and then $\curlywedge_{a \in \mathbb{C}} a \multimap a \in \mathcal{S}$. ◀

▶ **Theorem 12.** *The semantic typing rules of Figure 1 are sound for all unitary CS. Moreover, for any CIMA $(\mathbb{C}, \mathcal{S})$ if $\vdash a : \Gamma$ is provable, then $a \in \mathcal{S}$.*

Proof. The soundness of rule ($\mathbb{1}$) is just reflexivity of $\preccurlyeq$ and that of rules (Ax), ($\forall$) and ($\exists$) holds by definition of the meet operation. For the other rules, remark that if for some $a, b, t, u \in \mathbb{C}$ we have $t \preccurlyeq a \multimap b$ and $u \preccurlyeq a$, then $t \preccurlyeq u \multimap b$, hence $t * u \preccurlyeq b$ by definition of $*$. This entails the soundness of (Ex), ($\otimes$) and (Cut) by definition of $\mathbf{I}$, $\mathbf{t}$ and $\mathbf{c}$.

Besides, by definition of $*$, we also have $a \preccurlyeq b \multimap a * b$ for all a and b, which implies that separators are closed under $*$. This, with the results of Proposition 11, entails that the left-hand sides of derivable judgements are always in $\mathcal{S}$. ◀

As a consequence, we conclude that all provable MLL formulas are interpreted by elements in the separator (i.e. validated by the model).

As stated in Remark 6, unitary CISs are implicative structures while in general CIMAs are not implicative algebras. Due to the chosen notion of separator, CIMAs do not (in general) model intuitionistic logic. Nevertheless, some specific CIMAs can model intuitionistic or classical logics if their separators contains non linear terms.

3 Processes with global fusions

In this section, we define a computational structure that will serve as our reference. We first recall the basic definitions of a standard π-calculus, then we build processes with fusions as a formal extension of this language. These fusions, being an additional structure to extend the expressiveness of an existing algebra (as opposed to a feature that we would impose on the underlying calculus, which would restrict the range of calculi our study applies to), bring the necessary connections to realize MLL axioms.

Hence, it should be made clear that the point of this section is not to define “yet another π-calculus” but, on the contrary, to show that our study applies to any process calculus. The only technical constraint is that the combinators from Definition 7 should not be degenerate (i.e. equal to $\bot$), otherwise there is no meaningful seperator. This essentially requires to have realizers that act as substitutions (or “forwarders”). The construction below consists in formally completing a process algebra with such objects.

So the choice of a particular variant of the π-calculus is essentially arbitrary, we pick this one for the purpose of illustration. The only features we need are the ability to perform renaming and the availability of parallel composition and hiding with usual properties. The point of the construction is to extend the language with name fusions at top-level while preserving this composition structure. Keeping fusions at top-level is the only generic choice that does not impose to reason on the syntax of the calculus we build on, thus allowing for working with fusions in the algebraic structure without imposing constraints on the underlying process language.

3.1 Basic processes

The following recalls the standard definitions of the polyadic asynchronous π-calculus with its operational semantics by reduction [21]. We leave out replication and addition because these will not be used in this paper, but including them is not a problem.

▶ **Definition 13.** *The set Π of π-processes is defined by the following grammar:*

$$P, Q ::= \mathbb{1} \mid (P \mid Q) \mid u(\vec{x}).P \mid \bar{u}\,\langle \vec{v} \rangle \mid (\nu y)P$$

where $\vec{x}$ is a finite sequence of pairwise distinct names that range over the set $\mathbb{N}$ of natural numbers, $\vec{v}$ is a finite sequence of names, y and u are names. The term $\mathbb{1}$ is the unit *and the operators $|, u(\vec{x}), \bar{u}\langle v\rangle, (\nu y)$ are respectivelly called* parallel composition, reception, emission *and* hiding.

The names in $\vec{x}$ are bound in $u(\vec{x}).P$, the name y is bound in $(\nu y)P$. The set of free names *of a term P is denoted by $\mathrm{FN}(P)$. Terms are considered up to renaming of bound names, also called α-conversion, written $\equiv_\alpha$.*

Structural equivalence *is the smallest congruence over terms such that*

- $(\Pi, |, \equiv)$ *is an commutative monoid, i.e.* $\mathbb{1}|P \equiv P$, $P|Q \equiv Q|P$ *and* $(P|Q)|R \equiv P|(Q|R)$,
- $P \equiv (\nu x)P$ *and* $P \mid (\nu x)Q \equiv (\nu x)(P \mid Q)$ *whenever* $x \notin \mathrm{FN}(P)$,
- $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$ *and we write* $(\nu xy)P \triangleq (\nu x)(\nu y)P$.

▶ **Definition 14.** *A* substitution *is a function $\sigma : \mathbb{N} \to \mathbb{N}$. Provided that the names in $\vec{x}$ are pairwise distinct and that $\vec{x}$ and $\vec{v}$ have the same length, $\{\vec{x} := \vec{v}\}$ denotes the substitution which replaces $\vec{x}$ by $\vec{v}$ and leaves all other names unchanged.*

Given $X \subseteq \mathbb{N}$ we denote by $\sigma_{\restriction X}$ the substitution which coincides with σ on X and is the identity outside. We write $\sigma \backslash X$ for the substitution $\sigma_{\restriction X^c}$.

We denote by P^σ the result of applying σ to $P \in \Pi$, defined inductively in the standard way. Given substitutions $\sigma_1, \ldots, \sigma_k$ we denote by $P^{\sigma_1 \cdots \sigma_k}$ the term $(\ldots(P^{\sigma_1})\ldots)^{\sigma_k}$.

▶ **Definition 15.** *One-step reduction is the smallest binary relation $\longrightarrow_1$ over Π such that*

- $\bar{u}\langle\vec{v}\rangle \mid u(\vec{x}).P \longrightarrow_1 P^{\{\vec{x}:=\vec{v}\}}$,
- $\longrightarrow_1$ *is compatible with $\equiv$, i.e. if $P \equiv P'$, $Q \equiv Q'$ and $P \longrightarrow_1 Q$ then $P' \longrightarrow_1 Q'$,*
- $\longrightarrow_1$ *is compatible with parallel composition and hiding, i.e. if $x \in \mathbb{N}$ and $P \longrightarrow_1 P'$, then $P \mid Q \longrightarrow_1 P' \mid Q$ and $(\nu x)P \longrightarrow_1 (\nu x)P'$.*

We do not define a higher-level semantics at this point. The semantics induced by the conjunctive structures defined in the following sections will be a generic form of testing and most of the construction will be parametric in the particular testing protocol.

3.2 Processes with fusions

A *fusion* is, intuitively, a connection between channels. The defining property of a fusion $u \leftrightarrow v$ is that its presence within a process allows actions on channel u to synchronize with actions on channel v. Several works have defined extensions of the π-calculus with this feature [7, 19, 23] and studied its effects on the theory and the expressiveness of the language.

Instead, the construction presented in this section rather aims at extending an arbitrary process calculus with fusions *without* affecting its theory, for our study to be compatible with any particular process language. This is made possible by the fact that we only need fusions at top-level, so that they can live side-by-side with standard processes. Intuitively, a fusion $u \leftrightarrow v$ at top level in a process can be interpreted as the information that u and v will eventually get substituted with the same name.

▶ **Definition 16.** *A fusion is an equivalence relation over $\mathbb{N}$. The set of fusions is written $\mathcal{E}$. Given $e \in \mathcal{E}$ we write $x \underset{e}{\sim} y$ instead of $(x, y) \in e$. For $e, f \in \mathcal{E}$ we write ef for the smallest equivalence that contains both e and f.*

▶ Remark 17. The set $\mathcal{E}$ is a complete lattice under inclusion, with the identity $\Delta_\mathbb{N} \triangleq \{(x, x) \mid x \in \mathbb{N}\}$ as minimum and the full relation $\nabla_\mathbb{N} \triangleq \mathbb{N} \times \mathbb{N}$ as maximum. The meet of a set $S \subseteq \mathcal{E}$ is the intersection $\bigcap S$ and its join $\bigvee S$ is the transitive closure of the union $\bigcup S$.

▶ **Definition 18.** *Given $e \in \mathcal{E}$, $x, y \in \mathbb{N}$, $X \subseteq \mathbb{N}$ and $\sigma : \mathbb{N} \to \mathbb{N}$, define*

- *the* equivalence class *of x as $[x]_e \triangleq \{y \in \mathbb{N} \mid x \underset{e}{\sim} y\}$, extended as $[X]_e \triangleq \bigcup_{x \in X}[x]_e$;*
- *the* domain *of e as $|e| \triangleq \{x \in \mathbb{N} \mid [x]_e \neq \{x\}\}$;*
- *the* restriction *of e to X as $e \cap X \triangleq (e \cap (X \times X)) \cup \Delta_{\mathbb{N}}$;*
- *the* hiding *of X in e as $e \setminus X \triangleq e \cap X^c$;*
- *the* elementary fusion *of x and y as $x{\leftrightarrow}y \triangleq \{(x, y), (y, x)\} \cup \Delta_{\mathbb{N}}$.*

An equivalence e can be induced by a function σ, considering that equivalent elements are the ones with the same image under σ. Such a function can always be deduced from e by picking a representative in each equivalence class. The following definitions formalize these constructions, exploiting the well-ordering over $\mathbb{N}$ to get canonical representatives.

▶ **Definition 19.** *Given a fusion $e \in \mathcal{E}$, $x \in \mathbb{N}$ and a function $\tau : \mathbb{N} \to \mathbb{N}$, we define:*

- $x^{\bullet}_e \triangleq \min[x]_e$ *and* $x^{*}_e \triangleq \begin{cases} x & \textit{if } [x]_e = \{x\} \\ \min([x]_e \backslash \{x\}) & \textit{if } [x]_e \neq \{x\} \end{cases}$
- *$\sigma^{\bullet}_e : x \mapsto x^{\bullet}_e$ for all x, the substitution induced by e;*
- *$\varepsilon_\tau \triangleq \bigvee_{x \in \mathbb{N}}(x{\leftrightarrow}\tau(x))$, the fusion induced by τ;*
- *$e^\tau \triangleq \bigvee_{x \underset{e}{\sim} y}(\tau(x){\leftrightarrow}\tau(y))$, the composition of e with τ.*

We are now ready to introduce the notion of processes with global fusions, and extend the constructions on processes to this generalization. This is similar to that of unitization in algebras, where a non-unital algebra is extended with a formal extra element and saturated to preserve the structure and endow the extra element with desired properties.

▶ **Definition 20.** *The set of* processes with global fusions *(PGF) is defined as*

$$\bar{\Pi} \triangleq \Pi \times \mathcal{E} = \{(P, e) \mid P \in \Pi, e \in \mathcal{E}\}$$

Over $\bar{\Pi}$, define substitution, free names, α-equivalence and structural equivalence $\equiv$ as

$$\begin{aligned} (P, e)^\tau &\triangleq (P^\tau, e^\tau) & (P, e) \equiv_\alpha (Q, f) &\iff e = f \textit{ and } P^{\sigma^{\bullet}_e} \equiv_\alpha Q^{\sigma^{\bullet}_f} \\ \mathrm{FN}(P, e) &\triangleq \mathrm{FN}(P) \cup |e| & (P, e) \equiv (Q, f) &\iff e = f \textit{ and } P^{\sigma^{\bullet}_e} \equiv Q^{\sigma^{\bullet}_f} \end{aligned}$$

When it is not necessary to specify process and fusion, we will use the lowercase letters p, q, r, s to refer to processes with fusions.

Remark that the definitions of α-equivalence and structural equivalence allow renaming of free names as long as the equivalence class of each name is unchanged, so that we have for instance $(a(x).P, a{\leftrightarrow}b) \equiv (b(x).P, a{\leftrightarrow}b)$. Including $|e|$ in $\mathrm{FN}(P, e)$ ensures that, despite such renaming, FN is invariant under $\equiv_\alpha$ as expected. Note also that if we identify each term P with the pair $(P, \Delta_{\mathbb{N}})$, then the operations defined above coincide with those of Π.

▶ **Definition 21.** *One-step reduction in $\bar{\Pi}$ is the relation $\longrightarrow_1$ such that $(P, e) \longrightarrow_1 (Q, f)$ if and only if $e = f$ and $P^{\sigma^{\bullet}_e} \longrightarrow_1 Q^{\sigma^{\bullet}_f}$. Multistep reduction $\longrightarrow$ is the reflexive and transitive closure of $\longrightarrow_1$.*

Again, this extends the associated relation from Π to $\bar{\Pi}$ and allows more reductions when the fusion relates two names on which compatible actions occur. For instance we have $(u(x).P \mid \bar{v}\,\langle y\rangle\,, u{\leftrightarrow}v) \longrightarrow_1 (P\{x := y\}, u{\leftrightarrow}v)$.

▶ **Definition 22.** *For $(P, e), (Q, f) \in \bar{\Pi}$, define $(P, e) \mid (Q, f) \triangleq (P \mid Q, ef)$.*

Identifying a pure fusion e with the pair $(\mathbb{1}, e)$, we have $(P, e) \equiv P \mid e$ for every P, since $P \mid \mathbb{1} \equiv P$ in Π. So in this sense Π and $\mathcal{E}$ generate $\bar{\Pi}$ under parallel composition. Besides, reduction $\longrightarrow_1$ is compatible with $\equiv$ and parallel composition, as in the basic calculus.

We now need to extend name hiding to $\bar{\Pi}$. The result of $(\nu x)(P, e)$ must be (Q, f) where $x \notin \mathrm{FN}(Q, f)$, hence $x \notin \mathrm{FN}(Q) \cup |f|$. Consider a pair $(\nu x)(\bar{x}, x{\leftrightarrow}y)$: even if x must be bound in $(\nu x)(\bar{x}, x{\leftrightarrow}y)$, since x is fused with y, we have $(\bar{x}, x{\leftrightarrow}y) \equiv_\alpha (\bar{y}, x{\leftrightarrow}y)$ so the only reasonable definition for $(\nu x)(\bar{x}, x{\leftrightarrow}y)$ is $(\bar{y}, \Delta_{\mathbb{N}})$, which is structurally equivalent to $((\nu x)\bar{y}, \Delta_{\mathbb{N}})$. On the other hand, if we consider $(\nu x)(\bar{x}, \Delta_{\mathbb{N}})$, we expect to get simply $((\nu x)\bar{x}, \Delta_{\mathbb{N}})$. Definition 19 provides the notation x^*_e to represent this in a general way.

▶ **Definition 23.** *For $(P, e) \in \bar{\Pi}$ and $x \in \mathbb{N}$, define $(\nu x)(P, e) \triangleq \big((\nu x)P\{x := x^*_e\}, e \setminus \{x\}\big)$.*

▶ Remark 24. Whenever x is not equivalent to another name, *i.e.* $[x]_e = \{x\}$, we have $x = x^*_e$ and therefore the binder νx on a PGF process only computes as expected on the process side: $(\nu x)(P, e) = ((\nu x)P, e)$. If, instead, x is equivalent to other names, *i.e.* $[x]_e \neq \{x\}$, then $x \neq x^*_e$ and νx will disconnect x from its fusion-side equivalents, while replacing it by an equivalent name on the process side: $((\nu x)P\{x := x^*_e\}, e\backslash\{x\}) \equiv (P\{x := x^*_e\}, e\backslash\{x\})$.

To extend the definition of ν to finite sets, we first need to ensure that, up to α-equivalence, the order of application of ν is irrelevant.

▶ **Lemma 25.** *Consider $(P, e) \in \bar{\Pi}$ and $x, y \in \mathbb{N}$, then $(\nu x)(\nu y)(P, e) \equiv_\alpha (\nu y)(\nu x)(P, e)$.*

Proof. On the one hand

$$(\nu x)(\nu y)(P, e) = (\nu x)((\nu y)P\{y := y^*_e\}, e\backslash\{y\}) = ((\nu xy)P\{y := y^*_e\}\{x := x^*_{e\backslash\{y\}}\}, e\backslash\{x, y\}).$$

On the other hand, it can be shown that the last process is α-equivalent to

$$((\nu yx)P\{x{:=}x^*_e\}\{y{:=}y^*_{e\backslash\{x\}}\}, e\backslash\{x, y\}) = (\nu y)((\nu x)P\{x{:=}x^*_e\}, e\backslash\{x\}) = (\nu y)(\nu x)(P, e). \blacktriangleleft$$

▶ **Definition 26.** *Given a* finite *set of names $X = \{x_1, \ldots, x_k\}$ with $x_1 < \cdots < x_k$, define $(\nu X)(P, e) \triangleq (\nu x_k) \cdots (\nu x_1)(P, e)$.*

Later on, we will need to extend this definition for a possibly infinite set X. Even if, for any PGF p, there exist only finitely many free names of p in X which we need to hide, for each of them we need to treat them differently depending on whether their equivalence class is included in X or not. The following definitions generalize the unary definition of ν in that regards.

▶ **Definition 27.** *Consider a set $X \subseteq \mathbb{N}$ and a fusion $e \in \mathcal{E}$. Define the substitution*

$$\mu_{(X,e)}(x) \triangleq \begin{cases} x^\bullet_e & \textit{if } [x]_e \subseteq X \\ \min([x]_e \setminus X) & \textit{if } [x]_e \setminus X \neq \emptyset \end{cases}$$

▶ **Definition 28.** *Let $(P, e) \in \bar{\Pi}$ and $X \subseteq \mathbb{N}$. Since $\mathrm{FN}(P)$ is finite, there exists classes $[x_1]_e, \ldots, [x_{h+k}]_e$ in the quotient $\mathbb{N}/e$ such that for each $i \leq h$ we have $[x_i]_e \subseteq X$ and for each $j > h$ we have $[x_j] \setminus X \neq \emptyset$, and moreover $\mathrm{FN}(P) \cap X \subseteq \bigcup_{i=1}^{h+k} [x_i]_e$. Define*

$$(\nu X)(P, e) \triangleq \big((\nu x^\bullet_{1,e}, \ldots, x^\bullet_{h,e})P^{\mu_{(X,e)}}, e \setminus X\big)$$

Let us denote by $\bar{\nu}$ the binder $(\nu\mathbb{N})$.

Finally, let us state a technical result that emphasizes how we can embed a substitution within the calculus by means of the corresponding fusion.

▶ **Proposition 29.** *Consider a substitution $\tau : X \to X^c$ for a set $X \subset \mathbb{N}$ and $p, q \in \bar{\Pi}$ such that $\mathrm{FN}(p) \subseteq X$ and $\mathrm{FN}(q) \subseteq X^c$. Then $(\nu X)(p \,|\, q \,|\, \varepsilon_\tau) \equiv_\alpha p^\tau \mid q$.*

We defer the proof to the extended version of this paper, as it requires a number of technical lemmas whose statements and proofs are of little interest by themselves. The property above will be a crucial ingredient later on to build appropriate realizers. Obtaining it is the point of the PGF construction and we will not rely on more details when studying the conjunctive structure.

4 Conjunctive structure of processes

Following the first author's work [2], we shall now see how to define realizability models which interpret MLL formulas as sets of processes. These models actually induce a conjunctive involutive monoidal algebra and as such provides a sound interpretation of MLL. In particular, we would like to emphasize that this illustrates a significant benefit of our approach, which allows us to isolate logical properties that are shared by *all* concurrent realizability models sharing this algebraic structure, ensuring that these properties are indeed insensitive to implementation details such as the choice of a particular syntax for processes.

4.1 Concurrent realizability with PGF

We build upon the main insight of Krivine's realizability [14], by parameterizing the interpretation by a *pole*, a set of processes somewhat characterizing valid interactions between terms. In this setting, we will consider poles that are simply closed under structural equivalence to characterize the soundness of interaction with respect to the renaming mechanism that is necessary for pure parallel composition, without focusing so far on the reduction of processes. From now on, we denote by $\mathbb{P}$ the set of all sets of processes $\mathbb{P} \triangleq \mathcal{P}(\bar{\Pi})$.

▶ **Definition 30.** *A* pole *is a set of closed processes $⫫ \subseteq \{p \in \bar{\Pi} \mid \mathrm{FN}(p) = \emptyset\}$ that is closed under $\equiv$,* i.e. *if $p \equiv q$ and $q \in ⫫$ then $p \in ⫫$.*

Given a pole $⫫$, we say that two processes p, q are *orthogonal*, which we write $p \perp q$, whenever $\bar{\nu}(p \,|\, q) \in ⫫$. This naturally induces an orthogonal operator $(\cdot)^\perp : \mathbb{P} \to \mathbb{P}$ on sets of processes by defining $A^\perp \triangleq \{p \in \bar{\Pi} : \forall q {\in} A,\ p {\perp} q\}$. This operator satisfies the usual property of orthogonality.

▶ **Proposition 31.** *For all $A, B \in \mathbb{P}$ and for any non-empty $\mathfrak{B} \subseteq \mathbb{P}$ we have:*

1. *If $A \subseteq B$ then $B^\perp \subseteq A^\perp$*
2. *$A \subseteq A^{\perp\perp}$ and $A^\perp = A^{\perp\perp\perp}$*
3. *$\left(\bigcup_{X \in \mathfrak{B}} X\right)^\perp = \bigcap_{X \in \mathfrak{B}} X^\perp$ and if $\mathfrak{B} \neq \emptyset$ then $\left(\bigcap_{X \in \mathfrak{B}} X\right)^\perp \supseteq \left(\bigcup_{X \in \mathfrak{B}} X^\perp\right)$*
4. *$\left(\bigcup_{X \in \mathfrak{B}} X^{\perp\perp}\right)^\perp = \left(\bigcup_{X \in \mathfrak{B}} X\right)^\perp$*

Proof. Part 1, 2, 3 directly follow from the definition, in particular $(\bar{\Pi}, \bar{\Pi}, \perp)$ where $\perp = \{(p, q) \mid p \perp q\}$ is called a *polarity* in [6] and it defines a Galois connection [3]. For part 4, it suffices to see that $\left(\bigcup_{X \in \mathfrak{B}} X^{\perp\perp}\right)^\perp = \bigcap_{X \in \mathfrak{B}} X^{\perp\perp\perp} = \bigcap_{X \in \mathfrak{B}} X^\perp = \left(\bigcup_{X \in \mathfrak{B}} X\right)^\perp$. ◀

Our realizability interpretation is inspired by the semantics of phases, in particular we interpret formulas in the set of *behaviors* $\mathbb{B} \triangleq \{A \in \mathbb{P} : A^{\perp\perp} = A\}$, *i.e.* the sets that are closed under double orthogonal. This set can be endowed with a structure of complete lattice, as the next proposition shows.

▶ **Proposition 32.** *Let us define* $\bigvee \mathfrak{B} \triangleq (\bigcup \mathfrak{B})^{\perp\perp}$ *for all* $\mathfrak{B} \subseteq \mathbb{B}$*. Then:*

1. $(\mathbb{P}, \bigcap, \bigcup, \subseteq)$ *is a complete lattice with top element* $\overline{\Pi}$ *and bottom element* $\emptyset$*.*
2. $(\mathbb{B}, \bigcap, \bigvee, \subseteq)$ *is a complete lattice with top element* $\overline{\Pi}$ *and bottom element* $\emptyset^{\perp\perp}$*.*

In order to interpret MLL formulas in this framework, the main idea consists in seeing the tensor $\otimes$ as defining the parallel composition of two processes in such a way that they do not communicate. This is achieved by making extensive use of substitutions to assign each process a disjoint space of names. Technically, we rely on the fact that $\mathbb{N}$ is in bijection with the set of even (or odd) natural numbers, somehow reflecting the tree structure of the formulas into the space of names.

▶ **Definition 33.** *We fix injections* $\iota_1, \iota_2 : \mathbb{N} \to \mathbb{N}$ *such that*[1] $\iota_1(\mathbb{N}) \cap \iota_2(\mathbb{N}) = \emptyset$ *and* $\iota_1(\mathbb{N}) \cup \iota_2(\mathbb{N}) = \mathbb{N}$*. We use the following notations:* $\mathbb{N}^i \triangleq \iota_i(\mathbb{N})$*,* $n.i := \iota_i(n)$*,* $p^i \triangleq p^{\iota_i}$ *and* $e^i \triangleq e^{\iota_i}$*. Given* $p \in \overline{\Pi}$ *s.t.* $\mathrm{FN}(p) \subseteq \mathbb{N}^i$ *we denote as* p^{-i} *the process* $p^{\iota_i^{-1}}$*. Similarly, if* e *is a fusion s.t.* $|e| \subseteq \mathbb{N}^i$*, we denote as* e^{-i} *the fusion* $e^{\iota_i^{-1}}$*.*

We can now define the semantic counterpart of MLL connectives on processes, which can then lift to behaviors.

For technical purposes, we also define an operator $\cdot * \cdot$, which in fact coincides with the usual application, *i.e.* the left adjoint of the linear implication $\multimap$ as Proposition 37 shows.

▶ **Definition 34.** *For all* $p = (P, e)$ *and* $q = (Q, f)$ *define the following operations*[2]*:*

- $p \bullet q \triangleq p^1 \mid q^2 = (P^1|Q^2, e^1 f^2)$
- $p * q \triangleq ((\nu \mathbb{N}^1)(p \mid q^1))^{-2} = ((\nu \mathbb{N}^1)(P|Q^1, ef^1))^{-2}$.

▶ **Definition 35.** *We define the following operations on* $\mathbb{P}$*:*

$$
\begin{array}{rcl|rcl}
 & & & A \otimes B & := & (A \bullet B)^{\perp\perp} \\
1 & := & \{(\mathbb{1}, \Delta_{\mathbb{N}})\}^{\perp\perp} & A \parr B & := & (A^{\perp} \otimes B^{\perp})^{\perp} \\
A \bullet B & := & \{p \bullet q \mid p \in A,\ q \in B\} & A \multimap B & := & (A \otimes B^{\perp})^{\perp} \\
A \mid B & := & \{p \mid q \mid p \in A,\ q \in B\} & A \parallel B & := & (A \mid B)^{\perp\perp} \\
A * B & := & \{p * q \mid p \in A,\ q \in B\}^{\perp\perp} & A \upharpoonright X & := & \{p \in A \mid \mathrm{FN}(p) \subseteq X\}
\end{array}
$$

Observe that the definitions of $1, *, \otimes, \parr, \multimap, \parallel$ define behaviors, and besides, they are insensitive to double orthogonal:

▶ **Proposition 36.** *For any* $A, B \in \mathbb{P}$*, it holds that* $(A \bullet B)^{\perp} = (A^{\perp\perp} \bullet B^{\perp\perp})^{\perp}$*. As a consequence,* $A \otimes B = A^{\perp\perp} \otimes B^{\perp\perp}$*,* $A \parr B = A^{\perp\perp} \parr B^{\perp\perp}$ *and* $A \multimap B = A^{\perp\perp} \multimap B^{\perp\perp}$*.*

Proof. The inclusion $(A^{\perp\perp} \bullet B^{\perp\perp})^{\perp} \subseteq (A \bullet B)^{\perp}$ is true by anti monotonicity of the orthogonal operator. For the reverse inclusion, let us consider $r \in (A \bullet B)^{\perp}$ and let us prove that $r \in (A^{\perp\perp} \bullet B^{\perp\perp})^{\perp}$. First, observe that for any processes $p, q,$, we have the following equivalences: $\bar{\nu}(p^1 \mid q^2 \mid r) \equiv \bar{\nu}(((\nu \mathbb{N}^2)(r \mid q^2))^{-1} \mid p) \equiv \bar{\nu}(((\nu \mathbb{N}^1)(r \mid p^1))^{-2} \mid q)$. In particular, for $p \in A$ and $q \in B$, we have $\bar{\nu}(((\nu \mathbb{N}^2)(r|q^2))^{-1}|p) \equiv \overline{\nu}(p^1|q^2|r) \in \perp\!\!\!\perp$. Then $((\nu \mathbb{N}^2)(r|q^2))^{-1} \in A^{\perp} = A^{\perp\perp\perp}$ and thus $\overline{\nu}(p^1 \mid q^2 \mid r) \in \perp\!\!\!\perp$ for all $p \in A^{\perp\perp}, q \in B$. Using the same observation, we get that $((\nu \mathbb{N}^1)(r \mid p^1))^{-2} \in B^{\perp\perp\perp}$ for all $p \in A^{\perp\perp}$. Finally, using this observation again we obtain that $\overline{\nu}(p^1 \mid q^2 \mid r) \in \perp\!\!\!\perp$ for all $p \in A^{\perp\perp}, q \in B^{\perp\perp}$, which is what we wanted to prove. ◀

[1] For instance, we can take $\iota_1(n) \triangleq 2n + 1$ and $\iota_2(n) \triangleq 2n$.

[2] Observe that $\mathrm{FN}((\nu \mathbb{N}^1)((P, e)|(Q, f)^1)) \subseteq \mathbb{N}^2$ and thus we can apply ι_2^{-1} to this process.

▶ **Proposition 37.** *Let us consider $A, B, C \in \mathbb{P}$. Then:*
1. $C \subseteq A \multimap B \iff C * A \subseteq B^{\perp\perp}$.
2. $C * A = \bigcap\{B \in \mathbb{B} \mid C \subseteq A \multimap B\} = \min\{B \in \mathbb{B} \mid C \subseteq A \multimap B\}$.

(In particular, if $B \in \mathbb{B}$ then $C \subseteq A \multimap B \iff C * A \subseteq B$).

Proof.
1. Observe that $A \multimap B = (A \bullet B^{\perp})^{\perp}$. Therefore, we have:

$$\begin{array}{rcl} C \subseteq (A \bullet B^{\perp})^{\perp} & \Leftrightarrow & \forall p \in C.\forall q \in A.\forall r \in B^{\perp}.\, p \perp q \bullet r \\ & \Leftrightarrow & \forall p \in C.\forall q \in A.\forall r \in B^{\perp}.\, \bar{\nu}(p|q^1|r^2) \in \perp\!\!\!\perp \\ & \Leftrightarrow & \forall p \in C.\forall q \in A.\forall r \in B^{\perp}.\, (\nu\mathbb{N}^2)\Big((\nu\mathbb{N}^1)(p|q^1)|r^2\Big) \in \perp\!\!\!\perp \\ & \Leftrightarrow & \forall p \in C.\forall q \in A.\forall r \in B^{\perp}.\, \bar{\nu}\Big((\nu\mathbb{N}^1)(p|q^1)^{-2}|r\Big) \in \perp\!\!\!\perp \\ & \Leftrightarrow & \forall p \in C.\forall q \in A.\forall r \in B^{\perp}.\, p * q \perp r \\ & \Leftrightarrow & C * A \subseteq B^{\perp\perp} \end{array}$$

2. Using Part 1, we get that $C * A \subseteq C * A \iff C \subseteq A \multimap (C * A)$ from which we deduce that $C * A \in \{B \mid C \subseteq A \multimap B\}$. On the other hand, if $C \subseteq A \multimap B$, then $C * A \subseteq B$. ◀

To complete the definition of the concurrent realizability interpretation of MLL using processes, it only remains to define the realizability relation.

▶ **Definition 38.** *We say that a PGF p* realizes *a behavior A, which we write $p \Vdash A$, if $p \in A$.*

A detailed study of this interpretation and of the computational content of realizers, which mostly ensure adequate communication between adequate channels using an appropriate fusion, is out of the scope of this paper. We simply illustrate how the identity can be realized.

▶ **Proposition 39.** *For any $A \in \mathbb{P}$, we have that $I \triangleq \bigvee\limits_{n\in\mathbb{N}} (n.1{\leftrightarrow}n.2) \Vdash A \multimap A$.*

Proof. We have $A \multimap A = (A \bullet A^{\perp})^{\perp}$ (using Proposition 37). Let us consider $p \in A$ and $q \in A^{\perp}$, it suffices to prove that $I \perp\!\!\!\perp p^1 \mid q^2$. Using Proposition 29, we get that $\bar{\nu}(p^1 \mid q^2 \mid I) \equiv_{\alpha} (\nu_2)(\nu_1)(p^1 \mid q^2 \mid I) \equiv_{\alpha} (\overline{\nu})(p \mid q)$ and we conclude by observing that $(\overline{\nu})(p \mid q) \in \perp\!\!\!\perp$. ◀

4.2 The induced conjunctive involutive monoidal algebra

Most importantly, we can prove that this construction actually defines a conjunctive involutive monoidal algebra. In particular, this entails (via Theorem 12) that it defines a valid interpretation of MLL. The proof is in two parts, we prove that the interpretation induces a CIS, then we show that the set of realized behaviors defines a valid separator.

▶ **Proposition 40.** *The tuple $(\mathbb{B}, \subseteq, \otimes, (\cdot)^{\perp})$, where $(\mathbb{B}, \subseteq)$ is the lattice defined in Proposition 32, is a conjunctive involutive structure.*

Proof. We have to check that the axioms of Definition 1 are satisfied. The variance properties and De Morgan law directly follow from the definition of $\mathbb{B}$, $\otimes$ and from Proposition 31. We prove the first distributivity law, the other one being similar. For any $\mathfrak{B} \subseteq \mathbb{B}$, we have:

$$\begin{aligned} A \otimes \bigvee_{B\in\mathfrak{B}} B \underset{\text{def } \vee}{=} A \otimes \Big(\bigcup_{B\in\mathfrak{B}} B\Big)^{\perp\perp} \underset{\text{Prop 36}}{=} A \otimes \bigcup_{B\in\mathfrak{B}} B \underset{\text{def } \otimes}{=} \Big(A \bullet \bigcup_{B\in\mathfrak{B}} B\Big)^{\perp\perp} \\ = \Big(\bigcup_{B\in\mathfrak{B}} (A \bullet B)\Big)^{\perp\perp} \underset{\text{Prop 31.4}}{=} \Big(\bigcup_{B\in\mathfrak{B}} (A \bullet B)^{\perp\perp}\Big)^{\perp\perp} \underset{\text{def } \otimes}{=} \Big(\bigcup_{B\in\mathfrak{B}} (A \otimes B)\Big)^{\perp\perp} \underset{\text{def } \vee}{=} \bigvee_{B\in\mathfrak{B}} (A \otimes B) \end{aligned}$$

◀

$$\frac{}{F(a,\mathbb{1}) \equiv_F a} \qquad \frac{}{F(\mathbb{1},a) \equiv_F a} \qquad \frac{}{F(a,b) \equiv_F F(b,a)} \qquad \frac{}{F(a,F(b,c)) \equiv_F F(F(a,b),c)}$$

$$\frac{a \equiv_F a' \quad b \equiv_F b'}{F(a,b) \equiv_F F(a',b')} \qquad \frac{a \equiv_F a' \quad b \equiv_F b'}{a \otimes b \equiv_F a' \otimes b'} \qquad \frac{a \equiv_F a'}{a^\perp \equiv_F a'^\perp} \qquad \frac{a \equiv_F a' \quad b \equiv_F b' \quad a \preccurlyeq b}{a' \preccurlyeq b'}$$

Figure 2 Axioms for the compositional structure associated to F.

▶ **Proposition 41.** *The following behaviors are realized by pure fusions:*

1. $\bigcap_{A\in\mathbb{B}} A \multimap A$.
2. $\bigcap_{A,B\in\mathbb{B}} (A\otimes B) \multimap (B\otimes A)$.
3. $\bigcap_{A,B,C\in\mathbb{B}} (A\multimap B)\multimap(B\multimap C)\multimap A\multimap C$.
4. $\bigcap_{A,B,C\in\mathbb{B}} ((A\otimes B)\otimes C) \multimap (A\otimes(B\otimes C))$.
5. $\bigcap_{A\in\mathbb{B}} A \multimap (1\otimes A)$.
6. $\bigcap_{A\in\mathbb{B}} (1\otimes A) \multimap A$.
7. $\bigcap_{A,B\in\mathbb{B}} (A\multimap B) \multimap (B^\perp \multimap A^\perp)$.

Proof. The first statement is Proposition 39, and the other statements follow similar ideas. We give here a realizer for the second statement as an example.

By Definition 35, we have $(A\otimes B) \multimap (B\otimes A) = \big((A\bullet B)\bullet(B\bullet A)^\perp\big)^\perp$. Consider $p\in(B\bullet A)^\perp$, $q\in A$ and $r\in B$, in particular, they satisfy $\overline{\nu}(r^1 \mid q^2 \mid p) \in \perp\!\!\!\perp$. By definition $q^{1.1} \mid r^{2.1} \mid p^2 \in (A\otimes B) \multimap (B\otimes A)$. To get a realizer, we define the substitution $\tau : \mathbb{N}^1 \to \mathbb{N}^2$ s.t. $\tau(n.1.1) := n.2.2$ and $\tau(n.2.1) := n.1.2$. It is now enough to prove that $\varepsilon_\tau \perp\!\!\!\perp q^{1.1} \mid r^{2.1} \mid p^2$, i.e.: $\overline{\nu}(q^{1.1} \mid r^{2.1} \mid p^2 \mid \varepsilon_\tau) \in \perp\!\!\!\perp$. Moreover $\overline{\nu}(q^{1.1} \mid r^{2.1} \mid p^2 \mid \varepsilon_\tau) \equiv (\nu\mathbb{N}^2)(\nu\mathbb{N}^1)(q^{1.1} \mid r^{2.1} \mid p^2 \mid \varepsilon_\tau) \equiv (\nu\mathbb{N}^2)(q^{2.2} \mid r^{1.2} \mid p^2) \equiv \overline{\nu}(r^1 \mid q^2 \mid p) \in \perp\!\!\!\perp$, which ends the proof. ◀

▶ **Proposition 42.** *The set of non-empty behaviors* $\mathcal{S}_\mathbb{B} \triangleq \mathbb{B}\backslash\emptyset$ *defines a separator for the conjunctive structure* $(\mathbb{B},\subseteq,\otimes,(\cdot)^\perp)$.

Proof. $\mathcal{S}_\mathbb{B}$ is upwards closed by construction, and the proof that all the combinators are inhabited by fusions is given in Proposition 41. To prove that it is compatible with the modus ponens, let $a,b\in\mathbb{B}$ be behaviors and p,q be processes such that $p\in a\multimap b$ and $q\in a$. By Proposition 37, we get $(a\multimap b) * a \subseteq b$, and thus in particular $p*q\in b$, *i.e.* $b\in\mathcal{S}_\mathbb{B}$. ◀

5 Conjunctive algebras for concurrency

5.1 Parallel composition

So far, we identified the conjunctive part of the realizability model, that is, processes handling disjoint namespaces. We shall now investigate the necessary structure to algebrize parallelism.

▶ **Definition 43.** *Let us consider a CS* $(\mathbb{C},\otimes,(\cdot)^\perp,\preccurlyeq)$. *A* composition *on* $\mathbb{C}$ *is an increasing and* $\bigcurlyvee$*-continuous function* $F:\mathbb{C}\times\mathbb{C}\to\mathbb{C}$.

Given a composition F *on* $\mathbb{C}$, *define the induced* compositional structure *associated to* F *as the quotient* $\mathbb{C}/\equiv_F$ *where the equivalence relation* $\equiv_F$ *is the minimum equivalence relation that satisfies the rules of Figure 2.*

Observe that the first four rules express that $(\mathbb{C},F,\mathbb{1},\equiv_F)$ is an commutative monoid while the last ones express that the structure of the CS $\mathbb{C}$ is lifted to the quotient $\mathbb{C}/\equiv_F$, thus inducing respectively the operations $\otimes_F$, $(\cdot)^\perp_F$ and the preorder $\preccurlyeq_F$. It is a direct verification that the structure $(\mathbb{C}/{\equiv_F},\otimes_F,(\cdot)^\perp_F,\preccurlyeq_F)$ is also a CS. Provided there is no ambiguity, we will avoid the subscripts F on the induced operations. The structure $(\mathbb{C}/{\equiv_F},\otimes,(\cdot)^\perp,F,\preccurlyeq)$ induced by F on $\mathbb{C}$ is denoted as $\mathbb{C}_F$.

▶ **Definition 44.** *Let us consider* $(\mathbb{C},\otimes,(\cdot)^\perp,1,F,\preccurlyeq)$ *a CS with composition* F. *We define* $\triangleright_F:\mathbb{C}\times\mathbb{C}\to\mathbb{C}$ *by means of* $b\triangleright_F c \triangleq \bigcurlyvee\{x\in\mathbb{C} \mid xFb \preccurlyeq c\}$.

It is straightforward to check that $\rhd_F$ defines a right adjoint to F.

▶ **Proposition 45.** *Let us consider* $(\mathbb{C}, \otimes, (\cdot)^\perp, 1, F, \preccurlyeq)$ *a CS with composition* F*. Then* $aFb \preccurlyeq c \iff a \preccurlyeq b \rhd_F c$ *for all* $a, b, c \in \mathbb{C}$*.*

Proof. For the direct implication, $aFb \preccurlyeq c$ implies that $a \in \{x \in \mathbb{C} \mid xFb \preccurlyeq c\}$ hence $a \preccurlyeq b \rhd_F c$. For the reverse implication, since F is monotone, we have

$$aFb \preccurlyeq (b \rhd_F c)Fb \;=\; \Big(\curlyvee \{x \in \mathbb{C} \mid xFa \preccurlyeq c\}\Big) \mid b \;\preccurlyeq\; \curlyvee \{xFb \mid x \in \mathbb{C}, xFb \preccurlyeq c\} \;\preccurlyeq\; c. \blacktriangleleft$$

▶ **Proposition 46.** *Consider the CS* $(\mathbb{B}, \otimes, (\cdot)^\perp, 1, \subseteq)$ *of behaviors over* $\bar{\Pi}$*.*

1. *The operation* $\parallel$ *(from Definition 35) is a composition over* $\mathbb{B}$*, the equivalence* $\equiv_\parallel$ *is equality and its right adjoint is* $B \rhd_\parallel C := (B \parallel C^\perp)^\perp$*.*
2. *The operation* $\otimes$ *(from Definition 35) is a composition over* $\mathbb{B}$ *and its right adjoint is* $B \rhd_\otimes C := \big(((B^2 \parallel C^\perp)^\perp)_{\restriction \mathbb{N}^1}\big)^{-1}$*.*

Proof.

1. Since $(\overline{\Pi}, |, \mathbb{1})$ is a commutative monoid, then $(\mathbb{B}, \|, \mathbf{1})$ is also a commutative monoid with unit $\mathbf{1} := \{\mathbb{1}\}^{\perp\perp}$. The function $\|$ is increasing by definition. For the continuity, we can check that $(\bigvee \mathfrak{B})\,|A \subseteq \bigvee \{B\|A \mid B \in \mathfrak{B}\}$ for all $\mathfrak{B} \subseteq \mathbb{B}$ and $A \in \mathbb{B}$, and we prove the continuity in the right argument similarly. Besides, $A\|B \subseteq C$ is equivalent to $C^\perp \subseteq (A\|B)^\perp$, which is equivalent to $\forall p \in A, q \in B, r \in C^\perp, \overline{\nu}(p \mid q \mid r) \in \perp\!\!\!\perp$. The last condition is equivalent to $A \subseteq (B\|C^\perp)^\perp$, which proves that $B \rhd_\| C := (B\|C^\perp)^\perp$ is the right adjoint of $\|$.
2. We have $(\bigvee \mathfrak{B}) \otimes A = \bigvee \{B \otimes A \mid B \in \mathfrak{B}\}$ and $A \otimes (\bigvee \mathfrak{B}) = \bigvee \{A \otimes B \mid B \in \mathfrak{B}\}$ from Proposition 40, and by construction $\otimes$ is increasing. Moreover, $A \otimes B \subseteq C$ is equivalent to $C^\perp \subseteq (A \otimes B)^\perp$, which is equivalent to $\forall p \in A, q \in B, r \in C^\perp, \overline{\nu}(p^1 \mid q^2 \mid r) \in \perp\!\!\!\perp$. The last condition is equivalent to $A^1 \subseteq ((B^2 \mid C^\perp)^\perp)_{\restriction \mathbb{N}^1}$ hence $A \subseteq \big(((B^2 \mid C^\perp)^\perp)_{\restriction \mathbb{N}^1}\big)^{-1}$, which proves that $B \rhd_\otimes C := \big(((B^2 \mid C^\perp)^\perp)_{\restriction \mathbb{N}^1}\big)^{-1}$ is the right adjoint of $\otimes$. ◀

5.2 Embedding the π-calculus through Honda-Yoshida combinators

Following previous works on algebraic structures for realizability models based on sequential calculi [15, 16], we face two options to show that the π-calculus can be faithfully embedded with conjunctive involutive structure with compositions. Either we find a way to soundly embed the different constructs $u(\vec{x}).P, \bar{u}\,\langle\vec{v}\rangle\,, |\;(\nu y)P$, or we can define a set of combinators that is complete for this calculus (as are S and K for the λ-calculus). We opt for the second approach, building on Honda & Yoshida combinators for the π-calculus [10].

▶ **Definition 47.** *A* Honda-Yoshida structure *(HYS) is a CIS* $(\mathbb{C}, \preccurlyeq, \otimes, |, (\cdot)^\perp, 1)$ *with composition* $|$ *together with a function* $\mathsf{M} : \mathbb{N} \times \mathbb{N} \to \mathbb{C}$*. Given such a structure, we define the* Honda-Yoshida combinators *of* $\mathbb{C}$ *by means of:*

$$\begin{array}{r|r}
\mathsf{K}(a) \triangleq \curlywedge_{x\in\mathbb{N}} \big(\mathsf{M}(a,x) \rhd 1\big) & \mathsf{Br}(a,b) \triangleq \curlywedge_{x\in\mathbb{N}} \big(\mathsf{M}(a,x) \rhd \mathsf{F}(b,x)\big) \\
\mathsf{F}(a,b) \triangleq \curlywedge_{x\in\mathbb{N}} \big(\mathsf{M}(a,x) \rhd \mathsf{M}(b,x)\big) & \mathsf{D}(a,b,c) \triangleq \curlywedge_{x\in\mathbb{N}} \big(\mathsf{M}(a,x) \rhd \mathsf{M}(b,x)|\mathsf{M}(c,x)\big) \\
\mathsf{Bl}(a,b) \triangleq \curlywedge_{x\in\mathbb{N}} \big(\mathsf{M}(a,x) \rhd \mathsf{F}(x,b)\big) & \mathsf{S}(a,b,c) \triangleq \curlywedge_{x\in\mathbb{N}} \big(\mathsf{M}(a,x) \rhd \mathsf{F}(b,c)\big)
\end{array}$$

These definitions are sound with respect to the expected reductions of these combinators, in the sense that each reduction rule $p \to q$ (in [10]) directly yields an inequality $p \preccurlyeq q$.

▶ **Proposition 48.** *Given a HYS* $(\mathbb{C}, \preccurlyeq, \otimes, |, (\cdot)^\perp, 1, \mathsf{M})$ *we have:*

$$\begin{array}{rcl|rcl} \mathsf{K}(a)|\mathsf{M}(a,x) & \preccurlyeq & 1 & \mathsf{Bl}(a,b)|\mathsf{M}(a,x) & \preccurlyeq & \mathsf{F}(x,b) \\ \mathsf{F}(a,b)|\mathsf{M}(a,x) & \preccurlyeq & \mathsf{M}(b,x) & \mathsf{Br}(a,b)|\mathsf{M}(a,x) & \preccurlyeq & \mathsf{F}(b,x) \\ \mathsf{D}(a,b,c)|\mathsf{M}(a,x) & \preccurlyeq & \mathsf{M}(b,x)|\mathsf{M}(c,x) & \mathsf{S}(a,b,c)|\mathsf{M}(a,x) & \preccurlyeq & \mathsf{F}(b,c) \end{array}$$

Proof. Straightforward from the definitions. ◀

▶ **Definition 49.** *A* Honda-Yoshida algebra *(HYA) is a HYS* $(\mathbb{C}, \preccurlyeq, \otimes, |, (\cdot)^\perp, 1, \mathsf{M})$ *together with a monoidal separator* $\mathcal{S} \subseteq \mathbb{C}$ *s.t. all Honda-Yoshida combinators belong to* $\mathcal{S}$*.*

So far we have not needed to introduce a notion of reduction because the constructors of Linear Logic in our types reflect the properties of connections and handling of names. However, to obtain a model of behaviors $\mathcal{B}$ where the Honda-Yoshida combinators are inhabited, we ask the poles to be closed by anti-reduction. This choice follows the first author's design when defining the *regular poles* in [2].

▶ **Definition 50.** *Let us consider a pole* $\perp\!\!\!\perp \subseteq \bar{\Pi}$*. We say that* $\perp\!\!\!\perp$ *is* regular *iff for all* $(P, e), (Q, f)$ *(PGF)-processes and names* $u, \vec{x}$*,* $\{(P \mid Q, ef)\}^\perp \subseteq \{(\bar{u}\,\langle\vec{x}\rangle\,.P \mid v(\vec{x}).Q, ef)\}^\perp$ *whenever* $u \underset{ef}{\sim} v$*.*

Following [11], the combinators of Honda & Yoshida can be encoded into PGF using the usual π-calculus (pure) processes:

$$\begin{array}{rcl|rcl} \mathsf{m}(a,x) & \triangleq & \bar{a}\,\langle x\rangle & & & \\ \mathsf{k}(a) & \triangleq & a(x) & \mathsf{br}(a,b) & \triangleq & a(x).\mathsf{f}(b,x) \\ \mathsf{f}(a,b) & \triangleq & a(x).\bar{b}\,\langle x\rangle & \mathsf{d}(a,b,c) & \triangleq & a(x).(\mathsf{m}(b,x) \mid \mathsf{m}(c,x)) \\ \mathsf{bl}(a,b) & \triangleq & a(x).\mathsf{f}(x,b) & \mathsf{s}(a,b,c) & \triangleq & a(x).\mathsf{f}(b,c) \end{array}$$

▶ **Proposition 51.** *A CIMA of behaviours* $(\mathbb{B}, \preccurlyeq, \otimes, \|, (\cdot)^\perp, 1, \mathcal{S}_\mathbb{B})$ *induced by a regular pole* $\perp\!\!\!\perp$ *is a HYA with the definition* $\mathsf{M}(a,b) \triangleq \{(\bar{a}\,\langle b\rangle\,.\mathbb{1}, \Delta_\mathbb{N})\}^{\perp\perp}$*.*

Proof. It suffices to prove that all the combinators of Definition 47 are inhabited. For that we consider the standard encoding of the negative combinators into the π-calculus (which is embedded into PGF). Then, using the regularity of the pole, we prove that the encoding belongs to the corresponding combinator defined in $\mathbb{C}$. Since $\mathsf{M}(a,b)$ is not empty, we have $\mathsf{M}(a,b) \in \mathcal{S}_\mathbb{B}$ by definition of $\mathcal{S}_\mathbb{B}$. The other combinators $\mathsf{K}(a), \mathsf{F}(a,b), \mathsf{Bl}(a,b), \mathsf{Br}(a,b), \mathsf{D}(a,b,c)$ and $\mathsf{S}(a,b,c)$ are given by Definition 49 and they are inhabited by their respective encoding on (PGF).

Let us illustrate the technique by proving the case of $\mathsf{K}a$. Observe that since the pole is regular, we have $\{\mathbb{1}\}^\perp \subseteq \{\mathsf{k}(a)|\mathsf{m}(a,x)\}^\perp = \Big(\{\mathsf{k}(a)\}^{\perp\perp}|\mathsf{M}(a,x)\Big)^\perp = \Big(\{\mathsf{k}(a)\}^{\perp\perp} \parallel \mathsf{M}(a,x)\Big)^\perp$. Applying the orthogonal map we get: $\{\mathsf{k}(a)\}^{\perp\perp} \parallel \mathsf{M}(a,x) \subseteq \{\mathbb{1}\}^{\perp\perp} \overset{\text{def}}{=} 1$. By Proposition 45 we deduce that $\{\mathsf{k}(a)\}^{\perp\perp} \subseteq \mathsf{M}(a,x) \rhd 1$ for all names a, b, x. Taking the meet over x, we obtain $\mathsf{k}(a) \in \{\mathsf{k}(a)\}^{\perp\perp} \subseteq \bigwedge_{x\in\mathbb{N}} (\mathsf{M}(a,x) \rhd 1) = \mathsf{K}(a)$ and thus $\mathsf{k}(a) \in \mathsf{K}(a)$. ◀

6 Parallel composition cannot be derived

We shall now see that the extra structure presented in the previous section to encompass parallelism within conjunctive involutive monoidal algebra is actually necessary, in the sense that parallelism cannot be derived from the ground structure of a conjunctive involutive

$$\frac{(x : P^{\perp}) \in \Gamma}{\vdash x : P \mid \Gamma}\,(\textsc{Ax}_{+}) \qquad \frac{\vdash t : A \mid \Gamma \quad \vdash u : B \mid \Gamma}{\vdash (t, u) : A \otimes B \mid \Gamma}\,(\otimes) \qquad \frac{\vdash t : A[P/X] \mid \Gamma}{\vdash t : \exists X.A \mid \Gamma}\,(\exists)$$

$$\frac{(\alpha : N^{\perp}) \in \Gamma}{\vdash \alpha : N \mid \Gamma}\,(\textsc{Ax}_{-}) \qquad \frac{c : (\vdash \kappa : A, \kappa' : B, \Gamma)}{\vdash \mu(\kappa, \kappa').c : A \parr B \mid \Gamma}\,(\parr) \qquad \frac{\vdash V : A \mid \Gamma \quad X \notin FV(\Gamma)}{\vdash V : \forall X.A \mid \Gamma}\,(\forall)$$

$$\frac{c : (\vdash \kappa : A, \Gamma)}{\vdash \mu\kappa.c : A \mid \Gamma}\,(\mu) \qquad \frac{\vdash t : A \mid \Gamma \quad \vdash u : A^{\perp} \mid \Gamma}{\langle t \parallel u \rangle : (\vdash \Gamma)}\,(\textsc{Cut})$$

Figure 3 System L, typing rules.

monoidal algebra. Indeed, we observed that in the CIMA induced by PGF behaviors, the parallel composition is preexistent since there exists not only an external[3] continuous function, but even a term Φ s.t. $\Phi * t * u = t \| u$ for all t, u. This explains why, in this case, there is no need for a quotient (see Proposition 46.1).

In the general case, when there is no such preexisting term, it is always possible to follow Definition 43 to recover the expected structure through a quotient. We will now highlight that this is, in general, necessary, in the sense that parallel composition can not always be defined with only terms and application in an arbitrary CIMA. We do so by providing a concrete example of a conjunctive involutive monoidal algebra in which we prove that there is no term satisfying the axioms of parallel composition.

We build on a realizability interpretation based on a fragment of Munch-Maccagnoni's system L [17], a polarised sequent calculus tailored to give a term language to Girard's classical logic LC. To draw the comparison with other sequent calculi such as Curien-Herbelin's $\bar{\lambda}\mu\tilde{\mu}$-calculus [4], the main difference lies in the use of polarisation, which makes lazy and strict qualify logical connectives rather than evaluation strategies: instead of having to enforce globally a strategy to avoid critical pairs, lazy and strict evaluations coexist and are dictated locally by the polarities of the corresponding connectives. For a more detailed introduction on the rationale of L, we refer the interested reader to [18].

We work here with a fragment of L, keeping only the connectives necessary to induce a CIMA. As explained above, formulas are divided into positive and negative ones:

$$\begin{array}{ll|l} \textbf{Positive formulas} & P ::= X \ \mid A \otimes B \mid \exists X.A & \\ \textbf{Negative formulas} & N ::= X^{\perp} \mid A \parr B \mid \forall X.A & \textbf{Formulas} \quad A, B ::= P \mid N \end{array}$$

To each positive formula P is associated a dual negative formula $P^{\perp}$ (and vice-versa), defined by induction on the syntax of formulas. As is usual in linear logic, the map $(\cdot)^{\perp}$ defines an involutive function on formulas:

$$\begin{array}{r|c|l} (X)^{\perp} = X^{\perp} & (A \otimes B)^{\perp} = A^{\perp} \parr B^{\perp} & (\forall X.A)^{\perp} = \exists X.A^{\perp} \\ (X^{\perp})^{\perp} = X & (A \parr B)^{\perp} = A^{\perp} \otimes B^{\perp} & (\exists X.A)^{\perp} = \forall X.A^{\perp} \end{array}$$

Similarly, the syntax of terms and values reflect these distinction between negative and positive connectives. To ease the connection with [17], we stick to the same presentation:

$$\begin{array}{lrl|ll} \textbf{Variables} & \kappa & ::= x \mid \alpha & \textbf{Commands} & c ::= \langle t_{+} \parallel t_{-} \rangle \mid \langle t_{-} \parallel t_{+} \rangle \\ \textbf{Terms} & t_{-} & ::= \alpha \mid \mu x.c \mid \mu(\kappa, \kappa).c & & \\ & t_{+} & ::= x \mid \mu\alpha.c \mid (t, u) & \textbf{Values} & V_{+} ::= x \mid (V, V') \\ & t & ::= t_{+} \mid t_{-} & & V ::= V_{+} \mid t_{-} \end{array}$$

[3] That is a meta-theoretic function, as opposed to an *internal* term definable in the language of the algebraic structure at play.

In the sequel, we write $\mathcal{T}_+$ (resp. $\mathcal{T}_-$, $\mathcal{C}$, $\mathbb{V}$) for the set of positive terms (resp. negative terms, commands, values), and denote by $\mathcal{T}_+^0$, etc... the corresponding set of closed terms. As in [17], we use monolateral sequents, following Girard's tradition to have all the formulas on the right. To that end, we need to quotient the syntax with a new α-equivalence $\langle t \,\|\, u\rangle \equiv \langle u \,\|\, t\rangle$, which will simplify overall the presentation. The type system, which distinguishes between sequents of the shape $\vdash t : A \mid \Gamma$ to type terms (where A is the formula *in the stoup*) and sequents of the shape $c : (\vdash \Gamma)$ to type commands, is given in Figure 3. The reader may observe that the rule for the universal quantifier has to be restricted to values in order to avoid the usual inconsistency of polymorphism in presence of side-effects [17].

The operational semantics is defined as a weak-head reduction relation on commands, by means of the following reduction rules:

$$\begin{array}{lcl|rcl}
\langle \mu\alpha.c \,\|\, V\rangle & \to_\mu & c[V/\alpha] & \langle (V,V') \,\|\, \mu(\kappa,\kappa').c\rangle & \to_\beta & c[V/\kappa, V'/\kappa'] \\
\langle V \,\|\, \mu x.c\rangle & \to_\mu & c[V/x] & \langle (t,t') \,\|\, u\rangle & \to_\xi & \Big\langle t \,\Big\|\, \mu\kappa.\langle t' \,\|\, \mu\kappa'.\langle(\kappa,\kappa') \,\|\, u\rangle\rangle\Big\rangle
\end{array}$$

The different binders only reduce in front of values, while the $\to_\xi$ rules specify how constructors should be expanded when they are built on top of a term instead of a value. In particular, it is worth noting that the reduction $\to \triangleq \to_\mu \cup \to_\beta \cup \to_\xi$ has no critical pair, which entails the Church-Rosser property when considering its extension to subcommands.

Following [17], we can now define a realizability interpretation *à la* Krivine relying on this calculus. As is usual in Krivine realizability, we say that a *pole* is any subset of $\mathcal{C}^0$ that is closed by anti-reduction. This induces an orthogonality relation on terms, which we extend to any sets $T \in \mathcal{P}(\mathcal{T}_+^0)$ or $U \in \mathcal{P}(T_-^0)$ by defining:

$$T^\perp \triangleq \{t_- \in \mathcal{T}_-^0 : \forall t_+ \in T, \langle t_+ \,\|\, t_-\rangle \in \perp\!\!\!\perp\} \quad \Big| \quad U^\perp \triangleq \{t_+ \in \mathcal{T}_+^0 : \forall t_- \in U, \langle t_+ \,\|\, t_-\rangle \in \perp\!\!\!\perp\}$$

In this context, we call *behavior* any subset T of $\mathcal{T}_+^0$ or $\mathcal{T}_-^0$ such that $T = T^{\perp\perp}$ and we denote by $\mathbf{H}$ the set of all behaviors. Observe that $(\cdot)^\perp$ defines an involutive map on $\mathbf{H}$.

We are now ready to define a realizability interpretation that will associate to any formula a behavior. More precisely, we extend the language of formulas to consider formulas with (positive) parameters R in the set $\Pi \triangleq \mathcal{P}(\mathcal{T}_+^0 \cap \mathbb{V})$, and to any such positive formula P we associate a behavior $|P| \in \mathcal{P}(\mathcal{T}_+^0)$ while to any negative formula N we associate a behavior $|N| \in \mathcal{P}(\mathcal{T}_-^0)$. The interpretation $|\cdot|$ is defined by induction on formulas:

$$\begin{array}{rcl|rcl}
|R| & \triangleq & R^{\perp\perp} & |R^\perp| & \triangleq & R^\perp \\
|A \otimes B| & \triangleq & (|A| \otimes |B|)^{\perp\perp} & |A \parr B| & \triangleq & (|A^\perp| \otimes |B^\perp|)^\perp \\
|\exists X.A| & \triangleq & \left(\bigcup_{R\in\mathbb{V}} |A[R/X]|_{\mathbb{V}}\right)^{\perp\perp} & |\forall X.A| & \triangleq & \left(\bigcap_{R\in\mathbb{V}} |A[R/X]|_{\mathbb{V}}\right)^{\perp\perp}
\end{array}$$

where $|A|_{\mathbb{V}} \triangleq |A| \cap \mathbb{V}$ and $T \otimes U \triangleq \{(t,u) : t \in T \wedge u \in U\}$. Observe that by construction, for any closed formula A, it holds that $|A^\perp| = |A|^\perp$.

▶ **Definition 52.** *For any term t, one says that t realizes a formula A whenever $t \in |A|$, which we denote by $t \Vdash A$. Similarly, we write $\sigma \Vdash \Gamma$ to denote that a substitution σ realizes a context $\Gamma \equiv x_1 : A_1, ..., x_n : A_n$ when for any $1 \leq i \leq n$, $\sigma(x_i) \Vdash A_i$.*

We say that t is a universal realizer *of A and we write $t \Vvdash A$ when a term t realizes A for any possible choice of pole.*

We will not develop any further the properties of this construction, but we shall at least emphasize that the typing rules defined in Figure 3 are adequate with respect to this interpretation, which is proven as usual by induction over typing derivations [17].

▶ **Proposition 53** (Adequacy). *Let Γ be a typing context, $\perp\!\!\!\perp$ a pole and σ be a substitution such that $\sigma \Vdash \Gamma$, then for any term t and formula A, if $\vdash t : A \mid \Gamma$, then $t^\sigma \Vdash A$.*

Even though the fragment of L we have selected does not account for linearity, the last proposition shows that the realizability interpretation still defines a model for MLL. In particular, as was observed in [16], algebraically it induces a conjunctive structure. This observation can be adapted to our framework, to show that the realizability interpretation defined above actually induces a conjunctive involutive monoidal algebra.

▶ **Proposition 54.** *The tuple $(\boldsymbol{H}, \subseteq, \otimes, (\cdot)^\perp, \mathbb{1}, \mathcal{S})$, where $\mathbb{1} \triangleq \mathcal{T}_+^0$ and $\mathcal{S} = \boldsymbol{H} \backslash \emptyset$, is a CIMA.*

Proof. The proof is analogous to the proof in [16], up to the observation that $(\cdot)^\perp$ is involutive on $\mathbf{H}$ (by definition of $\mathbf{H}$). We first show that $(\mathbf{H}, \subseteq, \otimes, (\cdot)^\perp)$ is a conjunctive involutive structure. Indeed, the set of behaviors, ordered by inclusion, defines a complete lattice whose join is given by the union, and the map $\cdot \otimes \cdot$ is clearly covariant in its two arguments. while it is clear that $(\cdot)^\perp$ is anti-monotonic. Regarding De Morgan law, we have for any set $\mathcal{U} \subseteq \mathcal{T}^0$:

$$\Big(\bigcup_{T \in \mathcal{U}} T\Big)^\perp = \Big(\bigcup_{T_+ \in \mathcal{U} \cap \mathcal{T}_+} T_+ \;\cup \bigcup_{T_- \in \mathcal{U} \cap \mathcal{T}_-} T_-\Big)^\perp = \Big(\bigcup_{T_+ \in \mathcal{U} \cap \mathcal{T}_+} T_+\Big)^\perp \cap \Big(\bigcup_{T_- \in \mathcal{U} \cap \mathcal{T}_-} T_-\Big)^\perp$$
$$= \bigcap_{T_+ \in \mathcal{U} \cap \mathcal{T}_+} T_+^\perp \;\cap \bigcap_{T_- \in \mathcal{U} \cap \mathcal{T}_-} T_-^\perp \;= \bigcap_{T \in \mathcal{U}} T^\perp$$

Then we prove that, when considering the unit $\mathbb{1} \triangleq \mathcal{T}_+^0$ (which is indeed a behavior), the set of non-empty behaviors $\mathcal{S} = \mathbf{H} \backslash \emptyset$ defines a valid separator. This mostly amounts to finding a realizer for each of the different axioms, since $\mathcal{S}$ is clearly upwards-closed. and is compatible with modus ponens. Indeed, if $A, B \in \mathbf{H}$ are such that $A \multimap B \in \mathcal{S}$ and $A \in \mathcal{S}$, by definition of $\mathcal{S}$ it means that there exists two terms t and u such that $t \in A \multimap B$ and $u \in A$. Then $t\,u = \mu\kappa.\langle t \parallel (u, \alpha)\rangle \Vdash B$, using the typing rule for application above and Proposition 53. To find realizers for the axiom, it suffices to find a term of the expected type, and to use adequacy (cf. Proposition 53). For instance, we can pick:

- $\mu(x, \alpha).\langle x \parallel \mu(\kappa_1, \kappa_2).\langle(\kappa_2, \kappa_1) \parallel \alpha\rangle\rangle \Vdash (A \otimes B) \multimap (B \otimes A)$
- $\lambda z.\mu(x, \beta).\langle x \parallel \mu(\kappa, \kappa').\langle(z\,\kappa, \kappa') \parallel \beta\rangle\rangle \Vdash (A \multimap B) \multimap (A \otimes C) \multimap (B \otimes C)$
- $\mu(x, \alpha).\Big\langle x \,\Big\|\, \mu(y, \kappa_3).\langle y \parallel \mu(\kappa_1, \kappa_2).\langle(\kappa_1, (\kappa_2, \kappa_3)) \parallel \alpha\rangle\rangle\Big\rangle \Vdash ((A \otimes B) \otimes C) \multimap (A \otimes (B \otimes C))$
- For any $t_+ \in \mathcal{T}_+^0$, $\lambda\kappa.(t_+, \kappa) \Vdash A \multimap \mathbb{1} \otimes A$
- $\mu(x, \kappa).\langle x \parallel \mu(\_, \kappa').\langle \kappa' \parallel \kappa\rangle\rangle \Vdash \mathbb{1} \otimes A \multimap A$
- $\mu(x, \alpha).\langle x \parallel \mu(\kappa, \kappa').\langle(\kappa, \kappa') \parallel \alpha\rangle\rangle \Vdash (A \multimap B) \multimap (B^\perp \multimap A^\perp)$ ◀

We have shown so far that the realizability interpretation based on L defines a conjunctive involutive monoidal algebra as expected. Besides, we can show that this particular kind of CIMA does not admit parallelism, thus justifying the necessity of the extra requirements formulated in Definition 43. Indeed, to obtain a compositional structure in such a model without adding some extra structure would mean the desired composition F could be defined by means of a term t (*i.e.* $F(u, v) \triangleq t\,u\,b$) such that the expected axioms are satisfied.

We prove that this is not possible, in that the first two axioms ($F(a, \mathbb{1}) \equiv a$ and $F(\mathbb{1}, a) \equiv a$) cannot be satisfied together. The proof, which is folklore in Krivine realizability, mostly consists in proving that a term that realizes both of these axioms should essentially behave like a parallel ⫛ operator, which does not exist in L. This should not come as a surprise, since, to put it differently, it essentially means that to get the algebraic structure corresponding to parallelism, we actually need to have parallel computations in our calculus.

▶ **Proposition 55.** *There is no $t \in \mathcal{T}_+^0$ s.t. $t \Vvdash \forall X.(X \multimap \mathbb{1} \multimap X)$ and $t \Vvdash \forall X.(\mathbb{1} \multimap X \multimap X)$.*

Proof. To simplify the proof, let us assume that the calculus is extended with two inert (positive) constants $\boldsymbol{\kappa_1}, \boldsymbol{\kappa_2}$ and a negative one $\boldsymbol{\alpha}$ (alternatively, one could consider any terms $u_1, u_2 \in \mathcal{T}_+^0$ and $v \in \mathcal{T}_-^0$ such that the commands $\langle u_1 \parallel v\rangle$ and $\langle u_2 \parallel v\rangle$ are blocked).

We reason by contradiction by assuming that such a term t exists. Let $u \in \mathcal{T}_-^0$ be any term, and $\mathbb{X} \triangleq \{u\} \in \Pi$. Consider the pole $\perp\!\!\!\perp_1 = \{c : c \to \langle \kappa_1 \parallel u\rangle\}$. By construction, we have that $\boldsymbol{\kappa_1} \in |\mathbb{X}|$, $\boldsymbol{\kappa_2} \in |\mathbb{1}|$ and $\boldsymbol{\alpha} \in |\mathbb{X}|^{\perp}$. Since by assumption $t \Vdash \mathbb{X} \multimap \mathbb{1} \multimap \mathbb{X}$, we get $\langle t \parallel (\boldsymbol{\kappa_1}, (\boldsymbol{\kappa_2}, \boldsymbol{\alpha}))\rangle \in \perp\!\!\!\perp_1$, *i.e.* $\langle t \parallel (\boldsymbol{\kappa_1}, (\boldsymbol{\kappa_2}, \boldsymbol{\alpha}))\rangle \to \langle \boldsymbol{\kappa_1} \parallel \boldsymbol{\alpha}\rangle$.

By taking $\perp\!\!\!\perp_2 = \{c : c \to \langle \boldsymbol{\kappa_2} \parallel \boldsymbol{\alpha}\rangle\}$, we prove similarly that $\langle t \parallel (\boldsymbol{\kappa_1}, (\boldsymbol{\kappa_2}, \boldsymbol{\alpha}))\rangle \to \langle \boldsymbol{\kappa_2} \parallel \boldsymbol{\alpha}\rangle$. Since both commands $\langle \boldsymbol{\kappa_1} \parallel \boldsymbol{\alpha}\rangle$ and $\langle \boldsymbol{\kappa_2} \parallel \boldsymbol{\alpha}\rangle$ are blocked and the calculus is deterministic, we indeed found a contradiction. ◀

7 Conclusion and future work

7.1 Exponentials

In addition to studying in more depth the properties of the various structures introduced in this paper, a natural prolongation of this work would be to investigate the possibility to build on conjunctive involutive monoidal algebra to account for more expressive logical systems, in particular MALL with exponentials. This would provide us with a complete algebraic presentation of the first author's work [2]. To that end, we could rely on Honda & Yoshida combinatorial approach to replication [11]. Another path to follow in that direction would consist in exploring the connections with the Geometry of Interaction as it is presented in Duchesne's work [5], where MLL is interpreted using partial permutations and the tensor corresponds to an appropriate disjoint union. This bears similarities with our interpretation using injections $\iota_i : \mathbb{N} \to \mathbb{N}$ for the tensor to ensure disjoint namespaces.

7.2 Concurrency

A central element in our study of parallelism and its axiomatization is the role of non-determinism. Indeed, the presence in a process of several actions that may synchronize together in different ways is both a source of expressiveness, as witnessed by the famous "parallel or" which relies on this feature, but also an obstacle for logical study. In some sense, our search for an axiomatization of parallel composition aims at identifying the part of the logical structure that is responsible for such phenomena. An analogy can be made with the computational interpretation of classical logic: it is well known that the classical sequent calculus LK is not confluent, to the point that any direct denotational semantics of proofs is degenerated, but interesting semantics can be obtained by means of translations into better behaved logics. Such translations are effective because they impose constraints on proof reduction, like call-by-name or call-by-value policies in languages with control. Our system with parallelism could be envisaged as a linear analogue of full classical logic with no a priori constraints on strategies.

$$\begin{tikzcd}
\text{process calculus} \arrow[r, dotted, leftrightarrow] \arrow[d, "\text{determinization}"'] & \text{axiomatization of concurrency?} \arrow[d, dotted] & \text{LK} \arrow[l, dotted] \arrow[dl, "\text{various}"'] \arrow[d, "\neg\neg\text{ and CPS translations}"] \\
\text{deterministic processes} \arrow[r, leftrightarrow, "\text{typing}"'] & \text{linear logic} & \text{LJ} \arrow[l, "\text{linearization}"]
\end{tikzcd}$$

In this line of thought, the conjunctive structures we are after will provide the fundamental structure for logics of concurrent behaviours, which we can call an axiomatization of concurrency. Particular logics or type systems would be built on top of this structure, through the choice of appropriate operators, and possibly extra axioms, so that soundness of a particular type system for a particular calculus reduces to the fact that the considered calculus does provide an appropriate conjunctive structure.

References

1 Samson Abramsky, Esfandiar Haghverdi, and Philip Scott. Geometry of interaction and linear combinatory algebras. *Mathematical Structures in Computer Science*, 12(5):625–665, 2002. doi:10.1017/S0960129502003730.

2 Emmanuel Beffara. *Logique, réalisabilité et concurrence*. PhD Thesis, Université Paris 7, December 2005.

3 G. Birkhoff. *Lattice theory*, volume 25 of *Colloquium publications*. American Mathematical Society, 1940.

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 Étienne Duchesne. *La localisation en logique : géométrie de l'interaction et sémantique dénotationnelle*. Thèse de doctorat, Université Aix-Marseille 2, 2009. URL: http://www.theses.fr/2009AIX22080.

6 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.

7 Yuxi Fu. The χ-Calculus. In *1997 Advances in Parallel and Distributed Computing Conference (APDC)*, pages 74–81. Computer Society Press, 1997.

8 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987. doi:10.1016/0304-3975(87)90045-4.

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

10 Kohei Honda and Nobuko Yoshida. Combinatory representation of mobile processes. In *ACM-SIGACT Symposium on Principles of Programming Languages*, 1994.

11 Kohei Honda and Nobuko Yoshida. Replication in concurrent combinators. In Masami Hagiya and John C. Mitchell, editors, *Theoretical Aspects of Computer Software*, pages 786–805, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

12 J.M.E. Hyland. The effective topos. *Studies in Logic and the Foundations of Mathematics*, 110:165–216, 1982. The L. E. J. Brouwer Centenary Symposium. doi:10.1016/S0049-237X(09)70129-6.

13 J.-L. Krivine. Realizability in classical logic. In Interactive models of computation and program behaviour. *Panoramas et synthèses*, 27:197–229, 2009. Publisher: Société Mathématique de France.

14 Jean-Louis 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.

15 Alexandre Miquel. Implicative algebras: a new foundation for realizability and forcing. *Mathematical Structures in Computer Science*, 30(5):458–510, 2020. doi:10.1017/S0960129520000079.

16 Étienne Miquey. Revisiting the Duality of Computation: An Algebraic Analysis of Classical Realizability Models. In Maribel Fernández and Anca Muscholl, editors, *28th EACSL Annual Conference on Computer Science Logic (CSL 2020)*, volume 152 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:18, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CSL.2020.30.

17 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.

18 Guillaume Munch-Maccagnoni. *Syntax and Models of a non-Associative Composition of Programs and Proofs.* PhD thesis, Univ. Paris Diderot, 2013.

19 Joachim Parrow and Björn Victor. The Fusion Calculus: Expressiveness and Symmetry in Mobile Processes. In *13th IEEE Symposium on Logic in Computer Science (LICS)*, pages 176–185, 1998.

20 Andrew M. Pitts. Tripos theory in retrospect. *Mathematical Structures in Computer Science*, 12(3):265–279, 2002. doi:10.1017/S096012950200364X.

21 Davide Sangiorgi and David Walker. *The π-Calculus: A Theory of Mobile Processes.* Cambridge University Press, 2001.

22 Thomas Streicher. Krivine's classical realisability from a categorical perspective. *Mathematical Structures in Computer Science*, 23(6):1234–1256, 2013. doi:10.1017/S0960129512000989.

23 Lucian Wischik and Philippa Gardner. Explicit fusions. *Theoretical Computer Science*, 340(3):606–630, August 2005. doi:10.1016/j.tcs.2005.03.017.

A Quantitative Version of Simple Types

Daniele Pautasso
Dipartimento di Informatica, University of Torino, Italy

Simona Ronchi Della Rocca
Dipartimento di Informatica, University of Torino, Italy

Abstract

This work introduces a quantitative version of the simple type assignment system, starting from a suitable restriction of non-idempotent intersection types. The resulting system is decidable and has the same typability power as the simple type system; thus, assigning types to terms supplies the very same qualitative information given by simple types, but at the same time can provide some interesting quantitative information. It is well known that typability for simple types is equivalent to unification; we prove a similar result for the newly introduced system. More precisely, we show that typability is equivalent to a unification problem which is a non-trivial extension of the classical one: in addition to unification rules, our typing algorithm makes use of an expansion operation that increases the cardinality of multisets whenever needed.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases λ-calculus, intersection types, unification

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.29

Acknowledgements We would like to thank Delia Kesner and Antonio Bucciarelli for many useful discussions about the topic of this paper. We are also grateful to the anonymous referees for their careful reading and suggestions.

1 Introduction

Simple types. Simple type assignment for λ-calculus [8, 13] is a way of assigning types, which are formulas of minimal intuitionistic logic, to λ-terms. It enjoys important properties: to be typable implies strong normalization, and both typability and inhabitation are decidable – typability being the problem of determining, given a term, whether it is possible to assign a type to it, and inhabitation the problem of determining, given a type, if there is a term to which it can be assigned. Considering the λ-calculus as a general paradigm for functional programming languages, types are program specifications, so the decidability of typability provides tools for proving program correctness, that of inhabitation for program synthesis. Simple type assignment is the basis of typed functional languages, like ML and Haskell.

Quantitative intersection types. Recently the scientific community interest turned to a quantitative approach for program semantics, and from this point of view non-idempotent intersection types are a powerful tool. Intersection types have been introduced in [9], in order to increase the typability power of simple type assignment systems, but quite early they turned out to be useful in characterizing qualitative semantic properties of λ-calculus like solvability and strong normalization, and in describing models of λ-calculus in various settings [2, 20]. Intersection types are built by adding to the connective → of simple types an intersection connective $\wedge$ which enjoys associativity, commutativity and idempotency, i.e., $A \wedge A = A$; in other words, an intersection of types can be seen as a notation for a set of types. A variant of intersection types, where intersection is no more idempotent (so that intersection of types becomes a notation for multisets), was first designed by [10], and later used by De Carvalho [12], for the purpose of studying the complexity of reduction.

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 29; pp. 29:1–29:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Indeed non-idempotency has a quantitative flavour: non-idempotent intersection types have been used to design relational models of λ-calculus [17, 5], to characterize polynomial time computational complexity [4], and to compute the exact number of reductions to reach the normal form of a term [14]. The typability problem is undecidable both in idempotent and non-idempotent intersection type assignment systems; on the other hand, the inhabitation problem is decidable if we consider non-idempotent intersection [6], while it is undecidable for the idempotent case [22].

A question naturally arises: *is there a quantitative version of simple types?* Rephrasing it in a more technical way: *is there a restriction of the non-idempotent intersection type system with the same typability power as the simple type system?*

Contribution of this paper. In this paper we give a positive answer: we introduce a non-idempotent intersection type assignment system which is a restriction of the one defined in [7], and prove that the typability problem is decidable. The key idea is to restrict the multiset formation to uniform types, two types being uniform if they differ only in the cardinality of the multisets occurring in it. Assigning types to terms in such a system, which we name system $\mathcal{U}$, supplies the very same qualitative information given by simple types, but at the same time provides some interesting quantitative information.

In our analysis we take the well-known equivalence between typability in simple types and unification as a starting point, and prove a similar result for system $\mathcal{U}$. More precisely, we show that typability is equivalent to an extension of the classical unification problem: in addition to unification rules, our algorithm makes use of an operation, called expansion, which increases the cardinality of multisets whenever needed. Indeed, in the simple type system all the derivations for a given term share the same structure (as trees), and differ from each other only in the types occurring in their nodes. But, in the intersection type system, derivations for the same term can differ both in the previous sense and in the structure of the derivation. So unification is used to match the types in the nodes of a derivation, while the expansion modifies its structure.

Related works. Typability for intersection types has been intensively studied. A first approach, introducing the notion of expansion, can be found in [10]; the result was then extended to a more general case in [21, 19]. Both type systems considered in the aforementioned works enjoy an approximation theorem, where approximants are head normal forms in the λ-calculus extended with a constant Ω. This property allows the principal pair to be defined by induction on the structure of terms. We cannot use a similar technique here, since there is no syntactical characterization of simply typable terms. A different methodology has been explored in [16], by enriching types with pointers to the subtypes that can be modified by the expansion. In our work this role is played by a system of constraints, which also keeps track of the restrictions on the construction of multisets.

In the literature, some decidable restrictions of idempotent intersection types have been proposed, namely in [11] and [15]. In both cases, however, they are obtained by limiting the shape of types, and do not supply subsets of terms with interesting properties.

Paper organization. In Section 2 some well-known and essential facts about the simply typed λ-calculus are presented, together with the notion of principal derivation. In Section 3 we introduce the type assignment system $\mathcal{U}$ of uniform intersection types, extend the notion of principal derivation to it, and design a typing algorithm. Section 4 contains some clarifying examples and Section 5 the main result of the paper. A short conclusion is in Section 6. The most technical proof can be found in the Appendix.

$$\frac{}{\Gamma, x : A \vdash x : A}\ (\mathtt{var}) \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}\ (\to_{\mathtt{I}})$$

$$\frac{\Gamma \vdash M : A \to B \qquad \Delta \vdash N : A \qquad \Gamma \smile \Delta}{\Gamma \cup \Delta \vdash MN : B}\ (\to_{\mathtt{E}})$$

Figure 1 The system $\mathcal{S}$.

2 Simple types

We briefly recall the λ-calculus grammar and the simple type assignment system. Terms and term contexts of the λ-calculus are generated by the following grammars, respectively:

$$M, N, P ::= x \mid \lambda x.M \mid MN \qquad\qquad \mathtt{C} ::= \Box \mid \lambda x.\mathtt{C} \mid \mathtt{C}M \mid M\mathtt{C}$$

where x ranges over a countable set of variables. $\mathtt{FV}(M)$ denotes the set of free variables of M. We will use for λ-calculus the notations in [1].

▶ **Definition 1.** *The set $\mathcal{T}_{\mathcal{S}}$ of* simple types *is defined by the following grammar:*

$$A, B, C \quad ::= \quad a \mid A \to B$$

where a ranges over a countable set of type variables.

A *context* is a set of pairs $x : A$, where x is a term variable and $A \in \mathcal{T}_{\mathcal{S}}$; contexts are ranged over by Γ, Δ. If $x : A \in \Gamma$, then $\Gamma(x) = A$; the domain of a context Γ is $\mathtt{dom}(\Gamma) = \{x \mid x : A \in \Gamma\}$; the writing $\Gamma \smile \Delta$ means that Γ and Δ agree, i.e. if $x \in \mathtt{dom}(\Gamma) \cap \mathtt{dom}(\Delta)$ then $\Gamma(x) = \Delta(x)$; if S is a set of variables, then $\Gamma|_S$ denotes the restriction of $\mathtt{dom}(\Gamma)$ to S; moreover Γ, Δ is short for $\Gamma \cup \Delta$ in case $\mathtt{dom}(\Gamma) \cap \mathtt{dom}(\Delta) = \emptyset$.

The simple type assignment system $\mathcal{S}$ is a set of rules proving statements (typings) of shape $\Gamma \vdash M : A$, where Γ is a context, M a term and $A \in \mathcal{T}_{\mathcal{S}}$. The rules are shown in Figure 1. A *derivation* is a tree of rules, such that its leaves are applications of rule $(\mathtt{var})$ and the root is its conclusion. Derivations are ranged over by Π, Σ, Θ. We write $\Gamma \vdash M : A$ as a shorthand for the existence of a derivation proving $\Gamma \vdash M : A$, and when we want to name a particular derivation with such conclusion we write $\Pi \triangleright \Gamma \vdash M : A$. The system enjoys two important properties:

▶ **Theorem 2** (subject reduction). *$\Gamma \vdash M : A$ and $M \to_\beta N$ imply $\Gamma \vdash N : A$.*

▶ **Theorem 3** (strong normalization). *$\Gamma \vdash M : A$ implies M is strongly normalizing.*

The principal derivation

System $\mathcal{S}$ is decidable, and it enjoys the principal typing property, i.e., each typing for a term M is obtained from the principal one by substitution and weakening [3, 13].

Let $\mathtt{Var}(A)$ be the set of type variables occurring in the type A; we say that A and B are disjoint (notation $A * B$) if $\mathtt{Var}(A) \cap \mathtt{Var}(B) = \emptyset$. Both the definition of $\mathtt{Var}$ and the notion of disjointness can be extended to contexts and derivations in the standard way. A type A is fresh w.r.t. a derivation Π if $A * B$ for each B occurring in Π. Throughout the work we will frequently use indexed types, where indexes are natural numbers, and the symbols I, J will denote sets of indexes.

$$\frac{}{\Pi \rhd x : a \vdash_{\mathtt{p}} x : a}\ (\mathtt{var}) \qquad (E_\Pi, V_\Pi) = (\emptyset, \emptyset)$$

$$\frac{\Sigma \rhd \Gamma, x : a \vdash_{\mathtt{p}} M : A}{\Pi \rhd \Gamma \vdash_{\mathtt{p}} \lambda x.M : a \to A}\ (\to_{\mathtt{I}}) \qquad (E_\Pi, V_\Pi) = (E_\Sigma, V_\Sigma)$$

$$\frac{\Sigma \rhd \Gamma \vdash_{\mathtt{p}} M : A \quad x \notin \mathtt{dom}(\Gamma) \quad a \text{ fresh}}{\Pi \rhd \Gamma \vdash_{\mathtt{p}} \lambda x.M : a \to A}\ (\to_{\mathtt{I}})_{\emptyset} \qquad (E_\Pi, V_\Pi) = (E_\Sigma, V_\Sigma)$$

$$\frac{\Sigma_1 \rhd \Gamma \vdash_{\mathtt{p}} M : a \to B \quad \Sigma_2 \rhd \Delta \vdash_{\mathtt{p}} N : A \quad \Sigma_1 * \Sigma_2}{\Pi \rhd \Gamma \cup \Delta \vdash_{\mathtt{p}} MN : B}\ (\to_{\mathtt{E1}})$$

$$E_\Pi = E_{\Sigma_1} \cup E_{\Sigma_2} \cup \{a \doteq A\}$$
$$V_\Pi = V_{\Sigma_1} \cup V_{\Sigma_2} \cup \{\Gamma(x) \doteq \Delta(x) \mid x \in \mathtt{dom}(\Gamma) \cap \mathtt{dom}(\Delta)\}$$

$$\frac{\Sigma_1 \rhd \Gamma \vdash_{\mathtt{p}} M : a \quad \Sigma_2 \rhd \Delta \vdash_{\mathtt{p}} N : A \quad b \text{ fresh} \quad \Sigma_1 * \Sigma_2}{\Pi \rhd \Gamma \cup \Delta \vdash_{\mathtt{p}} MN : b}\ (\to_{\mathtt{E2}})$$

$$E_\Pi = E_{\Sigma_1} \cup E_{\Sigma_2} \cup \{a \doteq A \to b\}$$
$$V_\Pi = V_{\Sigma_1} \cup V_{\Sigma_2} \cup \{\Gamma(x) \doteq \Delta(x) \mid x \in \mathtt{dom}(\Gamma) \cap \mathtt{dom}(\Delta)\}$$

Figure 2 Principal derivation for $\mathcal{S}$.

A pseudo-derivation is a tree of rules assigning simple types to terms; we extend to pseudo-derivations all the notations for derivations. To each pseudo-derivation Π is associated a system of constraints (E_Π, V_Π), where E_Π and V_Π are sets of equations between types, written $A \doteq B$; we will drop subscripts from E_Π and V_Π when they are clear from the context.

▶ **Definition 4.** *The principal derivation for a term M, denoted by $\mathtt{PD}(M)$, is a pair $(\Pi, (E_\Pi, V_\Pi))$, where Π is a pseudo-derivation with subject M and (E_Π, V_Π) is the associated system of constraints, as defined in Fig. 2. $\mathtt{PD}(M)$ is unique, modulo renaming of type variables.*

▶ **Definition 5.** *Let $S = \{A_i \doteq B_i \mid i \in I\}$ be a set of equations between types.*

- *S is solvable if there is a substitution θ, replacing type variables by types, such that $\theta(A_i) = \theta(B_i)$ for all $i \in I$.*
- *S is in solved form iff:*
  - *every A_i is a variable a_i, and all a_i are distinct;*
  - *no left-hand side a_i appears in some right-hand side B_j for all $i, j \in I$.*
- *If $S = \{a_i \doteq A_i \mid i \in I\}$ is in solved form, $\vec{S}$ such that $\vec{S}(a_i) = A_i$ is the most general substitution solving it, also called most general unifier (m.g.u.).*
- *S is in unsolvable form iff it contains a circular equation, i.e. an equation $a \doteq A$ such that a occurs in A.*

The rules reducing an equational system either to solved or unsolvable form are given in Fig. 3; they are directly obtained from the classical Robinson's unification algorithm [18]. We will write $\mathtt{Unify}(S)$ for the procedure applying to S the rules of Fig. 3 as much as possible. A substitution θ can be extended to types and pseudo-derivations in the standard way.

$$\frac{S \cup \{A \doteq A\}}{S}\ \textit{erase} \qquad \frac{S \cup \{A \to B \doteq C \to D\}}{S \cup \{A \doteq C\} \cup \{B \doteq D\}}\ \textit{decompose}$$

$$\frac{S \cup \{A \to B \doteq a\}}{S \cup \{a \doteq A \to B\}}\ \textit{swap} \qquad \frac{S \cup \{a \doteq A\} \quad a \notin \mathtt{Var}(A) \quad a \in \mathtt{Var}(S)}{S[A/a] \cup \{a \doteq A\}}\ \textit{substitute}$$

Figure 3 Unification rules for simple types.

▶ Remark 6. A principal derivation $\mathtt{PD}(M) = (\Pi, (E, V))$ essentially becomes a derivation in $\mathcal{S}$ under a substitution θ solving the system of constraints; due to the differences between system $\mathcal{S}$ and the rules in Fig. 2, however, some minor adjustments are needed. To ease the notation we will leave such transformations implicit and just write $\theta(\Pi)$ for the derivation in which all $(\to_{\mathtt{I}})_\emptyset$ rules have been replaced by $(\to_{\mathtt{I}})$ rules (by suitably exploiting weakening in the axioms to introduce the abstracted dummy variables), and the distinction between rules $(\to_{\mathtt{E1}})$ and $(\to_{\mathtt{E2}})$ has been dropped, yielding rule $(\to_{\mathtt{E}})$.

The following result, first proved in [3], is well known:

▶ **Theorem 7.** *Let $\mathtt{PD}(M) = (\Pi, (E, V))$, where $\Pi \triangleright \Gamma \vdash_{\mathtt{p}} M : A$.*

1. *For every substitution θ which is a solution of $E \cup V$, $\theta(\Pi) \triangleright \theta(\Gamma) \vdash M : \theta(A)$.*
2. *For every derivation $\Sigma \triangleright \Delta \vdash M : B$ there is a substitution θ, solution of $E \cup V$, such that $\theta(\Gamma) = \Delta|_{\mathtt{FV}(M)}$ and $\theta(A) = B$.*

▶ Remark 8. In the literature the property speaks about principal pair, not principal derivation. The result is the same; here the presentation through the derivation is useful for the comparison with intersection types.

3 Uniform intersection types

We now define the uniform intersection type assignment system, based on the notion of uniform multiset. A multiset is an unordered list of elements; $\uplus$ denotes the union of multisets taking into account the multiplicities, and $|\sigma|$ denotes the cardinality of the multiset σ. The system is obtained from the system $\mathcal{H}_{e,w}$ in [6] by requiring that multisets contain only equivalent types.

▶ **Definition 9.**

- *Intersection types $(\mathcal{I})$ are defined by the following grammar:*

$$\begin{array}{lll} (\textbf{\textit{Intersection Types}}) & A, B, C & ::= \quad a \mid \sigma \to A \\ (\textbf{\textit{Multisets}}) & \sigma, \tau & ::= \quad [A_1, \dots, A_n] \quad (n \geq 1) \end{array}$$

  where a ranges over a countable set of type variables.

- *Equivalence relation on intersection types:*

$$\begin{array}{l} a \sim a \ \ \textit{for all variables } a; \\ \sigma \to A \sim \tau \to B \ \ \textit{iff} \ \ \sigma \sim \tau \textit{ and } A \sim B; \\ [A_1, ..., A_m] \sim [B_1, ..., B_n] \ \ \textit{iff} \ \ \forall i, j. A_i \sim B_j \quad (1 \leq i \leq m, 1 \leq j \leq n) \end{array}$$

- *Uniform intersection types $(\mathcal{T}_{\mathcal{U}})$ are defined by the following grammar:*

$$\begin{array}{lll} (\textbf{\textit{Unif. Int. Types}}) & A, B, C & ::= \quad a \mid \sigma \to A \\ (\textbf{\textit{Uniform Multisets}}) & \sigma, \tau & ::= \quad [A_1, ..., A_n] \quad \forall i, j. A_i \sim A_j \quad (1 \leq i, j \leq n) \end{array}$$

$$\frac{A \in \sigma}{\Gamma, x : \sigma \vdash_{\mathtt{i}} x : A}\ (\mathtt{var}) \qquad \frac{\Gamma, x : \sigma \vdash_{\mathtt{i}} M : A}{\Gamma \vdash_{\mathtt{i}} \lambda x.M : \sigma \to A}\ (\to_{\mathtt{I}})$$

$$\frac{\Gamma \vdash_{\mathtt{i}} M : [A_1, ..., A_n] \to B \quad (\Delta_i \vdash_{\mathtt{i}} N : A_i)_{1 \le i \le n} \quad \forall i.\Gamma \sim \Delta_i \quad \forall ij.\Delta_i \sim \Delta_j}{\Gamma \uplus_{1 \le i \le n} \Delta_i \vdash_{\mathtt{i}} MN : B}\ (\to_{\mathtt{E}})$$

Figure 4 The system $\mathcal{U}$.

For the sake of brevity, we well speak about types and multisets in both cases, when being uniform or not is clear from the context. Remark that in both grammars the empty multiset is not allowed. The types $A = [[a, a, a] \to b, [a] \to b] \to c$ and $B = [[a, a] \to b] \to c$ are uniform, whereas $[a, [a] \to b] \to c$ is not; moreover, observe that $A \sim B$.

The system $\mathcal{U}$, assigning types in $\mathcal{T}_{\mathcal{U}} \subset \mathcal{I}$ to terms, is shown in Fig. 4. We use the same notations as for simple types, but note that now a context assigns uniform multisets to term variables. If σ and τ are multisets, $\sigma \subseteq \tau$ means $\exists \rho.\tau = \sigma \uplus \rho$. If Γ and Δ are two contexts, $\Gamma \sim \Delta$ means that $\forall x \in \mathtt{dom}(\Gamma) \cap \mathtt{dom}(\Delta).\Gamma(x) \sim \Delta(x)$. The system is quantitative in the following sense:

▶ **Property 10.** *Let $\Pi \triangleright \Gamma, x : \sigma \vdash_{\mathtt{i}} M : A$.*

- *The number of free occurrences of x in M is bounded by $|\sigma|$.*
- *If n is the maximal cardinality of a multiset in Π, then every variable in M, either free or bound, has a number of occurrences $\le n$.*

Both the following properties are inherited from [7].

▶ **Theorem 11** (Subject reduction). $\Gamma \vdash_{\mathtt{i}} M : A$ *and* $M \to_\beta N$ *imply* $\Gamma \vdash_{\mathtt{i}} N : A$.

▶ **Theorem 12** (Strong normalization). $\Gamma \vdash_{\mathtt{i}} M : A$ *implies M is strongly normalizing.*

3.1 Principal derivations

We adapt the notion of principal derivation to system $\mathcal{U}$. In this setting, we will deal both with sets of equations and sets of equivalences between intersection types (multisets), written respectively as $A \doteq B$ ($\sigma \doteq \tau$), and $A \approx B$ ($\sigma \approx \tau$). A pseudo-derivation is a tree of rules assigning intersection types to terms. From now on multisets in a pseudo-derivation will be considered as lists, and consequently also the minor premises of the rules $(\to_{\mathtt{E1}})$ and $(\to_{\mathtt{E2}})$ will be ordered; this is essential to limit the complexity of the algorithm `Solve` we will discuss in the following section. To each pseudo-derivation Π we associate a system of constraints (E_Π, V_Π), where E_Π is a set of equations between intersection types and V_Π is a set of equivalence constraints between type variables.

▶ **Definition 13.**

- *A principal derivation for a term M is a pair $\mathtt{PD_i}(M) = (\Pi, (E_\Pi, V_\Pi))$, where Π is a pseudo-derivation with subject M and (E_Π, V_Π) is the associated system of constraints, as defined in Fig. 5.*
- *The minimal principal derivation for a term M is a principal derivation for M obtained by posing $n = 0$ whenever dealing with rule $(\to_{\mathtt{I}})$ of Fig. 5, and $n = 1$ whenever dealing with rules $(\to_{\mathtt{I}})_\emptyset$ and $(\to_{\mathtt{E2}})$. Since the minimal principal derivation is unique, modulo renaming of type variables, we will refer to it as $\mathtt{PD_i^{min}}(M)$.*

$$\frac{}{\Pi \rhd x : [a] \vdash_{\mathtt{q}} x : a}\ (\mathtt{var}) \qquad (E_\Pi, V_\Pi) = (\emptyset, \emptyset)$$

$$\frac{\Sigma \rhd \Gamma, x : \sigma \vdash_{\mathtt{q}} M : A \qquad a_1, ..., a_n \text{ fresh}}{\Pi \rhd \Gamma \vdash_{\mathtt{q}} \lambda x.M : \sigma \uplus [a_1, ..., a_n] \to A}\ (\to_{\mathtt{I}}) \qquad \begin{array}{l} E_\Pi = E_\Sigma \\ V_\Pi = V_\Sigma \cup \{a \approx b \mid a, b \in \sigma \uplus [a_1, ..., a_n]\} \end{array}$$

$$\frac{\Sigma \rhd \Gamma \vdash_{\mathtt{q}} M : A \quad x \notin \mathtt{dom}(\Gamma) \quad a_1, ..., a_n \text{ fresh}}{\Pi \rhd \Gamma \vdash_{\mathtt{q}} \lambda x.M : [a_1, ..., a_n] \to A}\ (\to_{\mathtt{I}})_\emptyset \qquad \begin{array}{l} E_\Pi = E_\Sigma \\ V_\Pi = V_\Sigma \cup \{a \approx b \mid a, b \in [a_1, ..., a_n]\} \end{array}$$

$$\frac{\Sigma \rhd \Gamma \vdash_{\mathtt{q}} M : [a_1, ..., a_n] \to B \quad (\Sigma_i \rhd \Delta_i \vdash_{\mathtt{q}} N : A_i)_{1 \le i \le n} \quad \forall i.\Sigma * \Sigma_i \quad \forall ij.\Sigma_i * \Sigma_j\ (i \neq j)}{\Pi \rhd \Gamma \uplus_{1 \le i \le n} \Delta_i \vdash_{\mathtt{q}} MN : B}\ (\to_{\mathtt{E1}})$$

$$\begin{array}{l} E_\Pi = E_\Sigma \cup_{1 \le i \le n} E_{\Sigma_i} \cup \{a_i \doteq A_i\}_{1 \le i \le n} \\ V_\Pi = V_\Sigma \cup_{1 \le i \le n} V_{\Sigma_i} \cup \{a \approx b \mid x \in \mathtt{dom}(\Gamma) \cap_{1 \le i \le n} \mathtt{dom}(\Delta_i), a \in \Gamma(x), b \in \Delta_i(x)\} \end{array}$$

$$\frac{\Sigma \rhd \Gamma \vdash_{\mathtt{q}} M : a \quad (\Sigma_i \rhd \Delta_i \vdash_{\mathtt{q}} N : A_i)_{1 \le i \le n} \quad b \text{ fresh} \quad \forall i.\Sigma * \Sigma_i \quad \forall ij.\Sigma_i * \Sigma_j\ (i \neq j)}{\Pi \rhd \Gamma \uplus_{1 \le i \le n} \Delta_i \vdash_{\mathtt{q}} MN : b}\ (\to_{\mathtt{E2}})$$

$$\begin{array}{l} E_\Pi = E_\Sigma \cup_{1 \le i \le n} E_{\Sigma_i} \cup \{a \doteq [A_1, ..., A_n] \to b\} \\ V_\Pi = V_\Sigma \cup_{1 \le i \le n} V_{\Sigma_i} \cup \{a \approx b \mid x \in \mathtt{dom}(\Gamma) \cap_{1 \le i \le n} \mathtt{dom}(\Delta_i), a \in \Gamma(x), b \in \Delta_i(x)\} \end{array}$$

Figure 5 Principal derivations for $\mathcal{U}$.

▶ **Definition 14.** *Let us denote by ψ a substitution $\mathtt{Var} \to \mathcal{I}$, and by ϕ a substitution $\mathtt{Var} \to \mathcal{T}_{\mathcal{U}}$. Substitutions are extended to multisets and pseudo-derivations in the standard way.*

- *ψ solves $E = \{A_i \doteq B_i \mid i \in I\} \cup \{\sigma_j \doteq \tau_j \mid j \in J\}$ if $\psi(A_i) = \psi(B_i)$ for all $i \in I$ and $\psi(\sigma_j) = \psi(\tau_j)$ for all $j \in J$;*
- *ϕ satisfies $V = \{A_i \approx B_i \mid i \in I\} \cup \{\sigma_j \approx \tau_j \mid j \in J\}$ if $\phi(A_i) \sim \phi(B_i)$ for all $i \in I$ and $\phi(\sigma_j) \sim \phi(\tau_j)$ for all $j \in J$;*
- *E is solvable w.r.t. V if there is a substitution ϕ solving E and satisfying V.*

In order to solve a system of constraints, we extend to intersection types the unification rules for simple types: Fig. 6 introduces two sets of rules for solving sets of equations or equivalences (replacing $\simeq$ either by $\doteq$ or by $\approx$, respectively). All the rules but *decomposeM* are the same for the two cases; observe that, in particular, the definition of *decomposeM*$_{\doteq}$ for multisets relies on the fact that multisets are now ordered. Let $\mathtt{QuasiUnify}_{\doteq}(S)$ (resp. $\mathtt{QuasiUnify}_{\approx}(S)$) denote the application of the rules in Fig. 6, replacing $\simeq$ by $\doteq$ (resp. by $\approx$), to the set S as much as possible. Moreover, if S is a set of equations, let $S/_{\approx}$ denote the set of equivalence constraints obtained replacing $\doteq$ by $\approx$. The writing $S/_{\doteq}$ is used for the dual transformation.

▶ **Definition 15.**

- *A set of constraints between intersection types $S = \{A_1 \simeq B_1, \dots, A_n \simeq B_n, \sigma_1 \simeq \tau_1, \dots, \sigma_m \simeq \tau_m\}$, where $\simeq \in \{\doteq, \approx\}$, is in solved form iff:*
  - *$m = 0$, i.e. there is no constraint involving multisets;*
  - *every A_i is a variable a_i, and all variables a_i are distinct;*
  - *no left-hand side a_i appears in some right-hand side B_j $(1 \le i, j \le n)$.*

$$\frac{S \cup \{A \simeq A\}\ (\{\sigma \simeq \sigma\})}{S}\ \mathit{erase} \qquad \frac{S \cup \{\sigma \to A \simeq \tau \to B\}}{S \cup \{\sigma \simeq \tau\} \cup \{A \simeq B\}}\ \mathit{decomposeT}$$

$$\frac{S \cup \{\sigma \to A \simeq a\}}{S \cup \{a \simeq \sigma \to A\}}\ \mathit{swap} \qquad \frac{S \cup \{a \simeq A\} \quad a \notin \mathtt{Var}(A) \quad a \in \mathtt{Var}(S)}{S[A/a] \cup \{a \simeq A\}}\ \mathit{substitute}$$

$$\frac{S \cup \{[A_1, \ldots, A_n] \doteq [B_1, \ldots, B_n]\}}{S \cup \{A_i \doteq B_i\}_{1 \leq i \leq n}}\ \mathit{decomposeM}_{\doteq}$$

$$\frac{S \cup \{\sigma \approx \tau\}}{S \cup \{A \approx B \mid A \in \sigma, B \in \tau\}}\ \mathit{decomposeM}_{\approx}$$

Figure 6 Unification rules for intersection types.

- *A constraint of the form $a \simeq A$ such that a occurs in A is a* circular constraint.
- *An equation between multisets $\sigma \doteq \tau$ such that $|\sigma| \neq |\tau|$ is a* critical equation.
- *A set S is in unsolvable form if it contains at least one circular constraint.*
- *A set of equations S is in blocked form if it contains at least one critical equation.*

▶ **Property 16.**

1. *Let S be a set of equations. Then $\mathtt{QuasiUnify}_{\doteq}(S)$ outputs a set of equations either in solved, unsolvable, or blocked form. Moreover, $\mathtt{QuasiUnify}_{\doteq}(S) = S'$ in solved form if and only if $\vec{S'}$ is the m.g.u. of S.*
2. *Let S be a set of equivalences. Then $\mathtt{QuasiUnify}_{\approx}(S)$ outputs a set of equivalences either in solved or unsolvable form. Moreover, $\mathtt{QuasiUnify}_{\approx}(S) = S'$ in solved form implies $\vec{S'}$ satisfies S.*

Proof.

1. From the fact that the rules of Fig. 6, when $\simeq$ is replaced by $\doteq$, are precisely the Robinson's unification rules.
2. From the previous point and the fact that, when $\simeq$ is replaced by $\approx$, multisets can be decomposed even if they have different cardinalities. ◀

3.2 Expansion

In order to deal with a set of equations in blocked form, we introduce an operation called expansion, working on principal derivations. Intuitively, an expansion modifies a principal derivation by replicating the minor premises of a rule $(\to_{\mathtt{E}i})$, increasing the cardinality of multisets, and updating the associated constraints accordingly.

▶ **Definition 17.**

- *Let $(\Pi, (E, V))$ be a principal derivation for M, and let x be a bound variable occurring in M. This means that in Π there is a rule:*

$$\frac{\Gamma, x : \sigma \vdash_{\mathtt{q}} N : B \qquad \sigma \subseteq \tau}{\Gamma \vdash_{\mathtt{q}} \lambda x.N : \tau \to B}\ (\to_{\mathtt{I}})$$

If there is also a subsequent rule:

$$\frac{\Gamma' \vdash_{\mathtt{q}} P : \tau \to B \qquad (\Delta_i \vdash_{\mathtt{q}} Q : C_i)_{1 \leq i \leq |\tau|}}{\Gamma' \uplus_{1 \leq i \leq |\tau|} \Delta_i \vdash_{\mathtt{q}} PQ : B}\ (\to_{\mathtt{E}1})$$

then we say that x is an applied bound variable in Π, and this last rule is called the application rule for x. Otherwise x is an unapplied bound variable in Π. We denote $\mathcal{B}$ the set of applied bound variables in Π.

- *Let $(\Pi, (E, V))$ be a principal derivation for M, and let x, y be applied bound variables in Π. If $M = \mathtt{C}[\lambda x.\mathtt{C}'[\lambda y.N]]$, then define $y < x$.*

▶ **Definition 18.** *An expansion, denoted $\mathtt{Expand}(\sigma, n)$, has two parameters: a multiset σ and a natural number $n \geq 1$. Applying $\mathtt{Expand}(\sigma, n)$ to a principal derivation $(\Pi, (E, V))$ means to perform the steps described below. There are four cases:*

1. *If there is a rule $(\to_{\mathtt{I}})$ with conclusion $\Gamma \vdash_{\mathtt{q}} \lambda x.N : \sigma \to A$, and x is an unapplied bound variable in Π, then:*
   - **a.** *replace, in the conclusion of the rule $(\to_{\mathtt{I}})$ and in the subsequent rules, the multiset σ by $\sigma \uplus [a_1, ..., a_n]$ where all the a_i are fresh;*
   - **b.** *recompute the system of constraints (E, V) according to Fig. 5, taking into account the increased cardinality.*
2. *If $\sigma = [A_1, ..., A_m]$ and in Π there is an elimination rule of shape*

$$\frac{\Gamma \vdash_{\mathtt{q}} P : B \qquad (\Sigma_i \triangleright \Delta_i \vdash_{\mathtt{q}} Q : A_i)_{1 \leq i \leq m}}{\Xi = \Gamma \uplus_{1 \leq i \leq m} \Delta_i \vdash_{\mathtt{q}} PQ : C} \ (\to_{\mathtt{E}j})$$

   perform the following actions:
   - **a.** *update the structure of Π by adding n disjoint copies of $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(Q)$ as the rightmost premises of the $(\to_{\mathtt{E}j})$ rule;*
   - **b.** *compute the new conclusion context Ξ' by taking into account the newly added premises, and replace Ξ by Ξ' in the conclusion of the rule $(\to_{\mathtt{E}j})$ and in all subsequent rules;*
   - **c.** *if $B = [a_1, ..., a_m] \to C$, i.e. if the rule was the application rule for some variable x, then replace, in the conclusion of the $(\to_{\mathtt{I}})$ rule abstracting x and in the subsequent rules, the multiset $[a_1, ..., a_m]$ by $[a_1, ..., a_m, a_{m+1}, ..., a_{m+n}]$, where $a_{m+1}, ..., a_{m+n}$ are fresh;*
   - **d.** *let $\mathcal{B}$ be the set of applied bound variables in Π, and let $\mathcal{L} = \mathcal{B} \cap \mathtt{dom}(\Delta_i)$.*
     While $\mathcal{L} \neq \emptyset$ do:
     - *let $\mathtt{min}(\mathcal{L})$ be the minimum element of $\mathcal{L}$ according to the $<$ relation of Definition 17, and let its application rule be:*

$$\frac{\Phi \vdash_{\mathtt{q}} S : \tau \to D \qquad (\Theta_i \triangleright \Psi_i \vdash_{\mathtt{q}} T : D_i)_{1 \leq i \leq s}}{\Upsilon = \Phi \uplus_{1 \leq i \leq s} \Psi_i \vdash_{\mathtt{q}} ST : D} \ (\to_{\mathtt{E}1})$$

       the modifications until this point certainly increased $|\tau|$, so $|\tau| > s$. Identify in τ the $|\tau| - s$ type variables that were introduced by the expansion, i.e. the set $\mathcal{F} = \{a \mid a \in \tau, a \notin \mathtt{Var}(V)\}$;
     - *update the structure of Π by adding $|\tau| - s$ disjoint copies of $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(T)$ as premises of the $(\to_{\mathtt{E}1})$ rule, matching them to the type variables in $\mathcal{F}$;*
     - *compute the new conclusion context Υ' by taking into account the newly added premises, and replace Υ by Υ' in the conclusion of the rule $(\to_{\mathtt{E}1})$ and in all subsequent rules;*
     - *update $\mathcal{L}$ by posing $\mathcal{L} = (\mathcal{L} - \mathtt{min}(\mathcal{L})) \cup (\mathcal{B} \cap \mathtt{dom}(\Theta_i))$;*
   - **e.** *recompute the system of constraints (E, V) according to Fig. 5, taking into account the new structure of the pseudo-derivation Π.*

3. *If* $\sigma = [a_1, ..., a_m]$ *and in* Π *there is a rule:*

$$\frac{\Gamma \vdash_{\mathtt{q}} P : [a_1, ..., a_m] \to B \qquad (\Sigma_i \rhd \Delta_i \vdash_{\mathtt{q}} Q : A_i)_{1 \leq i \leq m}}{\Gamma \uplus_{1 \leq i \leq m} \Delta_i \vdash_{\mathtt{q}} PQ : B} \; (\to_{\mathtt{E1}})$$

then perform $\mathtt{Expand}([A_1, ..., A_m], n)$.

4. *If none of the above conditions is met, the expansion behaves as the identity, i.e.* $\mathtt{Expand}(\sigma, n)(\Pi, (E, V)) = (\Pi, (E, V))$

▶ **Lemma 19.** *Let* $(\Pi, (E, V))$ *be a principal derivation for* M. *For all* σ, n, *the expansion* $\mathtt{Expand}(\sigma, n)(\Pi, (E, V))$ *terminates, returning a principal derivation for* M.

Proof. Both in cases (1) and (4) the expansion obviously stops without altering the structure of Π, and the result is coherent with the definition of principal derivation. Case (3) reduces to (2), hence we limit our analysis to such case. Clearly the structure of the output derivation differs from the input one, as some minor premises of elimination rules are extended with multiple copies of the original subderivations; however, since such copies introduce only fresh variables, all the new subderivations are disjoint, in agreement with the definition of principal derivation. We now prove that the number of while iterations is finite. The key observation is the following one: choosing the minimal element of $\mathcal{L}$ ensures that any applied bound variable is considered at most once, because a variable cannot re-enter the set $\mathcal{L}$ after being selected as $\mathtt{min}(\mathcal{L})$ and then discarded. To see this, let $x, y \in \mathcal{B}$; moreover let Σ be the subderivation whose conclusion is the $(\to_{\mathtt{I}})$ rule abstracting x, and let Σ' be the subderivation whose conclusion is the $(\to_{\mathtt{I}})$ rule abstracting y. Indeed, $y < x$ means that Σ' is a subderivation of Σ. Hence modifications performed by the expansion when the while loop selects y as $\mathtt{min}(\mathcal{L})$ may increment the premise of x in the context (note that x is possibly free in the subderivation being replicated), but not the other way around. We conclude that the number of while iterations is bounded by $|\mathcal{B}|$. ◀

Thanks to the expansion, the notion of solvability can be extended to principal derivations. We stress that the considerations made in Remark 6 apply also to the present case, and therefore we adopt the same conventions.

▶ **Definition 20.** *Let* $\mathtt{PD_i}(M) = (\Pi \rhd \Gamma \vdash_{\mathtt{q}} M : A, (E, V))$. *A solution of* $\mathtt{PD_i}(M)$ *is a pair* $(\bar{e}, \phi)$, *where* $\bar{e}$ *is sequence of expansions such that* $\bar{e}(\Pi, (E, V)) = (\Pi' \rhd \Gamma' \vdash_{\mathtt{q}} M : A', (E', V'))$, *and* $\phi : \mathtt{Var} \to \mathcal{T_U}$ *is a solution of* E' *w.r.t.* V', *i.e.* $\phi(\Pi') \rhd \phi(\Gamma') \vdash_{\mathtt{i}} M : \phi(A')$.

For the sake of simplicity, from now on if $\bar{e}(\Pi, (E, V)) = (\Pi', (E', V'))$ we will also denote Π' by $\bar{e}(\Pi)$, E' by $\bar{e}(E)$ and V' by $\bar{e}(V)$.

3.3 The algorithm

In this subsection we present an algorithm $\mathtt{Solve}$ which tries to solve the system of constraints generated by $\mathtt{PD_i^{min}}(M)$. A solution, if it exists, is found by interleaving unification rules and expansions. In $\mathtt{Solve}$ we make use of the following notations:

- Let S be a set of equivalences. $\mathtt{Fail}(S) = \mathtt{true}$ iff S is in unsolvable form.
- Let S be a set of equations. $\mathtt{Blocked}(S) = \mathtt{true}$ iff S is in blocked form.

Informally, the algorithm behaves as follows. $\mathtt{QuasiUnify}_{\approx}(E/_{\approx} \cup V)$ is used to check if the constraints of V can be respected by E; in case of failure, there is no solution. Otherwise, the algorithm tries to solve E. If a blocked form is reached, a critical equation is dealt with using the operation of expansion. At every expansion, the system of constraints (E, V) is

Algorithm 1 The algorithm `Solve`. Input: a term M.

```
function Solve($M$)
    $(\Pi, (E, V)) \leftarrow \mathtt{PD}_{\mathtt{i}}^{\mathtt{min}}(M)$
    $C \leftarrow \mathtt{QuasiUnify}_{\approx}(E/_{\approx} \cup V)$
    if $\mathtt{Fail}(C)$ then FAIL
    end if
    $E^* \leftarrow \mathtt{QuasiUnify}_{\doteq}(E)$
    while $\mathtt{Blocked}(E^*)$ do
        choose $(\sigma \doteq \tau) \in E^*$ such that $|\sigma| \neq |\tau|$
        if $|\sigma| > |\tau|$ then
            $n \leftarrow |\sigma| - |\tau|$
            $(\Pi, (E, V)) \leftarrow \mathtt{Expand}(\tau, n)(\Pi, (E, V))$
        else
            $n \leftarrow |\tau| - |\sigma|$
            $(\Pi, (E, V)) \leftarrow \mathtt{Expand}(\sigma, n)(\Pi, (E, V))$
        end if
        $E^* \leftarrow \mathtt{QuasiUnify}_{\doteq}(E)$
    end while
    $V^* \leftarrow \mathtt{QuasiUnify}_{\approx}(\vec{E^*}(V))$
    $S \leftarrow \mathtt{QuasiUnify}_{\doteq}(E^* \cup V^*/_{\doteq})$
    return $(\Pi, \vec{S})$
end function
```

recomputed, then $\mathtt{QuasiUnify}_{\doteq}(E)$ is applied again; the loop stops as soon as a solved form is reached. Note that since the system of constraints is recomputed after each expansion, the only purpose of the calls to $\mathtt{QuasiUnify}_{\doteq}(E)$ in the while loop is to expose critical equations, thus guiding the expansions. Lastly, the final calls to $\mathtt{QuasiUnify}_{\approx}$ and $\mathtt{QuasiUnify}_{\doteq}$ generate the unifying substitution. The algorithm is non-deterministic, as critical equations are randomly chosen. Remark that, in an actual implementation, the efficiency of `Solve` could be improved by avoiding unnecessary and time consuming steps, such as recomputing the system of constraints from scratch after each expansion and repeating the whole unification procedure: this would require keeping track of the modified/added constraints only. Here we favoured both clarity and brevity of exposition over efficiency; we leave more detailed considerations about the time complexity, as well as possible optimizations, to future work. In what follows we prove that the algorithm behaves correctly.

▶ Remark 21. `Solve` does not find all possible solutions of $\mathtt{PD}_{\mathtt{i}}^{\mathtt{min}}(M)$, since expansions are used only when strictly necessary (i.e. when the set of equations E^* is in blocked form). Note also that not all cases considered in Definition 18 are actually needed for the purposes of the algorithm: in particular case 4 never applies, but has been included for completeness.

Let A be an intersection type. The syntactic tree of A, written $T(A)$, is a tree defined inductively as follows: if $A = a$, then $T(a)$ is a single node named a; otherwise, if $A = [A_1, ..., A_n] \to B$, $T(A)$ is an ordered tree whose root, labelled A, has $n + 1$ children, namely $T(A_1), ..., T(A_n)$ and $T(B)$. Two subtypes of A and B are *corresponding* if they occur at the same path in $T(A)$ and $T(B)$.

▶ **Lemma 22.** *Let* $\mathtt{PD}_{\mathtt{i}}^{\mathtt{min}}(M) = (\Pi \rhd \Gamma \vdash_{\mathtt{q}} M : A, (E, V))$ *and* $C = \mathtt{QuasiUnify}_{\approx}(E/_{\approx} \cup V)$. *If C is in unsolvable form then* $\mathtt{PD}_{\mathtt{i}}^{\mathtt{min}}(M)$ *is unsolvable.*

Proof. C in unsolvable form means that it contains a circular constraint, let it be $b \approx B$, such that b occurs in B. If E is unsolvable, i.e. b and B are corresponding subtypes of two equated intersection types, then the result is obtained by observing that no amount of expansions can get rid of the circular equation: indeed, an expansion can add new constraints, but not erase the existing ones. On the other hand, assume E solvable, i.e. there is a sequence s of expansions and applications of unification rules such that $s(E)$ is in solved form. Then the circularity has been introduced by the constraints in V, that is, b and B must be equivalent to each other. But by definition of $\sim$, it does not exist ϕ such that $\phi(b) \sim \phi(B)$, because if b occurs in B, $\phi(b)$ and $\phi(B)$ have syntactic trees of different depths, for all ϕ; again, no amount of expansions can get rid of the circularity. ◀

▶ **Theorem 23** (Partial Correctness). *Let* $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(M) = (\Pi \triangleright \Gamma \vdash_{\mathtt{q}} M : A, (E, V))$ *and assume* $\mathtt{Solve}(M)$ *terminates. If the result is FAIL then* M *is not typable in* $\mathcal{U}$*; otherwise the algorithm outputs* $(\Sigma \triangleright \Gamma' \vdash_{\mathtt{q}} M : A', \phi)$ *such that* $\phi(\Sigma) \triangleright \phi(\Gamma') \vdash_{\mathtt{i}} M : \phi(A')$.

Proof. Let $C = \mathtt{QuasiUnify}_{\approx}(E/_{\approx} \cup V)$. Then C is either in solvable or unsolvable form. In the latter case $\mathtt{Solve}$ immediately returns *FAIL*, and the result follows by Lemma 22. If C is in solved form, the algorithm continues by performing an interleaved sequence s of unification rules and expansions. First we show that, for all sequences s, both $\mathtt{QuasiUnify}_{\approx}(s(E)/_{\approx} \cup s(V))$ and $\mathtt{QuasiUnify}_{\doteq}(s(E))$ do no not contain circular constraints. The unification rules play no role in the introduction or elimination of circular constraints, hence we limit our analysis to the sequence of expansions $\bar{e}$ belonging to s. It is easy to check that no circularity can be introduced as a result of an expansion: indeed, each expansion creates disjoint replicas of sub-pseudo-derivations which were part of $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(M)$, thus it can only originate fresh copies of constraints that already existed from the beginning. We conclude by observing that the equations that can be generated during a call to $\mathtt{QuasiUnify}_{\doteq}(\bar{e}(E))$ form a subset of the constraints that can be generated during a call to $\mathtt{QuasiUnify}_{\approx}(\bar{e}(E)/_{\approx} \cup \bar{e}(V))$. The last thing to do is proving that if the check on C does not fail, then $\mathtt{Solve}$ outputs a pair (Σ, ϕ) with the desired properties. Since we assumed that $\mathtt{Solve}(M)$ terminates, the number of iterations of the while loop is finite; this means that the algorithm builds a principal derivation $(\Sigma, (E_\Sigma, V_\Sigma))$ such that $E^*_\Sigma = \mathtt{QuasiUnify}_{\doteq}(E_\Sigma)$ is in solved form. Clearly $\mathtt{dom}(\vec{E}^*_\Sigma) \cap \mathtt{Var}(\vec{E}^*_\Sigma(V_\Sigma)) = \emptyset$, so $V^* = \mathtt{QuasiUnify}_{\approx}(\vec{E}^*_\Sigma(V_\Sigma))$ induces a substitution $\vec{V}^*$ whose domain and codomain are disjoint from the domain of $\vec{E}^*_\Sigma$, that is, $(\mathtt{dom}(\vec{V}^*) \cup \mathtt{cod}(\vec{V}^*)) \cap \mathtt{dom}(\vec{E}^*_\Sigma) = \emptyset$. Therefore $S = \mathtt{QuasiUnify}_{\doteq}(E^*_\Sigma \cup V^*/_{\doteq})$ is in solved form. As $\vec{E}^*_\Sigma$ solves E^*_Σ and $\vec{V}^*$ satisfies V_Σ, it is easy to verify that $\phi = \vec{S}$ is a solution of E_Σ w.r.t V_Σ. Then $\phi(\Sigma)$ is a correct derivation, because (E_Σ, V_Σ) consists of all and only the constraints needed to convert a pseudo-derivation into an actual derivation. ◀

The following two results guarantee that the algorithm does not perform an infinite sequence of expansions and unification rules. Moreover they show that expansions and β-reductions are closely related.

▶ **Lemma 24.** *If* M *is in normal form then* $\mathtt{Solve}(M)$ *stops without performing any expansion.*

Proof. By induction on M. Recall that a normal form is defined by the following grammar:

$$M, N \quad ::= \quad \lambda x.M \mid x \underbrace{M ... M}_{n} \quad n \geq 0$$

If $M = x$ the proof is trivial, as $E = \emptyset$. $M = \lambda x.N$ follows by induction, as no new equation is added. Lastly, consider the case $M = xM_1...M_n$ for $n > 0$. Let $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(x) = (\Pi_0 \rhd x : [a_0] \vdash_{\mathtt{q}} x : a_0, (\emptyset, \emptyset))$ and $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(M_i) = (\Pi_i, (E_i, V_i))$ where $\Pi_i \rhd \Gamma_i \vdash_{\mathtt{q}} M_i : A_i$ $(1 \leq i \leq n)$. Then $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(M) = (\Pi, (E, V))$, where $E = \bigcup_{1 \leq i \leq n} E_i \cup \{a_0 \doteq [A_1] \rightarrow a_1,\ a_1 \doteq [A_2] \rightarrow a_2,\ ...,\ a_{n-1} \doteq [A_n] \rightarrow a_n\}$ and $a_1, \ldots, a_n$ are fresh. If the check on $\mathtt{QuasiUnify}_{\approx}(E/_{\approx} \cup V)$ fails, the algorithm immediately stops; otherwise it computes $\mathtt{QuasiUnify}_{\doteq}(E)$. Since by induction E_i does not generate critical equations, neither does E, because a_0 does not occur in E_i and $a_1, \ldots, a_n$ are fresh. As $\mathtt{QuasiUnify}_{\doteq}(E)$ does not contain critical equations, the algorithm exits the while loop without performing any expansion. ◀

▶ **Theorem 25** (Partial Termination). *Let M be strongly normalizing. Then* $\mathtt{Solve}(M)$ *terminates.*

Proof. The proof is in the Appendix. ◀

4 Some examples

▶ **Example 26.** In order to understand the necessity and the behaviour of expansions, consider the term $M = ((\lambda y.\lambda x.yxx)(\lambda p.\lambda r.wpp))(tz)$. The associated minimal principal derivation is $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(M) = (\Pi_M, (E, V))$, where:

- Π is:

$$\dfrac{\dfrac{\dfrac{\dfrac{y : [a_1] \vdash_{\mathtt{q}} y : a_1 \qquad x : [b_1] \vdash_{\mathtt{q}} x : b_1}{y : [a_1], x : [b_1] \vdash_{\mathtt{q}} yx : c_1} \qquad x : [b_2] \vdash_{\mathtt{q}} x : b_2}{y : [a_1], x : [b_1, b_2] \vdash_{\mathtt{q}} yxx : d_1}}{y : [a_1] \vdash_{\mathtt{q}} \lambda x.yxx : [b_1, b_2] \rightarrow d_1}}{\Pi \rhd\ \vdash_{\mathtt{q}} \lambda y.\lambda x.yxx : [a_1] \rightarrow [b_1, b_2] \rightarrow d_1}$$

  Σ is:

$$\dfrac{\dfrac{\dfrac{\dfrac{w : [e_1] \vdash_{\mathtt{q}} w : e_1 \qquad p : [f_1] \vdash_{\mathtt{q}} p : f_1}{w : [e_1], p : [f_1] \vdash_{\mathtt{q}} wp : g_1} \qquad p : [f_2] \vdash_{\mathtt{q}} p : f_2}{w : [e_1], p : [f_1, f_2] \vdash_{\mathtt{q}} wpp : h_1}}{w : [e_1], p : [f_1, f_2] \vdash_{\mathtt{q}} \lambda r.wpp : [k_1] \rightarrow l_1}}{\Sigma \rhd w : [e_1] \vdash_{\mathtt{q}} \lambda p.\lambda r.wpp : [f_1, f_2] \rightarrow [k_1] \rightarrow l_1}$$

  Θ_1 is:

$$\dfrac{t : [m_1] \vdash_{\mathtt{q}} t : m_1 \qquad z : [n_1] \vdash_{\mathtt{q}} z : n_1}{\Theta_1 \rhd t : [m_1], z : [n_1] \vdash_{\mathtt{q}} tz : o_1}$$

  Θ_2 is a disjoint instance of Θ_1, and lastly Π_M is:

$$\dfrac{\dfrac{\Pi \qquad \Sigma}{\vdash_{\mathtt{q}} (\lambda y.\lambda x.yxx)(\lambda p.\lambda r.wpp) : [b_1, b_2] \rightarrow d_1} \qquad \Theta_1 \qquad \Theta_2}{\Pi_M \rhd w : [e_1], t : [m_1, m_2], z : [n_1, n_2] \vdash_{\mathtt{q}} ((\lambda y.\lambda x.yxx)(\lambda p.\lambda r.wpp))(tz) : d_1}$$

- $E = \{a_1 \doteq [b_1] \rightarrow c_1,\ c_1 \doteq [b_2] \rightarrow d_1,\ e_1 \doteq [f_1] \rightarrow g_1,\ g_1 \doteq [f_2] \rightarrow h_1,\ a_1 \doteq [f_1, f_2] \rightarrow [k_1] \rightarrow l_1, m_1 \doteq [n_1] \rightarrow o_1,\ m_2 \doteq [n_2] \rightarrow o_2,\ b_1 \doteq o_1, b_2 \doteq o_2\}$.
- $V = \{b_1 \approx b_2,\ f_1 \approx f_2,\ m_1 \approx m_2,\ n_1 \approx n_2\}$.

Applying the unification rules to E, we obtain equation $[b_1] \to c_1 \doteq [f_1, f_2] \to [k_1] \to l_1$; decomposing yields the critical equation $[b_1] \doteq [f_1, f_2]$. Therefore we need to perform $\mathtt{Expand}(\sigma, n)$ with $\sigma = [b_1]$ and $n = 1$. Under the action of the expansion, the derivation Π becomes:

$$\dfrac{\dfrac{\dfrac{\dfrac{y : [a_1] \vdash_{\mathtt{q}} y : a_1 \qquad x : [b_1] \vdash_{\mathtt{q}} x : b_1 \qquad x : [b_3] \vdash_{\mathtt{q}} x : b_3}{y : [a_1], x : [b_1, b_3] \vdash_{\mathtt{q}} yx : c_1} \qquad x : [b_2] \vdash_{\mathtt{q}} x : b_2}{y : [a_1], x : [b_1, b_3, b_2] \vdash_{\mathtt{q}} yxx : d_1}}{y : [a_1] \vdash_{\mathtt{q}} \lambda x.yxx : [b_1, b_3, b_2] \to d_1}}{\Pi \triangleright \vdash_{\mathtt{q}} \lambda y.\lambda x.yxx : [a_1] \to [b_1, b_3, b_2] \to d_1}$$

Consequently Π_M becomes:

$$\dfrac{\dfrac{\Pi \qquad \Sigma}{\vdash_{\mathtt{q}} (\lambda y.\lambda x.yxx)(\lambda p.\lambda r.wpp) : [b_1, b_3, b_2] \to d_1} \qquad \Theta_1 \quad \Theta_2}{\Pi_M \triangleright w : [e_1], t : [m_1, m_2], z : [n_1, n_2] \vdash_{\mathtt{q}} ((\lambda y.\lambda x.yxx)(\lambda p.\lambda r.wpp))(tz) : d_1}$$

At this point Π_M is no longer a principal derivation for M, as its last rule is incorrect. But the expansion is not terminated yet: there is one applied bound variable in the context of the duplicated subderivation, namely x. Therefore we enter the while loop with $\mathcal{L} = \{x\}$, and since the application rule for x is the last rule of Π_M, we update the derivation as follows:

$$\dfrac{\dfrac{\Pi \qquad \Sigma}{\vdash_{\mathtt{q}} (\lambda y.\lambda x.yxx)(\lambda p.\lambda r.wpp) : [b_1, b_3, b_2] \to d_1} \qquad \Theta_1 \quad \Theta_3 \quad \Theta_2}{\Pi_M \triangleright w : [e_1], t : [m_1, m_3, m_2], z : [n_1, n_3, n_2] \vdash_{\mathtt{q}} ((\lambda y.\lambda x.yxx)(\lambda p.\lambda r.wpp))(tz) : d_1}$$

where Θ_3 is yet another disjoint instance of Θ_1. No applied bound variable is in the context of Θ_i, so $\mathcal{L} = \emptyset$ and we exit the while loop. Lastly we recompute the constraints, which now are:

- $E = \{a_1 \doteq [b_1, b_3] \to c_1,\ c_1 \doteq [b_2] \to d_1,\ e_1 \doteq [f_1] \to g_1,\ g_1 \doteq [f_2] \to h_1,\ a_1 \doteq [f_1, f_2] \to [k_1] \to l_1, m_1 \doteq [n_1] \to o_1,\ m_2 \doteq [n_2] \to o_2,\ m_3 \doteq [n_3] \to o_3,\ b_1 \doteq o_1, b_2 \doteq o_2,\ b_3 \doteq o_3\}$.
- $V = \{b_1 \approx b_2,\ b_2 \approx b_3,\ f_1 \approx f_2,\ m_1 \approx m_2,\ m_2 \approx m_3,\ n_1 \approx n_2,\ n_2 \approx n_3, \ldots\}$.

The reader can infer the omitted equivalence constraints by considering the equivalence relation induced by V. Note that all the constraints generated by Θ_i have been duplicated. Now E can be reduced to solved form by decomposing equation $[b_1, b_3] \doteq [f_1, f_2]$.

▶ **Example 27.** Let $M = \lambda x.xx$, which is not simply typable. $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(M) = (\Pi_M, (E, V))$ where:

$$\dfrac{\dfrac{x : [a] \vdash_{\mathtt{q}} x : a \qquad x : [b] \vdash_{\mathtt{q}} x : b}{x : [a, b] \vdash_{\mathtt{q}} xx : c}}{\Pi_M \triangleright \vdash_{\mathtt{q}} \lambda x.xx : [a, b] \to c}$$

$E = \{a \doteq [b] \to c\}$ and $V = \{a \approx b\}$. Clearly E is unsolvable w.r.t. V.

5 From simple types to intersection types and viceversa

In this section we will prove the main result of the paper, namely the correspondence between the two systems $\mathcal{S}$ and $\mathcal{U}$. First we define a translation from intersection types to simple types, which erases the multisets in $\mathcal{I}$.

▶ **Definition 28.** *The collapse translation t from $\mathcal{I}$ to $\mathcal{T}_\mathcal{S}$ is defined by induction on the size of types in the following way (recall that multisets are now ordered):*

$$\begin{array}{lcl} t(a) & = & a; \\ t(\sigma \to A) & = & t(\sigma) \to t(A); \\ t([A_1, ..., A_n]) & = & t(A_1). \end{array}$$

The following property is easy to prove, by induction on types.

▶ **Property 29.** *$A \sim B$ implies $t(A) = t(B)$.*

Observe that in the translation of an (ordered) multiset we could have chosen any of its elements, in a nondeterministic way. However, always choosing the first one allows us to easily extend the translation to pseudo-derivations and derivations.

▶ **Definition 30.**

- *$t(.)$ is extended to contexts by posing $t(\Gamma) = \{x : t(\sigma) \mid x : \sigma \in \Gamma\}$.*
- *$t(.)$ is extended to pseudo-derivations in the following way:*
  - *if $\Pi \triangleright x : [a] \vdash_\mathtt{q} x : a$, then $t(\Pi) \triangleright x : t([a]) \vdash_\mathtt{p} x : t(a)$;*
  - *the case the subject is $\lambda x.M$ is straightforward.*
  - *let Π be:*

$$\frac{\Gamma \vdash_\mathtt{q} M : A \qquad (\Delta_i \vdash_\mathtt{q} N : B_i)_{1 \leq i \leq n}}{\Gamma \uplus_{1 \leq i \leq n} \Delta_i \vdash_\mathtt{q} MN : C} \ (\to_{\mathtt{E}j})$$

then $t(\Pi)$ is:

$$\frac{t(\Gamma) \vdash_\mathtt{p} M : t(A) \qquad t(\Delta_1) \vdash_\mathtt{p} N : t(B_1)}{t(\Gamma) \cup t(\Delta_1) \vdash_\mathtt{p} MN : t(C)} \ (\to_{\mathtt{E}j})$$

- *$t(.)$ can be extended to principal derivations by posing $t(\Pi, (E, V)) = (t(\Pi), (E_{t(\Pi)}, V_{t(\Pi)}))$, where the system of constraints $(E_{t(\Pi)}, V_{t(\Pi)})$ is computed from the pseudo-derivation $t(\Pi)$ according to Fig. 2, as usual.*
- *$t(.)$ is extended to derivations in the same way as for pseudo-derivations.*

▶ **Property 31.** *Let $\mathtt{PD_i}(M) = (\Pi, (E, V))$. For each sequence of expansion $\bar{e}$, $\mathtt{PD}(M) = (t(\bar{e}(\Pi)), (E_{t(\bar{e}(\Pi))}, V_{t(\bar{e}(\Pi))}))$.*

Proof. In case $\bar{e}$ is the empty sequence the proof follows by induction on the definition of $\mathtt{PD}(M)$ and $\mathtt{PD_i}(M)$, recalling that $\mathtt{PD}(M)$ is defined modulo renaming of type variables. The general case follows from the definition of expansion. ◀

We are now able to prove the decidability of $\mathcal{U}$, and the fact that the systems $\mathcal{S}$ and $\mathcal{U}$ have the same typability power.

▶ **Theorem 32.** *The system $\mathcal{U}$ is decidable.*

Proof. Let $\mathtt{PD_i^{min}}(M) = (\Pi, (E, V))$ and $\mathtt{PD}(M) = (t(\Pi), (E_{t(\Pi)}, V_{t(\Pi)}))$. We already proved that $\mathtt{Solve}(M)$ terminates with a correct result in case M is strongly normalizing (Theorems 23 and 25). If M is not strongly normalizing, then it is not simply typable, i.e. $\mathtt{Unify}(E_{t(\Pi)} \cup V_{t(\Pi)})$ fails. Observe that $\mathtt{QuasiUnify}_\approx$ decomposes multiset equivalences by introducing equivalence constraints between all possible pairs of elements, hence it generates a circular constraint exactly when $\mathtt{Unify}$ does. We conclude that the failure of $\mathtt{Unify}(E_{t(\Pi)} \cup V_{t(\Pi)})$ implies the failure of the check on $\mathtt{QuasiUnify}_\approx(E/_\approx \cup V)$ performed at the beginning of the algorithm $\mathtt{Solve}$. ◀

▶ **Corollary 33.** $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(M)$ *is solvable if and only if* $\mathtt{PD}(M)$ *is solvable.*

Proof. Let $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(M) = (\Pi, (E, V))$. $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(M)$ is solvable if and only if (by Theorem 23) the check on $\mathtt{QuasiUnify}_{\approx}(E/_{\approx} \cup V)$ succeeds if and only if (looking at the proof of Theorem 32) $\mathtt{PD}(M)$ is solvable. ◀

The main result

The main result of this paper is the correspondence between the derivations in the two systems $\mathcal{S}$ and $\mathcal{U}$: from one side the translation of every derivation in $\mathcal{U}$ is a derivation in $\mathcal{S}$, and on the other side for each derivation Π in $\mathcal{S}$ there is an infinite set of derivations in $\mathcal{U}$ from which Π can be obtained through the translation.

▶ **Theorem 34.**

1. $\Gamma \vdash_{\mathtt{i}} M : A$ *implies* $t(\Gamma) \vdash M : t(A)$*;*
2. $\Pi \rhd \Gamma \vdash M : A$ *implies there exists* $\Pi_{\mathtt{i}} \rhd \Gamma_{\mathtt{i}} \vdash_{\mathtt{i}} M : A_{\mathtt{i}}$ *such that* $\Gamma = t(\Gamma_{\mathtt{i}})$ *and* $A = t(A_{\mathtt{i}})$*.*

Proof.

1. Easy, by definition of collapse translation.
2. First we consider the case $\mathtt{dom}(\Gamma) = \mathtt{FV}(M)$. Let $\mathtt{PD}(M) = (\Sigma, (E_\Sigma, V_\Sigma))$; clearly $\Pi \rhd \Gamma \vdash M : A$ implies that there is θ m.g.u. of $E_\Sigma \cup V_\Sigma$, and there is θ' such that $\theta' \circ \theta(\Sigma) = \Pi$. Moreover, by Corollary 33, $\mathtt{Solve}(M) = (\Sigma_{\mathtt{i}}, \phi)$; let $(E_{\Sigma_{\mathtt{i}}}, V_{\Sigma_{\mathtt{i}}})$ be the system of constraints associated to $\Sigma_{\mathtt{i}}$. It is easy to verify that $r \circ t \circ \phi(\Sigma_{\mathtt{i}}) = \theta(\Sigma)$ for some renaming of type variables $r : \mathtt{Var} \to \mathtt{Var}$ (trivially extended to derivations): this follows from the fact that both systems of constraints, namely (E_Σ, V_Σ) for simple types and $(E_{\Sigma_{\mathtt{i}}}, V_{\Sigma_{\mathtt{i}}})$ for uniform intersection types, contain all and only the constraints strictly needed to type M in the corresponding type system. Then, choosing any substitution $\phi' : \mathtt{Var} \to \mathcal{T}_{\mathcal{U}}$ such that $t \circ \phi' = \theta'$, one can build $\Pi_{\mathtt{i}} = \phi' \circ r \circ \phi(\Sigma_{\mathtt{i}})$. In case there is $x : B \in \Gamma$ such that $x \notin \mathtt{FV}(M)$, starting from the above construction and suitably exploiting $(\mathtt{var})$ rules it is always possible to obtain a derivation $\Pi_{\mathtt{i}}$ such that $x : \sigma \in \Gamma_{\mathtt{i}}$ and $t(\sigma) = B$. ◀

6 Conclusion

Starting from a non-idempotent intersection type assignment system which is undecidable, as it characterises strong normalisation, we have built a decidable restriction of it. More precisely, we designed an algorithm $\mathtt{Solve}$ that, given in input a λ-term, outputs either FAIL, if the term cannot be typed, or the most general typing for it. System $\mathcal{U}$ has the same typability power as the simple type assignment system, and the derivations in the two systems are related through a translating function. Furthermore, the system is quantitative, in the sense that some information about the number of occurrences of a variable in the subject can be deduced from a derivation in system $\mathcal{U}$. In the future we plan to use this new system to reason about properties related to the complexity of reductions of simply typed terms. Moreover we would like to investigate the existence of further decidable restrictions with interesting properties. From a technical point of view, we also aim at studying the time and space complexity of $\mathtt{Solve}$ in more detail, in order to come up with an efficient implementation.

References

1 Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in logic and the foundation of mathematics*. North-Holland, Amsterdam, 1984.

2 Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter model and the completeness of type assignment. *J. Symb. Log.*, 48(4):931–940, 1983. doi:10.2307/2273659.

3 Choukri-Bey Ben-Yelles. *Type-assignment in the lambda-calculus; syntax and semantics*. PhD thesis, University of Wales Swansea, 1979.

4 Erika De Benedetti and Simona Ronchi Della Rocca. A type assignment for lambda-calculus complete for fptime and strong normalization. *Inf. Comput.*, 248(195-214):195–214, 2016. doi:10.1016/j.ic.2015.12.012.

5 Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. Not enough points is enough. In Jacques Duparc and Thomas A. Henzinger, editors, *CSL 2007*, volume 4646, pages 298–312, 2007.

6 Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. Inhabitation for non-idempotent intersection types. *Log. Methods Comput. Sci.*, 14(3), 2018. doi:10.23638/LMCS-14(3:7)2018.

7 Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the lambda-calculus. *Logic Journal of the IGPL*, 25(4):431–464, 2017.

8 Alonzo Church. A formulation of simple theory of types. *J. Symbolic Logic*, 5:56–68, 1940.

9 Mario Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame J. Form. Log.*, 21(4):685–693, 1980.

10 Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. Functional characters of solvable terms. *Math. Log. Q.*, 27(2-6):45–58, 1981.

11 Mario Coppo and Paola Giannini. Principal types and unification for a simple intersection type system. *Inf. Comput.*, 122(1):70–96, 1995.

12 Daniel de Carvalho. Execution time of lambda-terms via denotational semantics and intersection types. *CoRR*, abs/0905.4251, 2009.

13 Roger J. Hyndley. *Basic Simple Type theory*. Hoepli, 1997.

14 Delia Kesner and Daniel Ventura. Quantitative types for the linear substitution calculus. In *TCS*, LNCS, 2014.

15 Assaf J. Kfoury and Joe B. Wells. Principality and decidable type inference for finite-rank intersection types. In Andrew W. Appel and Alex Aiken, editors, *POPL '99, Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Antonio, TX, USA, January 20-22, 1999*, pages 161–174. ACM, 1999.

16 Assaf J. Kfoury and Joe B. Wells. Principality and type inference for intersection types using expansion variables. *Theor. Comput. Sci.*, 311(1-3):1–70, 2004.

17 Luca Paolini, Mauro Piccolo, and Simona Ronchi Della Rocca. Essential and relational models. *Mathematical Strucures in Computer Science*, 27, 2017.

18 John A. Robinson. A machine-oriented logic based on the resolution principle. *J. Asoc. for Computing Machinery 12 (1965)*, 12:23–41, 1965.

19 Simona Ronchi Della Rocca. Principal type scheme and unification for intersection type discipline. *Theor. Comput. Sci.*, 59:181–209, 1988. doi:10.1016/0304-3975(88)90101-6.

20 Simona Ronchi Della Rocca and Luca Paolini. *The Parametric Lambda Calculus - A Metamodel for Computation*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

21 Simona Ronchi Della Rocca and Betti Venneri. Principal type schemes for an extended type theory. *Theor. Comput. Sci.*, 28:151–169, 1984. doi:10.1016/0304-3975(83)90069-5.

22 Pawel Urzyczyn. The emptiness problem for intersection types. In *Proceedings of the Ninth Annual Symposium on Logic in Computer Science (LICS '94), Paris, France, July 4-7, 1994*, pages 300–309. IEEE Computer Society, 1994. doi:10.1109/LICS.1994.316059.

A Proof of Theorem 25

Proof (Hint). In order to develop some useful intuitions, we start by considering M that reduces to N in one step, by reducing one of its innermost redexes; we denote this fact by writing $M \to^{\mathtt{in}}_{\beta} N$. Let $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(M) = (\Pi_M, (E_M, V_M))$, $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(N) = (\Pi_N, (E_N, V_N))$, $M = \mathtt{C}[(\lambda x.P)Q]$ and $N = \mathtt{C}[P[Q/x]]$. Assume $\mathtt{QuasiUnify}_{\approx}(E_M/_{\approx} \cup V_M)$ is in solved form, termination being obvious in the other case. We will prove that there is a run of $\mathtt{Solve}(M)$ performing an interleaved sequence s of unification rules and expansions such that $E_N \subset s(E_M)$ and $V_N \subset \bar{e}(V_M)$, where $\bar{e}$ is the sequence of expansions belonging to s. At the same time we will show that $s(E_M) - E_N$ is in solved form and cannot generate critical equations. The proof depends on the occurrences of x in P.

x does not occur in P. Let Π_M contain subderivations $\Pi^j_{(\lambda x.P)Q}$ $(j \in J)$ of shape:

$$(\Pi^j_{(\lambda x.P)Q}) \qquad \dfrac{\dfrac{\Gamma^j \vdash_{\mathtt{q}} P : B^j}{\Pi^j_{\lambda x.P} \triangleright \Gamma^j \vdash_{\mathtt{q}} \lambda x.P : [a^j] \to B^j} \qquad \Pi^j_Q \triangleright \Delta^j \vdash_{\mathtt{q}} Q : A^j}{\Gamma^j \uplus \Delta^j \vdash_{\mathtt{q}} (\lambda x.P)Q : B^j}$$

where $(\Pi^j_{\lambda x.P}, (E^j_{\lambda x.P}, V^j_{\lambda x.P}))$ and $(\Pi^j_Q, (E^j_Q, V^j_Q))$ are disjoint instances of $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(\lambda x.P)$ and $\mathtt{PD}^{\mathtt{min}}_{\mathtt{i}}(Q)$ respectively. Moreover let $E_M = E' \cup_{j \in J} (E^j_{\lambda x.P} \cup E^j_Q \cup E^j_{\text{app}})$ and $V_M = V' \cup_{j \in J} (V^j_{\lambda x.P} \cup V^j_Q \cup V^j_{\text{app}})$, where $E^j_{\text{app}} = \{a^j \doteq A^j\}$ and $V^j_{\text{app}} = \{b^j \approx c^j \mid y \in \mathtt{dom}(\Gamma^j) \cap \mathtt{dom}(\Delta^j), b^j \in \Gamma^j(y), c^j \in \Delta^j(y)\}$. First, observe that $E_N = E' \cup_{j \in J} E^j_P = E' \cup_{j \in J} E^j_{\lambda x.P}$ and $V_N = V' \cup_{j \in J} V^j_P \subset V_M$. Since Q is in normal form, by Lemma 24, for all j we know that $E^{j*}_Q = \mathtt{QuasiUnify}_{\doteq}(E^j_Q)$ is in solved form. Let $\bar{u}$ be the sequence of unification rules performed by all $\mathtt{QuasiUnify}_{\doteq}(E^j_Q)$ $(j \in J)$; clearly each a^j occurs only in E^j_{app}, hence $\bar{u}(E_M) - E_N = \bigcup_{j \in J} E^{j*}_Q \cup E^j_{\text{app}}$ is in solved form and cannot generate critical equations. Note that in this case $\bar{e}$ is the empty sequence.

x occurs in P. Let Π_M contain subderivations $\Pi^j_{(\lambda x.P)Q}$ $(j \in J)$ ending by:

$$(\Pi^j_{(\lambda x.P)Q}) \qquad \dfrac{\dfrac{\Gamma^j, x : [a^j_i]_{i \in I^j} \vdash_{\mathtt{q}} P : B^j}{\Gamma^j \vdash_{\mathtt{q}} \lambda x.P : [a^j_i]_{i \in I^j} \to B^j} \qquad (\Delta^j_i \vdash_{\mathtt{q}} Q : A^j_i)_{i \in I^j}}{\Gamma^j \uplus_{i \in I^j} \Delta^j_i \vdash_{\mathtt{q}} (\lambda x.P)Q : B^j}$$

where I^j is a suitable set of indexes. Focusing on the occurrences of x in functional position in P, each $\Pi^j_{(\lambda x.P)Q}$ contains disjoint subderivations of shape:

$$(\Sigma^j_i) \qquad \dfrac{x : [a^j_i] \vdash_{\mathtt{q}} x : a^j_i \qquad \Psi^j_i \vdash_{\mathtt{q}} R_i : C^j_i}{\{x : [a^j_i]\} \uplus \Psi^j_i \vdash_{\mathtt{q}} xR_i : e^j_i}$$

where $i \in I^j_{\text{fun}} \subseteq I^j$ (recall that, by construction, each application uses a different premise on x). The set of equations generated by the last rule of all the aforementioned Σ^j_i is:

$$E_{\text{fun}} = \{a^j_i \doteq [C^j_i] \to e^j_i \mid j \in J, i \in I^j_{\text{fun}}\}$$

Focusing instead on the occurrences of x in argument position, each $\Pi^j_{(\lambda x.P)Q}$ contains disjoint subderivations of shape:

$$(\Theta^j_k) \qquad \dfrac{\Phi^j_k \vdash_{\mathtt{q}} S_k : D^j_k \qquad (x : [a^j_i] \vdash_{\mathtt{q}} x : a^j_i)_{i \in I^j_k}}{\Phi^j_k \uplus_{i \in I^j_k} \{x : [a^j_i]\} \vdash_{\mathtt{q}} S_k x : F^j_k}$$

where $k \in K^j$ and the sets I^j_k form a partition of indexes, i.e. $I^j_k \cap I^j_{k'} = \emptyset$ if $k \neq k'$ and $\bigcup_{k \in K^j} I^j_k = I^j_{\text{arg}} = (I^j - I^j_{\text{fun}})$; observe that $|K^j| \leq |I^j_{\text{arg}}|$. Indeed, either the type of S_k is an arrow type, let it be $D^j_k = [d^j_i]_{i \in I^j_k} \to F^j_k$, or both $D^j_k = d^j_k$ and $F^j_k = f^j_k$ are type variables. Let $K^j_{\text{arr}} = \{k \in K^j \mid \text{the type of } S_k \text{ is an arrow type}\}$ and $K^j_{\text{var}} = K^j - K^j_{\text{arr}}$; note that $k \in K^j_{\text{var}}$ implies I^j_k is a singleton, hence K^j_{var} and $\bigcup_{k \in K^j_{\text{var}}} I^j_k$ can be identified. The last rule of all subderivations Θ^j_k generate equations:

$$E_{\text{arg}} = \{d^j_i \doteq a^j_i \mid j \in J,\, k \in K^j_{\text{arr}},\, i \in I^j_k\} \cup \{d^j_k \doteq [a^j_k] \to f^j_k \mid j \in J,\, k \in K^j_{\text{var}}\}$$

The sequence s varies depending on whether A^j_i is a type variable or not. In the former case no expansion is needed, i.e. s consists of unification rules only. We now analyse both cases in detail.

A^j_i is a variable. A^j_i is a variable means that Q is not an abstraction. Let $A^j_i = b^j_i$ and $E_M = E' \cup E_{\text{fun}} \cup E_{\text{arg}} \cup \{a^j_i \doteq b^j_i \mid j \in J,\, i \in I^j\}$. Consider the sequence of substitutions $\bar{u}$ replacing each a^j_i by b^j_i. These substitutions modify only the components E_{fun} and E_{arg}, which become:

$$E^*_{\text{fun}} = \{b^j_i \doteq [C^j_i] \to e^j_i \mid j \in J,\, i \in I^j_{\text{fun}}\}$$
$$E^*_{\text{arg}} = \{d^j_i \doteq b^j_i \mid j \in J,\, k \in K^j_{\text{arr}},\, i \in I^j_k\} \cup \{d^j_k \doteq [b^j_k] \to f^j_k \mid j \in J,\, k \in K^j_{\text{var}}\}$$

Observe that $E_N = E' \cup E^*_{\text{fun}} \cup E^*_{\text{arg}} \subset \bar{u}(E_M)$, because in Π_N the subderivations corresponding to Σ^j_i and Θ^j_k are, respectively:

$$(\Sigma'^j_i) \qquad \frac{\Delta^j_i \vdash_{\mathtt{q}} Q : b^j_i \qquad \Psi'^j_i \vdash_{\mathtt{q}} R_i[Q/x] : C^j_i}{\Delta^j_i \uplus \Psi'^j_i \vdash_{\mathtt{q}} Q(R_i[Q/x]) : e^j_i}$$

$$(\Theta'^j_k) \qquad \frac{\Phi'^j_k \vdash_{\mathtt{q}} S_k[Q/x] : D^j_k \qquad (\Delta^j_i \vdash_{\mathtt{q}} Q : b^j_i)_{i \in I^j_k}}{\Phi'^j_k \uplus_{i \in I^j_k} \Delta^j_i \vdash_{\mathtt{q}} (S_k[Q/x])Q : F^j_k}$$

The various a^j_i do not occur in Π_N, thus we also have $V_N = V_M - \{a^j_i \approx a^j_l \mid j \in J,\, i, l \in I^j\}$. We conclude that in this case $s = \bar{u}$, i.e. the sequence consists of the unification rules transforming $E_{\text{fun}} \cup E_{\text{arg}} \cup \{a^j_i \doteq b^j_i \mid j \in J,\, i \in I^j\}$ into $E^*_{\text{fun}} \cup E^*_{\text{arg}} \cup \{a^j_i \doteq b^j_i \mid j \in J,\, i \in I^j\}$. It is straightforward to check that $\bar{u}(E_M) - E_N = \{a^j_i \doteq b^j_i \mid j \in J,\, i \in I^j\}$ is in solved form, and that all variables a^j_i do not occur outside of $\bar{u}(E_M) - E_N$. Hence $\bar{u}(E_M) - E_N$ cannot play any role in generating critical equations.

A^j_i is not a variable. Since Q is in normal form, A^j_i not a variable means that Q is an abstraction; hence the reduction $\mathtt{C}[(\lambda x.P)Q] \to^{\mathtt{in}}_{\beta} \mathtt{C}[P[Q/x]]$ may generate new redexes. Let $A^j_i = \sigma^j_{i,0} \to \ldots \to \sigma^j_{i,q} \to c^j_i$ for some $q \geq 0$, and let $|\sigma^j_{i,p}| = n^j_{i,p}$ $(0 \leq p \leq q)$. For $j \in J\,, i \in I^j_{\text{fun}}$ and $p^j_i \leq q$, let $\sigma^j_{i,0} \ldots, \sigma^j_{i,p^j_i}$ be associated to applied bound variables in Π_N; then the application of unification rules generates one critical equation for each $\sigma^j_{i,p}$ such that $p \leq p^j_i$ and $n^j_{i,p} > 1$. During the while loop, `Solve` can choose exclusively these critical equations and perform the appropriate expansions. For any given $j \in J$, $i \in I^j_{\text{fun}}$, a single expansion $\texttt{Expand}([C^j_i], n^j_{i,0} - 1)$ transforms Σ^j_i into:

$$(\widehat{\Sigma}^j_i) \qquad \frac{x : [a^j_i] \vdash_{\mathtt{q}} x : a^j_i \qquad (\Psi^j_{i,m} \vdash_{\mathtt{q}} R_i : C^j_{i,m})_{1 \leq m \leq n^j_{i,0}}}{\{x : [a^j_i]\} \uplus_{1 \leq m \leq n^j_{i,0}} \Psi^j_{i,m} \vdash_{\mathtt{q}} xR_i : e^j_i}$$

Similarly, with the convention that $e^j_i = e^j_{i,1}$, an expansion for $1 \le p \le p^j_i$ transforms a subderivation like:

$$(\Omega^j_{i,p}) \qquad \frac{\Xi^j_{i,p} \vdash_{\mathtt{q}} U_{i,p} : e^j_{i,p} \qquad \Upsilon^j_{i,p} \vdash_{\mathtt{q}} V_{i,p} : G^j_{i,p}}{\Xi^j_{i,p} \uplus \Upsilon^j_{i,p} \vdash_{\mathtt{q}} U_{i,p}V_{i,p} : e^j_{i,p+1}}$$

into:

$$(\widehat{\Omega}^j_{i,p}) \qquad \frac{\Xi^j_{i,p} \vdash_{\mathtt{q}} U_{i,p} : e^j_{i,p} \qquad (\Upsilon^j_{i,p,m} \vdash_{\mathtt{q}} V_{i,p} : G^j_{i,p,m})_{1\le m\le n^j_{i,p}}}{\Xi^j_{i,p} \uplus_{1\le m\le n^j_{i,p}} \Upsilon^j_{i,p,m} \vdash_{\mathtt{q}} U_{i,p}V_{i,p} : e^j_{i,p+1}}$$

Observe that each espansion may increase the cardinality of the set of indexes I^j: think, for instance, of the term $P = x(xP')(P''x)$ when $Q : [a,a'] \to [b,b'] \to c$. This originates a sequence of expansions $\bar{e}$ and an increasing sequence $I^j_{e1}, I^j_{e2}, I^j_{e3}, \ldots$ of sets of indexes. Let $\widehat{I}^j$ denote the last set of the sequence: such a set must exist because functional occurrences of x may increment, to their right, the number of subderivations with subject x, but not vice versa. Let $\widehat{\Pi}_M = \bar{e}(\Pi_M)$ be derivation obtained after all the aforementioned expansions, and let $(\widehat{E}_M, \widehat{V}_M) = (\bar{e}(E_M), \bar{e}(V_M))$ be its associated constraints (w.l.o.g., we ignore the unification rules performed between expansions: they are not relevant in our proof because the system of constraints is recomputed after each expansion). In the same way as before, for each $j \in J$ it is possible to identify $\widehat{I}^j_{\text{fun}} \subseteq \widehat{I}^j$ and the partition $\{\widehat{I}^j_k \mid k \in \widehat{K}^j\}$ such that $\bigcup_{k\in\widehat{K}} \widehat{I}^j_k = \widehat{I}^j_{\text{arg}} = (\widehat{I}^j - \widehat{I}^j_{\text{fun}})$; with these subsets of indexes, the equations related to all occurrences of x in functional and argument position can be expressed respectively as:

$$\widehat{E}_{\text{fun}} = \{a^j_i \doteq [C^j_{i,0}, \ldots, C^j_{i,n^j_{i,0}}] \to e^j_i \mid j \in J,\, i \in \widehat{I}^j_{\text{fun}}\}$$
$$\widehat{E}_{\text{arg}} = \{d^j_i \doteq A^j_i \mid j \in J,\, k \in \widehat{K}^j_{\text{arr}},\, i \in \widehat{I}^j_k\} \cup \{d^j_k \doteq [A^j_k] \to f^j_k \mid j \in J,\, k \in \widehat{K}^j_{\text{var}}\}$$

Moreover, the equalities generated by the last elimination rule of the various $\widehat{\Omega}^j_{i,p}$ are (recall that $e^j_i = e^j_{i,1}$):

$$\widehat{E}_{\widehat{\Omega}} = \{e^j_{i,p} \doteq [G_{i,p,1}, \ldots, G_{i,p,n^j_{i,p}}] \to e^j_{i,p+1} \mid j \in J,\, i \in \widehat{I}^j_{\text{fun}},\, 1 \le p \le p^j_i\}$$

Now let $\sigma^j_{i,p} = [b^j_{i,p,1}, ..., b^j_{i,p,n^j_{i,p}}]$ $(0 \le p \le p^j_i)$ and observe that $\{a^j_i \doteq A^j_i \mid j \in J,\, i \in \widehat{I}^j\} \subset \widehat{E}_M$. Thanks to the now agreeing multiset cardinalities, the call to $\mathtt{QuasiUnify}_{\doteq}(\widehat{E}_M)$ on line 16 of the algorithm can finally apply a sequence $\bar{u}$ of unification rules that perform all substitutions involving the various $e^j_{i,p}$ and replace a^j_i by A^j_i, then decompose the resulting equations. Such a sequence transforms $\widehat{E}_{\text{fun}}$ and $\widehat{E}_{\text{arg}}$ into:

$$\widehat{E}^*_{\text{fun}} = \{b^j_{i,0,1} \doteq C^j_{i,1}, \ldots, b^j_{i,0,n^j_{i,0}} \doteq C^j_{i,n^j_{i,0}} \mid j \in J,\, i \in \widehat{I}^j_{\text{fun}}\}$$
$$\widehat{E}^*_{\text{arg}} = \{d^j_i \doteq A^j_i \mid j \in J,\, k \in \widehat{K}^j_{\text{arr}},\, i \in \widehat{I}^j_k\} \cup \{d^j_k \doteq [A^j_k] \to f^j_k \mid j \in J,\, k \in \widehat{K}^j_{\text{var}}\}$$

These equations are also part of E_N because, by construction, in Π_N the subderivations corresponding to $\widehat{\Sigma}^j_i$ and $\widehat{\Theta}^j_k$ $(j \in J,\, i \in \widehat{I}^j,\, k \in \widehat{K}^j)$ are, respectively:

$$(\Sigma'^j_i) \qquad \frac{\Delta^j_i \vdash_{\mathtt{q}} Q : \sigma^j_{i,0} \to \ldots \to \sigma^j_{i,q} \to c^j_i \qquad (\Psi'^j_{i,m} \vdash_{\mathtt{q}} R_i[Q/x] : C^j_{i,m})_{1\le m\le n^j_{i,0}}}{\Delta^j_i \uplus_{1\le m\le n^j_{i,0}} \Psi'^j_{i,m} \vdash_{\mathtt{q}} Q(R_i[Q/x]) : \sigma^j_{i,1} \to \ldots \to \sigma^j_{i,q} \to c^j_i}$$

where $\sigma^j_{i,0} = [b^j_{i,0,1}, ..., b^j_{i,0,n^j_{i,0}}]$, and

$$(\Theta'^j_k) \qquad \frac{\Phi'^j_k \vdash_{\mathtt{q}} S_k[Q/x] : D^j_k \qquad (\Delta^j_i \vdash_{\mathtt{q}} Q : A^j_i)_{i \in \widehat{I}^j_k}}{\Phi'^j_k \uplus_{i \in I^j_k} \Delta^j_i \vdash_{\mathtt{q}} (S_k[Q/x])Q : F^j_k}$$

Moreover, focusing on the equations that involve $\sigma^j_{i,p}$ $(1 \leq p \leq p^j_i)$, one can see that applying the multiset decompositions in $\bar{u}$ produces:

$$E_\sigma = \{b_{i,p,1} \doteq G_{i,p,1}, \ldots, b_{i,p,n^j_{i,p}} \doteq G_{i,p,n^j_{i,p}} \mid j \in J,\, i \in \widehat{I}^j_{\text{fun}},\, 1 \leq p \leq p^j_i\}$$

Clearly the above equations are part of E_N too, as Π_N contains subderivations of shape:

$$(\Omega'^j_{i,p}) \quad \frac{\Xi'^j_{i,p} \vdash_{\mathtt{q}} U_{i,p}[Q/x] : \sigma^j_{i,p} \to \ldots \to \sigma^j_{i,q} \to c^j_i \qquad (\Upsilon'^j_{i,p,m} \vdash_{\mathtt{q}} V_{i,p}[Q/x] : G^j_{i,p,m})_{1 \leq m \leq n^j_{i,p}}}{\Xi'^j_{i,p} \uplus_{1 \leq m \leq n^j_{i,p}} \Upsilon'^j_{i,p,m} \vdash_{\mathtt{q}} (U_{i,p}V_{i,p})[Q/x] : \sigma^j_{i,p+1} \to \ldots \to \sigma^j_{i,q} \to c^j_i}$$

Recall that $\widehat{E}_M = \bar{e}(E_M)$ and $\widehat{V}_M = \bar{e}(V_M)$. Hence, ignoring the unification rules performed between expansions, we have $E_N = E' \cup \widehat{E}^*_{\text{fun}} \cup \widehat{E}^*_{\text{arg}} \cup E_\sigma \subset \bar{u}(\widehat{E}_M) = \bar{u} \circ \bar{e}(E_M)$; moreover the various a^j_i do not occur in Π_N, so $V_N = \bar{e}(V_M) - \{a^j_i \approx a^j_l \mid j \in J,\, i, l \in \widehat{I}^j\}$. Summarising, if A^j_i is not a variable the sequence $s = \bar{u} \circ \bar{e}$ consists of a sequence of expansions $\bar{e}$ transforming (E_M, V_M) into $(\widehat{E}_M, \widehat{V}_M)$, followed by the unification rules $\bar{u}$ eventually leading to a superset of E_N. We conclude by observing that $s(E_M) - E_N = \bar{u}(\widehat{E}_{\widehat{\Omega}}) \cup \{a^j_i \doteq A^j_i \mid j \in J,\, i \in I^j\}$ is in solved form, and that all variables a^j_i and $e^j_{i,p}$ do not occur outside of $s(E_M) - E_N$. Thus, once again, equations belonging to $s(E_M) - E_N$ cannot play any role in generating critical equations.

The special case $P = x$. This is the only way x can occur in P, but neither in functional nor argument position. A sequence $s = \bar{u} \circ \bar{e}$ of expansions and unification rules similar to the one described above may be necessary also in this case, because $P[Q/x] = Q$ can generate new redexes in N if Q is an abstraction.

Now consider an innermost strategy, say the rightmost-innermost one; thanks to the fact that $s(E_M) - E_N$ cannot generate critical equations, termination of $\mathtt{Solve}(M)$ (more precisely, of a deterministic version that always performs the expansions associated to the rightmost-innermost redex) follows by induction on the length of the rightmost-innermost reduction from M to its normal form, using Lemma 24. The extension of this result to all possile sequences of expansions can then be obtained by observing that a critical equation always originates from a redex appearing in one of the reduction sequences from M to its normal form. Therefore, taking into account that:

- M is strongly normalizing, so the length of all reduction sequences from M to its normal form is finite;
- a β-reduction corresponds to a finite number ≥ 0 of expansions on $\mathtt{PD_i}(M)$;
- different critical equations are generated in different subtrees of $\mathtt{PD_i}(M)$, hence the order in which expansions are performed does not matter;

we can conclude that the number of expansions performed by the algorithm is always finite, i.e. $\mathtt{Solve}(M)$ terminates. ◀

Knowledge Problems in Security Protocols: Going Beyond Subterm Convergent Theories

Saraid Dwyer Satterfield
University of Mary Washington, Fredericksburg, VA, USA

Serdar Erbatur
University of Texas at Dallas, TX, USA

Andrew M. Marshall
University of Mary Washington, Fredericksburg, VA, USA

Christophe Ringeissen
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Abstract

We introduce a new form of restricted term rewrite system, the graph-embedded term rewrite system. These systems, and thus the name, are inspired by the graph minor relation and are more flexible extensions of the well-known homeomorphic-embedded property of term rewrite systems. As a motivating application area, we consider the symbolic analysis of security protocols, and more precisely the two knowledge problems defined by the deduction problem and the static equivalence problem. In this field restricted term rewrite systems, such as subterm convergent ones, have proven useful since the knowledge problems are decidable for such systems. However, many of the same decision procedures still work for examples of systems which are "beyond subterm convergent". However, the applicability of the corresponding decision procedures to these examples must often be proven on an individual basis. This is due to the problem that they don't fit into an existing syntactic definition for which the procedures are known to work. Here we show that many of these systems belong to a particular subclass of graph-embedded convergent systems, called contracting convergent systems. On the one hand, we show that the knowledge problems are decidable for the subclass of contracting convergent systems. On the other hand, we show that the knowledge problems are undecidable for the class of graph-embedded systems.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Theory of computation → Automated reasoning

Keywords and phrases Term rewriting, security protocols, verification

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.30

Funding *Christophe Ringeissen*: This work has been partly supported by the ANR Research and teaching chair in AI ASAP (ANR-20-CHIA-0024) and the Région Grand Est.

Acknowledgements We would like to thank Steve Kremer for his comments on the paper as they were very helpful in improving several technical results.

1 Introduction

In this paper we introduce a new form of term rewrite system, called the graph-embedded term rewrite systems, and motivate the study and use of such rewrite systems by demonstrating their usefulness in the application of security protocols.

The research area of cryptographic protocol analysis contains a number of innovative algorithms and procedures for checking various security properties of protocols, see for example [1, 12, 15, 17]. These procedures consider protocols modeled in a symbolic way, typically via a rewrite system or equational theory. Often the procedure is proven sound

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 30; pp. 30:1–30:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and complete for specific classes of theories. One of the most common classes are those theories that can be represented by subterm convergent term rewrite systems. That is, term rewrite systems where the right-hand side of the rules are ground or strict subterms of the left-hand side. For example, see the procedures developed in [1, 15]. Interestingly, many of these same procedures also work for theories that are "beyond subterm convergent". That is, they are not strictly subterm convergent. However, since these examples don't fit into a known class of theories for which soundness and completeness proofs already exist, they must be proven on an individual bases. For example, the procedures of [1, 12, 15, 17] are shown to work on the theory of blind signatures, see Example 2 below. However, the theory is not subterm convergent, notice in the final rule, $\mathit{unblind}(\mathit{sign}(\mathit{blind}(x, y), z), y) \to \mathit{sign}(x, z)$, that $\mathit{sign}(x, z)$ is not a strict subterm of $\mathit{unblind}(\mathit{sign}(\mathit{blind}(x, y), z), y)$. Thus, in each case a unique proof is needed to show applicability of the procedure on the theory of blind signatures. Several additional examples of beyond subterm theories are given throughout the paper. This begs the question of whether there is a syntactic definition of a class of term rewrite systems such that the definition encapsulates these beyond subterm examples yet still maintains some of the useful properties needed to ensure applicability of the above procedures.

In the paper we answer the question in the positive by introducing first graph-embedded term rewrite systems and then a particular subclass called contracting rewrite systems. These systems are inspired by the notions of graph embeddings and graph minors. Here we are able to translate the notion to term rewrite systems. This translation is done in a very similar fashion to what has been done with homeomorphic embeddings. We are able to provide a rewrite schema which induces graph-embedded systems in a similar way in which homeomorphic-embedded systems are induced by a rewrite system (see [7] for more details). To the best of our knowledge these systems have not been explored before. We then explore some of the properties of these new systems. Interestingly, the graph-embedded systems encompass most of the beyond subterm examples from many of the protocol analysis procedures [1, 12, 15, 17]. As an initial step, in this paper we concentrate on the knowledge problems considered in [1] using the notion of locally stable theories. Local stability is a desirable property which ensures the decidability of the critical symbolic security question of deducibility. In the class of graph-embedded convergent systems, we are now able to identify a particular subclass called the contracting convergent systems, which are beyond subterm convergent, encompass most of the beyond subterm examples of [1, 12, 15, 17], and are locally stable. As a consequence, the knowledge problems of deduction and static equivalence are decidable for the subclass of contracting convergent systems. Furthermore, we show that the knowledge problems are undecidable for the class of graph-embedded convergent systems in general.

Finally, this paper represents the initial exploration of graph-embedded term rewrite systems and their application to protocol analysis. We hope that the formulation proves useful in areas beyond security protocols as homeomorphic embeddings have proven useful in many areas. We conclude the paper with a discussion of several open questions related to graph-embedded systems.

Paper Outline. The remainder of the paper is organized as follows. Section 2 contains the preliminaries, introducing the necessary background material on term-rewrite systems, graph theory and security protocol analysis. Section 3 introduces the graph-embedded term rewrite systems and explores some of their basic properties. Section 4 introduces the motivating application area of this paper for graph-embedded systems, security protocol analysis. In that section, we show that the knowledge problems are undecidable for the class of graph-embedded convergent systems but decidable for the subclass of contracting convergent systems. Section 5 considers the relation to another common and useful property, the Finite Variant Property (FVP). Finally, Section 6 contains the concluding remarks, future work, and some open problems. Our decidability result relies on lemmas that are proven in Appendix A.

2 Preliminaries

We use the standard notation of equational unification [8] and term rewriting systems [7]. Given a first-order signature Σ and a (countable) set of variables V, the Σ-terms over variables V are built in the usual way by taking into account the arity of each function symbol in Σ. Each Σ-term is well-formed: if it is rooted by a n-ary function symbol in Σ, then it has necessarily n direct subterms. The set of Σ-terms over variables V is denoted by $T(\Sigma, V)$. Given a (countable) set of constants C disjoint from V and Σ, the set of Σ-terms over $V \cup C$ is denoted in the same way by $T(\Sigma, V \cup C)$. In the following, a Σ-term is assumed to be a term in $T(\Sigma, V \cup C)$. The set of variables (resp., constants) from V (resp., C) occurring in a term $t \in T(\Sigma, V \cup C)$ is denoted by $\mathit{Var}(t)$ (resp., $\mathit{Cst}(t)$). A term t is *ground* if $\mathit{Var}(t) = \emptyset$. A $\Sigma \cup C$-rooted term is a term whose root symbol is in $\Sigma \cup C$. For any position p in a term t (including the root position ϵ), $t(p)$ is the symbol at position p, $t|_p$ is the subterm of t at position p, and $t[u]_p$ is the term t in which $t|_p$ is replaced by u. A substitution is an endomorphism of $T(\Sigma, V \cup C)$ with only finitely many variables not mapped to themselves. A substitution is denoted by $\sigma = \{x_1 \mapsto t_1, \ldots, x_m \mapsto t_m\}$, where the domain of σ is $\mathit{Dom}(\sigma) = \{x_1, \ldots, x_m\}$ and the range of σ is $\mathit{Ran}(\sigma) = \{t_1, \ldots, t_m\}$. Application of a substitution σ to t is written $t\sigma$.

The size of a term t, denoted by $|t|$, is defined inductively as follows: $|f(t_1, \ldots, t_n)| = 1 + \Sigma_{i=1}^n |t_i|$ if f is a n-ary function symbol with $n \geq 1$, $|c| = 1$ if c is a constant, and $|x| = 1$ if x is a variable. The depth of a term t, denoted by $\mathit{depth}(t)$, is defined inductively as follows: $\mathit{depth}(f(t_1, \ldots, t_n)) = 1 + \max_{i=1,\ldots,n} \mathit{depth}(t_i)$ if f a n-ary function symbol with $n \geq 1$, $\mathit{depth}(c) = 0$ if c is a constant, and $\mathit{depth}(x) = 0$ if x is a variable.

A *context* is a term with holes. More formally, a context is a term where each variable occurs at most once. Thus, the size of a context follows from the size of a term, where any hole occurrence counts for 1.

Equational Theories

Given a set E of Σ-axioms (i.e., pairs of terms in $T(\Sigma, V)$, denoted by $l = r$), the *equational theory* $=_E$ is the congruence closure of E under the law of substitutivity (by a slight abuse of terminology, E is often called an equational theory). Equivalently, $=_E$ can be defined as the reflexive transitive closure $\leftrightarrow_E^*$ of an equational step $\leftrightarrow_E$ defined as follows: $s \leftrightarrow_E t$ if there exist a position p of s, $l = r$ (or $r = l$) in E, and substitution σ such that $s|_p = l\sigma$ and $t = s[r\sigma]_p$.

Rewrite Relations

A *term rewrite system* (TRS) is a pair (Σ, R), where Σ is a signature and R is a finite set of rewrite rules of the form $l \to r$ such that l, r are Σ-terms, l is not a variable and $\mathit{Var}(r) \subseteq \mathit{Var}(l)$. A term s *rewrites* to a term t w.r.t R, denoted by $s \to_R t$ (or simply $s \to t$), if there exist a position p of s, $l \to r \in R$, and substitution σ such that $s|_p = l\sigma$ and $t = s[r\sigma]_p$. If the rewrite step occurs at the root position of the term s we denote this as $s \rightarrow^{\epsilon}_{R} t$. When σ is a variable renaming, we say that s rewrites to t applying a *variable instance* of $l \rightarrow r$. A TRS R is *terminating* if there are no infinite reduction sequences with respect to $\rightarrow_R$. A TRS R is *confluent* if, whenever $t \rightarrow^*_R s_1$ and $t \rightarrow^*_R s_2$, there exists a term w such that $s_1 \rightarrow^*_R w$ and $s_2 \rightarrow^*_R w$. A confluent and terminating TRS is called *convergent.* In a convergent TRS R, we have the existence and the uniqueness of R-normal forms, denoted by $t{\downarrow_R}$ for any term t. When R is clear from the context, the normal form of t may be written $t{\downarrow}$. Given a substitution σ, $\sigma{\downarrow} = \{x \mapsto (x\sigma){\downarrow}\}_{x \in Dom(\sigma)}$ is the substitution corresponding to the normal form of σ.

A convergent *term rewrite system* (TRS) R is said to be *subterm convergent* if for any $l \rightarrow r \in R$, r is either a strict subterm of l or a ground term. An equational theory, E, is *subterm convergent* if it is presented by a subterm convergent TRS. That is, there exists a subterm convergent TRS, R, such that $=_E$ and $=_R$ coincide.

▶ **Definition 1** (Homeomorphic Embedding). *The* homeomorphic embedding, *$\trianglerighteq_{emb}$ is a binary relation on terms such that: $s \trianglerighteq_{emb} t$ if one of the following conditions hold:*

1. *$s = x = t$ for some variable x,*
2. *$s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ and $s_1 \trianglerighteq_{emb} t_1, \ldots, s_n \trianglerighteq_{emb} t_n$,*
3. *$s = f(s_1, \ldots, s_n)$ and $s_i \trianglerighteq_{emb} t$ for some i, $1 \leq i \leq n$.*

A TRS R is said to be a homeomorphic-embedded *TRS if for any $l \rightarrow r \in R$, $l \trianglerighteq_{emb} r$.*

More interestingly we can also define $\trianglerighteq_{emb}$ as the reduction relation $\rightarrow^*_{R_{emb}}$ induced by the rewrite system $R_{emb} = \{f(x_1, \ldots, x_n) \rightarrow x_i \mid f \text{ is } n\text{-ary}, n \geq 1,\ 1 \leq i \leq n\}$.

▶ **Example 2** (Blind Signatures). The theory of blind signatures [15] is a homeomorphic-embedded convergent TRS:

$$\begin{aligned} checksign(sign(x, y), pk(y)) &\rightarrow x \\ unblind(blind(x, y), y) &\rightarrow x \\ unblind(sign(blind(x, y), z), y) &\rightarrow sign(x, z) \end{aligned}$$

Notions of Knowledge

The applied pi calculus and frames are used to model attacker knowledge [2]. In this model, the set of messages or terms which the attacker knows, and which could have been obtained from observing one or more protocol sessions, are the set of terms in $Ran(\sigma)$ of the frame $\phi = \nu\tilde{n}.\sigma$, where σ is a substitution ranging over ground terms. We also need to model cryptographic concepts such as nonces, keys, and publicly known values. We do this by using names, which are essentially free constants. Here also, we need to track the names which the attacker knows, such as public values, and the names which the attacker does not know a priori, such as freshly generated nonces. $\tilde{n}$ consists of a finite set of restricted names, these names represent freshly generated names which remain secret from the attacker. The set of names occurring in a term t is denoted by $fn(t)$. For any frame $\phi = \nu\tilde{n}.\sigma$, let $fn(\phi)$ be the set of names $fn(\sigma)\backslash\tilde{n}$ where $fn(\sigma) = \bigcup_{t \in Ran(\sigma)} fn(t)$; and for any term t, let $t\phi$ denote by a slight abuse of notation the term $t\sigma$. We say that a term t *satisfies the name restriction (of ϕ)* if $fn(t) \cap \tilde{n} = \emptyset$.

▶ **Definition 3** (Deduction). *Let $\phi = \nu\tilde{n}.\sigma$ be a frame, and t a ground term. We say that t is* deduced from ϕ modulo E, *denoted by $\phi \vdash_E t$, if there exists a term ζ such that $\zeta\sigma =_E t$ and $fn(\zeta) \cap \tilde{n} = \emptyset$. The term ζ is called a* recipe *of t in ϕ modulo E.*

Another form of knowledge is the ability to tell if two frames are *statically equivalent* modulo E, sometimes also called *indistinguishability.*

▶ **Definition 4** (Static Equivalence)**.** *Two terms s and t are* equal *in a frame $\phi = \nu\tilde{n}.\sigma$ modulo an equational theory E, denoted $(s =_E t)\phi$, if $s\sigma =_E t\sigma$, and $\tilde{n} \cap (fn(s) \cup fn(t)) = \emptyset$. The set of all equalities $s = t$ such that $(s =_E t)\phi$ is denoted by $Eq(\phi)$. Given a set of equalities Eq, the fact that $(s =_E t)\phi$ for any $s = t \in Eq$ is denoted by $\phi \models Eq$. Two frames $\phi = \nu\tilde{n}.\sigma$ and $\psi = \nu\tilde{n}.\tau$ are* statically equivalent *modulo E, denoted as $\phi \approx_E \psi$, if $Dom(\sigma) = Dom(\tau)$, $\phi \models Eq(\psi)$ and $\psi \models Eq(\phi)$.*

Both deduction and static equivalence are known to be decidable in subterm convergent rewrite systems [1]. In this paper, we lift these results to rewrite systems that are beyond the class of subterm convergent rewrite systems.

▶ **Example 5.** Let E be the equational theory presented by the subterm convergent TRS $\{dec(enc(x, y), y) \to x\}$. Applying the decision procedure developed in [1], one can check that $\phi = \nu\{n\}.\{v \mapsto enc(a, n)\}$ and $\psi = \nu\{n\}.\{v \mapsto enc(b, n)\}$ are statically equivalent modulo E. Consider now $\phi' = \nu\{n\}.\{v \mapsto enc(a, n), w \mapsto n\}$ and $\psi' = \nu\{n\}.\{v \mapsto enc(b, n), w \mapsto n\}$. Since $dec(v, w) = a \in Eq(\phi')$ and $dec(v, w) = a \notin Eq(\psi')$, ϕ' and ψ' are not statically equivalent modulo E.

Term Graphs

Each term t can be viewed in a graphical representation, called a *term graph*. Each node in the graph is labeled either by a function symbol or a variable. Each function symbol node also has an associated successor number, corresponding to the arity of the function. Edges connect the nodes of the term graph based on the subterm relation. The notion of term graph is illustrated in Examples 19 and 20.

▶ **Definition 6** (Term Graph Measures)**.** *We introduce some convenient notation:*

- *Let $VP(t)$ denote the list of leaf nodes in the term graph of a term t labeled by a variable. Notice that two distinct nodes could be labeled by the same variable.*
- *Let $FP(t)$ denote the list of nodes in the term graph of t labeled by a function symbol. Notice that two distinct nodes could be labeled by the same function symbol.*
- *Let $FS(t)$ denote the set of function symbols in the term t.*

Some Graph Theory

We will also need a few notions from graph theory, we introduce those in this section. We will typically use G to denote a graph, V the set of vertex and E the set of edges of the graph.

▶ **Definition 7** (Graph Isomorphism)**.** *Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. We say that G and G' are isomorphic, denoted $G \simeq G'$, if there exists a bijection $\phi : V \to V'$ with $xy \in E$ iff $\phi(x)\phi(y) \in E'$, $\forall x, y \in V$.*

▶ **Definition 8** (Edge Contraction)**.** *Let $G = (V, E)$ and $e = xy$. G/e is the graph $G' = (V', E')$ such that $V' = (V \setminus \{x, y\}) \cup \{v_e\}$, where v_e is a new vertex, and $E' = \{vw \in E \mid \{v, w\} \cap \{x, y\} = \emptyset\} \cup \{v_e w \mid xw \in E \setminus \{e\}$ or $yw \in E \setminus \{e\}\}$.*

We say that G' is obtained from G by contracting *the edge e.*

We use the following definition of graph minor which essentially says that a graph minor of a graph G can be obtained by a series of graph contractions (see [16] for more details).

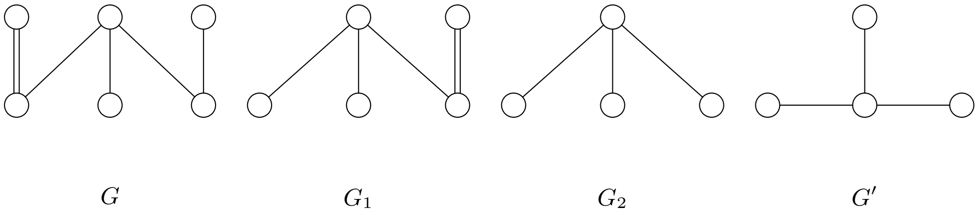
▶ **Definition 9** (Graph Minor). *The graph G' is a* graph minor *of the graph G, if there exist graphs G_0, G_1, $\ldots$, G_n and edges $e_i \in G_i$ such that $G = G_0$, $G_n \simeq G'$, and $G_{i+1} = G_i/e_i$ for $i = 0, \ldots, n-1$. We use the notation $G \succcurlyeq G'$ if G' is a graph minor of G.*

Further, we extend the pure graph definition above to terms using the same notion as follows. For terms t and t', $t \succcurlyeq t'$ if for the corresponding term graphs of t and t', denoted as G and G' respectively, we have $G \succcurlyeq G'$.

▶ Remark 10. Note, in the classical definition of graph minor (see [16]), if G is a subgraph of a larger graph G_{large}, then also $G_{large} \succcurlyeq G'$. However, this component of the definition is not necessary for the results of this paper and by leaving it out we are able to simplify the later definitions and presentation.

The above type of embedding, denoted by $\succcurlyeq$, provides more flexibility than the traditional subterm relation while still preserving some features we need.

▶ **Example 11.** Notice that G' is obtained from G by first applying a sequence of edge contractions, contracting the edge depicted by $||$ at each step, resulting in G_2, and finally $G_2 \simeq G'$. Therefore, $G \succcurlyeq G'$.



We can now extend the above graph-theoretic notions to the term rewrite setting.

3 Graph-Embedded Systems

The key to translating from the graph theory setting to the term setting is to use the same methods, *contractions*, but require that the *final* term graph constructed in this fashion represent a *well-formed* term. That is, we need to enforce the notion of a well formed term.

To begin we need to model the graph isomorphism. A restricted form of isomorphism can be translated into the term rewriting setting by considering permutations.

▶ **Definition 12** (Leaf and Subterm Permutations). *We define two types of permutations, $\approx_s$ and $\approx_l$:*

1. *For terms t and t', we say t is* subterm permutatively *equal to t', denoted $t \approx_s t'$, if one of the following is true:*
   a. *$t = t'$, where t and t' are constants or variables, or*
   b. *$t = f(u_1, \ldots, u_n)$ and $t' = f(u_{\sigma(1)}, \ldots, u_{\sigma(n)})$ where f is a n-ary function symbol, $n \geq 1$, and σ is a permutation of the indexes $(1, \ldots, n)$.*
2. *For terms t and t', we say t is* leaf permutatively *equal to t', denoted $t \approx_l t'$, if $t' = t\sigma$ and σ is the unique endomorphism of $T(\Sigma \cup V \cup C)$ such that its restriction to $Var(t) \cup Cst(t)$ is a permutation on $Var(t) \cup Cst(t)$ and its restriction to $(V \cup C) \backslash (Var(t) \cup Cst(t))$ is the identity.*

The first type of permutation, $\approx_s$, allows for permutation inside the term but preserves the layer like structure of the function symbols in the term graph. The second type of permutation in the classical leaf permutability and is restricted to the leaf nodes, i.e., just the variables and constants of the term graph. We will use a combination of the above two permutations in the definition employed for graph-embedded TRS.

▶ **Definition 13** (Permutatively Equal). *For terms t and t', we say t is* permutatively *equal to t', denoted $t \approx t'$, if $t \approx_s t'' \approx_l t'$, for some term t''.*

▶ Remark 14. It is useful here to remark on the motivation of the above definition, $\approx$. The goal is to model the graph isomorphism property. At the same time one needs to be careful not to be too broad and remove layer preserving properties of Definition 15 and thus later protocol properties such as local stability (see Definition 46). In addition, one cannot be too restricted and disallow working protocol representations such as Example 22 which requires more than just leaf permutability. However, it may be possible to improve upon the above definition and allow for additional systems while still maintaining the decidability of the knowledge problems shown here, see the discussion in Section 6.

The next step is to develop a set of rewrite *schema* which preserve a type of graph minor relation on the term graphs. This set of schema then induces a graph-embedded term rewrite system. Notice that this is very similar to what is often done when considering the homeomorphic embeddings, see Definition 1.

▶ **Definition 15** (Graph Embedding). *Consider the following reduction relation, $\rightarrow^*_{R_{gemb}}$, where R_{gemb} is the set of rewrite rules given by the instantiation of the following rule schema:*

$$\left\{\begin{array}{ll} & \textit{for any } f \in \Sigma \\ (1) & f(x_1, \ldots, x_n) \rightarrow x_i \\ (2) & f(x_1, \ldots,\ x_{i-1},\ x_i,\ x_{i+1}\ \ldots,\ x_n) \rightarrow f(x_1, \ldots,\ x_{i-1},\ x_{i+1},\ \ldots,\ x_n) \\ & \textit{and for any } f, g \in \Sigma \\ (3) & f(x_1, \ldots, x_{i-1}, g(\bar{z}), x_{i+1},\ \ldots, x_m) \rightarrow g(x_1, \ldots,\ x_{i-1}, \bar{z}, x_{i+1},\ \ldots, x_m) \\ (4) & f(x_1, \ldots, x_{i-1}, g(\bar{z}), x_{i+1},\ \ldots, x_m) \rightarrow f(x_1, \ldots,\ x_{i-1}, \bar{z}, x_{i+1},\ \ldots, x_m) \end{array}\right\}$$

We say a term t' is graph-embedded *in a term t, denoted $t \succcurlyeq_{gemb} t'$, if t' is a well formed term and there exists a term s such that $t \rightarrow^*_{R_{gemb}} s \approx t'$.*

A TRS R is graph-embedded *if for any $l \rightarrow r \in R$, $l \succcurlyeq_{gemb} r$ or r is a constant.*

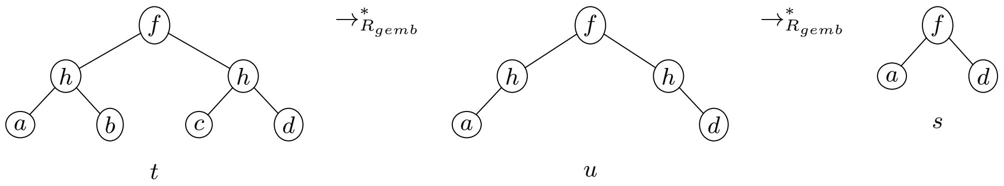
▶ Remark 16. Notice that the rules in R_{gemb} ignore function arity, thus intermediate terms between t and t' may not be well formed. It is only the final term for which function arity and the relation between variables and functions must obey the standard term definition requirements. Note also that, like homeomorphic embedding, the particular schema rules themselves allow for the rewriting of terms down to a single variable. However, the schema are being used to establish the graph-embedded property of a TRS with non-trivial normal forms.

▶ Remark 17. The rules of Definition 15 provide a convenient schema for defining graph-embedded systems. However, they are also very useful in proving properties about graph-embedded systems. Notice that any rewrite step in a graph-embedded system corresponds to one or more steps of the above rules, thus proofs about graph-embedded TRSs can often be reduced to arguments on the properties of the rules of Definition 15.

Definition 15 provides a rewrite relation interpretation of graph-embedded systems which is contained in the $\succcurlyeq$ relation given in Definition 9.

▶ **Lemma 18.** *For any terms t and t', $t \succcurlyeq_{gemb} t'$ implies $t \succcurlyeq t'$, i.e., $\succcurlyeq_{gemb} \subseteq \succcurlyeq$.*

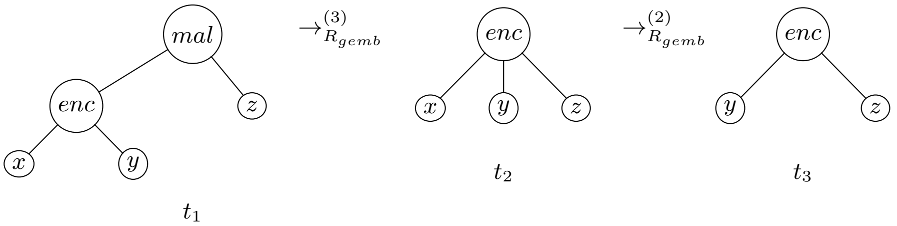
▶ **Example 19.** Consider the two terms $t = f(h(a, b), h(c, d))$ and $t' = f(d, a)$. Then, $t \succcurlyeq_{gemb} t'$, since $t \rightarrow^*_{R_{gemb}} s \approx t'$ where the derivation $t \rightarrow^*_{R_{gemb}} s$ is as follows:



▶ **Example 20** (Malleable Encryption)**.** Consider the theory of Malleable Encryption, R_{mal}:

$$dec(enc(x, y), y) \to x$$
$$mal(enc(x, y), z) \to enc(z, y)$$

For the second rule, let $t_1 = mal(enc(x, y), z)$ and the following derivation $t_1 \rightarrow^*_{R_{gemb}} t_3$:


mal
enc
z
x
y
t_1
$\rightarrow^{(3)}_{R_{gemb}}$
enc
x
y
z
t_2
$\rightarrow^{(2)}_{R_{gemb}}$
enc
y
z
t_3

Since $t_3 \approx enc(z, y)$, we have $mal(enc(x, y), z) \succcurlyeq_{gemb} enc(z, y)$. The first rule of R_{mal} being subterm, $dec(enc(x, y), y) \succcurlyeq_{gemb} x$. Thus, R_{mal} is a graph-embedded TRS.

▶ **Example 21.** The theory of blind signatures from Example 2 is also a graph-embedded TRS. All but the final rule are subterm. For the final rule,

$$unblind(sign(blind(x, y), z), y) \rightarrow_{R_{gemb}} sign(blind(x, y), z)$$

via rule (1). Then,

$$sign(blind(x, y), z) \rightarrow_{R_{gemb}} sign(x, y, z)$$

via rule (4). Notice again that this intermediate term is not well formed. Finally

$$sign(x, y, z) \rightarrow_{R_{gemb}} sign(x, z) \approx sign(x, z)$$

via rule (2).

▶ **Example 22** (Addition)**.** Consider the theory of Addition, R_{add}, from [1]:

$$plus(x, s(y)) \to plus(s(x), y)$$
$$plus(x, 0) \to x$$
$$pred(s(x)) \to x$$

R_{add} is a graph-embedded TRS. Notice that $plus(x, s(y)) \approx plus(s(x), y)$.

▶ **Example 23** (Prefix with Pairing)**.** The theory of prefix with pairing [14, 17] is a graph-embedded TRS:

$$dec(enc(x, y), y) \to x$$
$$prefix(enc(< x, y >, z)) \to enc(x, z)$$
$$fst(< x, y >) \to x$$
$$snd(< x, y >) \to y$$

▶ **Example 24** (Trap-door Commitment)**.** The theory of trap-door commitment [15] is a graph-embedded TRS:

$$\begin{aligned} \mathit{open}(\mathit{td}(x,y,z),y) &\to x \\ \mathit{open}(\mathit{td}(x_1,y,z),f(x_1,y,z,x_2)) &\to x_2 \\ \mathit{td}(x_2,f(x_1,y,z,x_2),z) &\to \mathit{td}(x_1,y,z) \\ f(x_2,f(x_1,y,z,x_2),z,x_3) &\to f(x_1,y,z,x_3) \end{aligned}$$

▶ **Example 25** (Strong Secrecy)**.** Subterm convergent theories where the right hand side can be a subterm or a constant are graph-embedded such as the following system for considering a form of strong secrecy [10, 12]:

$$\begin{aligned} \mathit{fst}(<x,y>) &\to x \\ \mathit{snd}(<x,y>) &\to y \\ \mathit{adec}(\mathit{aenc}(x,\mathit{pk}(y)),y) &\to x \\ \mathit{dec}(\mathit{enc}(x,y),y) &\to x \\ \mathit{check}(\mathit{sign}(x,y),\mathit{pk}(y)) &\to \mathit{ok} \\ \mathit{msg}(\mathit{sign}(x,y)) &\to x \end{aligned}$$

3.1 Some Properties of Graph-Embedded Systems

As an initial step we explore some of the basic properties of the graph-embedded TRSs. Similar to the class of subterm TRSs, the graph-embedded TRSs have several nice properties such as termination.

We can first note that the $\succcurlyeq_{gemb}$ relation is a partial order on the class of terms. This follows from Lemma 18 and the fact that the graph-embedded relation is a partial ordering on the class of finite graphs (See Proposition 1.7.3 from [16]).

In addition, rewriting at the root position preserves the graph-embedded property. This is due to the fact that for any graph-embedded TRS R and for any $l \to r \in R$, $l \succcurlyeq_{gemb} r$. Thus, $l\sigma = t_1 \succcurlyeq_{gemb} t_2 = r\sigma$. More formally, if t_1 and t_2 are terms, R a graph-embedded TRS, and $t_1 \to_R^{\epsilon} t_2$, then, $t_1 \succcurlyeq_{gemb} t_2$.

Graph-embedded systems also have the nice property of being size reducing when rewrite steps are applied and thus terminating.

▶ **Lemma 26.** *Let R be a graph-embedded TRS such that for all $l \to r \in R$, $l \to_{R_{gemb}}^{+} \cdot \approx r$. Assume $t \to_R t'$. Then,*

- $|\mathit{Var}(t')| \leq |\mathit{Var}(t)|$,
- $|\mathit{VP}(t')| \leq |\mathit{VP}(t)|$,
- $\mathit{FS}(t') \subseteq \mathit{FS}(t)$,
- $|\mathit{FP}(t')| < |\mathit{FP}(t)|$.

Proof. No rule from Definition 15 introduces additional function symbols or variables, satisfying the first and third condition. All rules from Definition 15 remove function symbols except the second rule and $\approx$. Notice that if rule 2 is applied then one of the other rules must also be applied to ensure the final term is well formed. Finally, since we require that at least one rewrite step is applied, the size of the term will be reduced even if $\approx$ doesn't reduce the size of the term. Thus the remaining conditions are satisfied. ◀

▶ Remark 27. Notice that if only $\approx$ steps are applied in a graph-embedded system then termination is not guaranteed. However, if at least one rewrite rule from R_{gemb} is applied then by Lemma 26, the system will be terminating.

Comparing Definitions

We can compare the two embedded definitions. Consider Malleable Encryption, R_{mal}, from Example 20. R_{mal} is a graph-embedded TRS, as is shown in Example 20. However, R_{mal} is not a homeomorphic-embedded TRS. This can be seen in the rule $mal(enc(x, y), z) \to enc(z, y)$. There is no way to obtain the term $enc(z, y)$ from the term $mal(enc(x, y), z)$ by application of only the projection rule, $f(x_1, \ldots, x_n) \to x_i$. Thus, it's easy to see that there exist graph-embedded TRSs which are not homeomorphic-embedded TRSs. Furthermore, we see that homeomorphic-embedded TRSs are a subset of graph-embedded TRSs.

▶ **Example 28.** Consider the theory of trap-door commitment from Example 24. Notice that this theory is not a homeomorphic-embedded TRS. For the final rule,

$$f(x_2, f(x_1, y, z, x_2), z, x_3) \to f(x_1, y, z, x_3),$$

we cannot obtain the right-hand side from the left by the simple projection type relation of Definition 1.

4 Knowledge Problems in Graph-Embedded Systems

In this section we look at how graph-embedded TRSs can be used to both extend results in security protocols and also give a formal syntactic definition to classes of protocol presentations for which the decidability of the two knowledge problems are already known. In this section we focus on theories with the local stability property as introduced in [1] (and extended in [6]). For this purpose, we need to consider a restricted form of graph-embedded system called contracting system introduced in Definition 33. One can show that without such a restriction, the knowledge problems for graph-embedded TRSs are undecidable in general.

4.1 Undecidable Knowledge Problems

It is shown in [1] that the knowledge problems are undecidable in general. For graph-embedded systems we can reuse, with modification, a proof that was developed in [3] for the unification problem in Δ-strong convergent theories. A very similar proof can be found in the research report [4] of the paper [5]. The proof in [3] is via a reduction from the Modified Post Correspondence Problem (MPCP). There, MPCP is needed to ensure solutions, such as $\sigma = \{x \mapsto c, y \mapsto c\}$, which don't actually solve an instance of the problem are not possible. We use a similar approach but reduce from the standard Post Correspondence Problem (PCP), using a similar rewrite system to that developed in [3]. We use the standard PCP instead of the MPCP since the encoding works better for the deduction problem where you are finding recipe terms not substitutions. This modification also requires adapting the reduction to the use of frames.

Let $\Gamma = \{a, b\}$ be the alphabet of the PCP problem. Then, an instance of the problem is a finite set of string pairs, $S = \{(\alpha_i, \beta_i) | \ i \in [1, n]\} \subseteq \Gamma^+ \times \Gamma^+$. A solution is a sequence of indexes $i_1, \ldots i_k \in [1, n]$ such that $\alpha_{i_1}\alpha_{i_2} \ldots \alpha_{i_k} = \beta_{i_1}\beta_{i_2} \ldots \beta_{i_k}$.

▶ **Lemma 29.** *The deduction problem is undecidable for the class of homeomorphic-embedded convergent TRSs.*

Proof. Let $PCP = \{(\alpha_i, \beta_i) |\ i \in [1, n]\}$ over the alphabet $\Gamma = \{a, b\}$. Consider unary function symbols a_1, b_1, a_2, b_2, and g_i for each $i \in [1, n]$. Let f be a ternary function symbol and let c be a constant. Each string from PCP can be viewed as a sequence of applications of the unary function symbols. Let $\gamma \in \Gamma$ and for each pair (α_i, β_i), $\widetilde{\alpha_i}(x) = \widetilde{\gamma\alpha'_i}(x) = \gamma_1(\widetilde{\alpha'_i}(x))$, and $\widetilde{\beta_i}(x) = \widetilde{\gamma\beta'_i}(x) = \gamma_2(\widetilde{\beta'_i}(x))$. Construct a TRS R as follows: Let $R = \bigcup_{i=1}^{n}\{f(\widetilde{\alpha_i}(x), g_i(y), \widetilde{\beta_i}(z)) \to f(x, y, z)\}$. The g_i ensure there are no critical pairs between rules and thus we have a convergent TRS. Consider the frame $\phi = \nu\tilde{n}.\sigma$ with $\tilde{n} = \{c\}$ and $\sigma = \bigcup_{i=1}^{n}\{x_i \mapsto \widetilde{\alpha_i}(c), y_i \mapsto \widetilde{\beta_i}(c), z_i \mapsto g_i(c)\}$

Finally, let the target ground term be $f(c, c, c)$. Notice that if there is a solution to the PCP then there exists a recipe term ζ such that $\zeta\sigma =_R f(c, c, c)$. Furthermore, if there is a recipe term ζ such that $\zeta\sigma =_R f(c, c, c)$ then a solution to the PCP can be extracted from the indexes of the g_i function symbol in the term $\zeta\sigma$. Finally, the recipe cannot just be $f(c, c, c)$ since $c \in \tilde{n}$. ◀

▶ **Example 30.** Consider the following PCP:

$$\overbrace{\left(\frac{ba}{baa}\right)}^{\text{pair 1}}, \overbrace{\left(\frac{ab}{ba}\right)}^{\text{pair 2}}, \overbrace{\left(\frac{aaa}{aa}\right)}^{\text{pair 3}}$$

Following the construction of Lemma 29:

$$R = \left\{ \begin{array}{ll} f(b_1(a_1(x)), g_1(y), b_2(a_2(a_2(z)))) & \to f(x, y, z) \\ f(a_1(b_1(x)), g_2(y), b_2(a_2(z))) & \to f(x, y, z) \\ f(a_1(a_1(a_1(x))), g_3(y), a_2(a_2(z))) & \to f(x, y, z) \end{array} \right\}$$

And we construct the frame $\phi = \nu\tilde{c}.\{x_1 \mapsto b_1(a_1(c)), y_1 \mapsto b_2(a_2(a_2(c))), z_1 \mapsto g_1(c), x_2 \mapsto a_1(b_1(c)), y_2 \mapsto b_2(a_2(c)), z_2 \mapsto g_2(c), x_3 \mapsto a_1(a_1(a_1(c))), y_3 \mapsto a_2(a_2(c)), z_3 \mapsto g_3(c)\}$

Then, a recipe is $\zeta = f(b_1(a_1(x_3)), g_1(z_3), b_2(a_2(a_2(y_3))))$ since $\zeta\sigma \to_R f(c, c, c)$.

As a corollary of Lemma 29 we obtain the following.

▶ **Corollary 31.** *The deduction problem is undecidable for the class of graph-embedded convergent TRSs.*

▶ Remark 32. Note that the knowledge problems of deduction and static-equivalence were already proven undecidable in general in [1], where a reduction from PCP is also used. However, the system used in the proof from [1] is not graph-embedded and it's not clear how to directly adapt that proof to the graph-embedded case of Lemma 29.

4.2 Decidable Knowledge Problems

We consider below a restricted form of the graph-embedded TRS for which we can show decidability of the knowledge problems.

▶ **Definition 33** (Contracting TRS). *A TRS, R, is* contracting *if for each $l \to r \in R$ such that r is not a constant, we have that $l \approx r$ and $depth(r) \leq 2$, or $l \to^{+}_{R_{gemb}} \cdot \approx r$ and r is a well-formed term and $depth(r) \leq 1$, where, considering the rules of Definition 15:*

- *root application of rule (1) are applied before other rules in R_{gemb}.*
- *if rule (1), $f(x_1, \dots, x_n) \to x_i$, is applied below the root then only a variable instance of it is applied and if x_i is not removed by a latter rule then there exist a rule $l' \to x_i \in R$ and a position q such that $l'|_q = f(x_1, \dots, x_n)$.*

- *if rule (2),* $f(x_1, \ldots,\ x_{i-1},\ x_i,\ x_{i+1}\ \ldots,\ x_n) \to f(x_1, \ldots,\ x_{i-1},\ x_{i+1},\ \ldots,\ x_n)$*, is applied then only a variable instance of it is applied.*
- *if rule (4),* $f(x_1, \ldots, x_{i-1}, g(\bar{z}), x_{i+1},\ \ldots, x_m) \to f(x_1, \ldots,\ x_{i-1}, \bar{z}, x_{i+1},\ \ldots, x_m)$*, is applied then only a variable instance of it is applied and if all* $\bar{z}$ *are not removed by latter rules, then for each* z_i *not removed there exists a rule* $l' \to z_i \in R$ *such that* $l'|_q = g(\bar{z})$*.*
- *for* $\approx$*, only a single* $\approx_s$*-permutation can be applied at the root followed by one or no* $\approx_l$*-permutations at the variable positions. In addition:*
  - *if a variable* x *occurs in a direct subterm* $C[x]$ *of* l *not equal to* x*, and by application of* $\approx$ *the variable* x *occurs in a direct subterm of* r *not equal to* $C[x]$*, then there exist a rule* $l' \to x \in R$ *and a position* q *such that* $l'|_q = C[x]$*.*

Although the definition restricts the set of graph-embedded systems it is still sufficient to model many security protocols of interest. We include several such examples below.

▶ **Example 34.** Consider several convergent TRSs given in previous examples:

- The theory of blind signatures from Example 2 is contracting. Consider the rule $\mathit{unblind}(\mathit{sign}(\mathit{blind}(x, y), z), y) \to \mathit{sign}(x, z)$ and the rewriting

  $$\mathit{unblind}(\mathit{sign}(\mathit{blind}(x, y), z), y) \to^*_{R_{gemb}} \mathit{sign}(x, z).$$

  The chain of rules in the rewriting could be:
  - rule (1) - placing *sign* at the root,

    $$\mathit{unblind}(\mathit{sign}(\mathit{blind}(x, y), z), y) \to_{R_{gemb}} \mathit{sign}(\mathit{blind}(x, y), z),$$

  - rule (4) - removing *blind*, $\mathit{sign}(\mathit{blind}(x, y), z) \to_{R_{gemb}} \mathit{sign}(x, y, z)$,
  - rule (2) - removing y, $\mathit{sign}(x, y, z) \to_{R_{gemb}} \mathit{sign}(x, z)$.

  Since rule (4) was applied, removing *blind* and variable x was not removed by rule (2), there needs to be a rule $l \to r$ in R such that $x = r$, which is the case with the rule $\mathit{unblind}(\mathit{blind}(x, y), y) \to x$.
- The theory of addition, introduced in Example 22, is a contracting TRS. Notice that $\mathit{plus}(x, s(y)) \approx \mathit{plus}(s(x), y)$ and that the cap, here $s()$, has been removed from y. Thus, there needs to be a rule, $l \to r$, such that $l|_p = s(y)$ and $r = y$. This is exactly the rule $\mathit{pred}(s(x)) \to x$.
- The theory of prefix with pairing from Example 23 is a contracting TRS.
- Any subterm convergent TRS such that the right-hand side is either a strict subterm or a constant is contracting, like for instance the theory of pairing with encryption, $R = \{\mathit{fst}(\langle x, y\rangle) \to x, \mathit{snd}(\langle x, y\rangle) \to y, \mathit{dec}(\mathit{enc}(x, y), y) \to x\}$, and the theory of Example 25.

▶ **Example 35.** Consider several of the previous example TRSs:

- The theory of trap-door commitment of Definition 24 is not a contracting TRS. Interestingly, this theory is also not locally stable [15]. However, if we add the rules,

  $$\mathit{fst}(f(x_1, x_2, x_3, x_4)) \to x_1, \mathit{snd}(f(x_1, x_2, x_3, x_4)) \to x_2, \mathit{thd}(f(x_1, x_2, x_3, x_4)) \to x_3,$$

  then the theory is contracting and locally stable.
- The theory of Example 20 is not contracting. Notice that for the rule $\mathit{mal}(\mathit{enc}(x, y), z) \to \mathit{enc}(z, y)$, the node labeled with z is moved under the *enc* node on the right-hand side. This violates the requirements of Definition 33, specifically requiring rule (3). Thus, even with additional rules, it cannot be made to be contracting. This theory is also not locally stable, as shown in [15].

▶ Remark 36. If we consider now the TRS from the undecidability proof of Lemma 29 we can see that while the system is graph-embedded it is not contracting. The system can be made contracting by adding a set of rewrite rules of the form $a_1(x) \to x,\ b_1(x) \to x,\ a_2(x) \to x, \ldots$. Then, clearly for this new system both the deduction problem and unification problem detailed in the proof are decidable.

We now develop a few results and definitions we need to show the decidability of the knowledge problems for contracting, graph-embedded convergent systems.

▶ **Definition 37** (Context Bound). *Let $ar(\Sigma)$ denote the maximal arity of any function symbol in Σ. Define the context bound of a graph-embedded TRS, $R = \{l_i \to r_i\},\ 1 \leq i \leq n$, as*

$$c_R = max_{1 \leq i \leq n}(|l_i|, ar(\Sigma) + 1)$$

▶ **Example 38.** For the theory of malleable encryption from Example 20, $c_{R_{mal}} = 5$. For the theory of blind signatures, R_{blind}, from Example 2, $c_{R_{blind}} = 7$.

▶ **Definition 39** (Graph-Embedded Subterms). *Let R be a contracting TRS and let $st(t)$ be the set of subterms of a term t. Then, the set of graph-embedded subterms of a term t, denoted as $gst(t)$, is defined as: $gst(c) = \{c\}$, where c is a name or a constant, $gst(t) = \{t' | t \to^*_{R_{gemb}} t'' \approx t'$, and t' is a well formed term $\} \cup \bigcup_{t'' \in st(t)} gst(t'')$. Let $\phi = \nu\tilde{n}.\sigma$ be a frame, then $gst(\phi) = \cup_{t \in Ran(\sigma)} gst(t)$.*

Notice that for any term t, $gst(t)$ is a finite set. This is due to the fact that when recursively constructing $gst(t)$ in the second rule of Definition 39, t' is equal or smaller in size to t, and any term $t'' \in st(t)$ must be strictly smaller than t. Thus, we have the following result.

▶ **Lemma 40.** *For any term t and any frame ϕ, $gst(t)$ and $gst(\phi)$ are finite sets.*

Based on the extended definition of subterms, gst, we can now construct a saturation set for frames. Computing such a saturation set is the goal of many procedures that consider security notions such as deducibility. The saturation set represents the knowledge of the attacker and their ability to deduce a term from that knowledge, see [1] for more background.

▶ **Definition 41** (Frame Saturation for Contracting Convergent TRS). *Let $\phi = \nu\tilde{n}.\sigma$ be a frame, and R a contracting convergent TRS. Define the set $sat(\phi)$ to be the smallest set such that $Ran(\sigma) \subseteq sat(\phi)$, and $n \in sat(\phi)$ for every $n \in fn(\phi)$, and closed under the following two rules:*

1. *if $M_1, \ldots, M_l \in sat(\phi)$ and $f(M_1, \ldots, M_l) \in gst(\phi)$, then $f(M_1, \ldots, M_l) \in sat(\phi)$,*
2. *if $M_1, \ldots, M_l \in sat(\phi)$, $C[M_1, \ldots, M_l] \to^{\epsilon}_R M$, where C is a context, $|C| \leq c_R$, $fn(C) \cap \tilde{n} = \emptyset$, and $M \in gst(\phi)$, then $M \in sat(\phi)$.*

▶ Remark 42. It is important to note that $sat(\phi)$ should contain the set of deducible terms from the frame. For example, it would be tempting to just place all of $gst(\phi)$ into $sat(\phi)$ immediately, but this would add non-deducible terms to the set and invalidate the results.

Also notice for Definition 41, by applying an empty context, the second rule ensures that for any $S \in sat(\phi)$, if $S \to^{\epsilon}_R S'$ and $S' \in gst(\phi)$, then $S' \in sat(\phi)$.

This set is also finite which is critical to computing the possible attackers knowledge thus having a finite set is useful for any practical procedure for deciding deducibility.

▶ **Lemma 43.** *For any frame ϕ, $sat(\phi)$ is finite.*

Proof. New terms not originally contained in ϕ are only added to $sat(\phi)$ if they are first contained in $gst(\phi)$. Since $gst(\phi)$ is finite by Lemma 40, $sat(\phi)$ is finite. ◀

The following definition and lemma will be useful in proving the main motivating result as they show key components of the local stability property given in Definition 46.

▶ **Definition 44** (Closure Under Small Context). *Let $\phi = \nu\tilde{n}.\sigma$ be a frame, and R a convergent TRS. A finite set of ground terms, $\mathcal{S}$, is* closed under small ϕ-restricted context *by R if the following property holds: for any context C with $|C| \leq c_R$ and $fn(C) \cap \tilde{n} = \emptyset$, and any $S_1, \ldots, S_l \in \mathcal{S}$, if $C[S_1, \ldots, S_l] \rightarrow^{\epsilon}_{R} M$ then there exist a context C' and $S'_1, \ldots, S'_k \in \mathcal{S}$ such that $|C'| \leq c^2_R$, $fn(C') \cap \tilde{n} = \emptyset$, and $M \rightarrow^{*}_{R} C'[S'_1, \ldots, S'_k]$. When ϕ is clear from the context, $\mathcal{S}$ is said to be* closed under small context *by R.*

Using c^2_R as an upper bound is somewhat arbitrary since we need just some fixed bound. We use c^2_R since it is sufficient for the results in this paper and it is the bound used in [1].

▶ **Lemma 45.** *For any frame ϕ and any contracting convergent TRS R, let $sat(\phi)$ be the set given in Definition 41. Then, $sat(\phi)$ is closed under small context by R.*

Proof. See Appendix A. ◀

We can now introduce the local stability property, which if satisfied ensures the decidability of deduction. The local stability property was introduced in [1] and improved in [6]. A simplified version of this definition is introduced below. It is simplified because we don't consider AC-symbols as in [1, 6].

▶ **Definition 46** (Local Stability [1]). *A convergent TRS, R, is* locally stable *if, for every frame $\phi = \nu\tilde{n}.\sigma$, where σ is a ground R-normalized substitution, there exists a finite set $sat(\phi)$ of ground terms such that:*

- *$Ran(\sigma) \subseteq sat(\phi)$ and $n \in sat(\phi)$, for all $n \in fn(\phi)$;*
- *if $M_1, \ldots, M_k \in sat(\phi)$ and $f(M_1, \ldots, M_k) \in st(sat(\phi))$, then $f(M_1, \ldots, M_k) \in sat(\phi)$;*
- *if $C[S_1, \ldots, S_l] \rightarrow^{\epsilon}_{R} M$, where C is a context with $|C| \leq c_R$ and $fn(C) \cap \tilde{n} = \emptyset$, and $S_1, \ldots, S_l \in sat(\phi)$, then there exist a context C' and $S'_1, \ldots, S'_k \in sat(\phi)$ such that $|C'| \leq c^2_R$, $fn(C') \cap \tilde{n} = \emptyset$, and $M \rightarrow^{*}_{R} C'[S'_1, \ldots, S'_k]$;*
- *if $M \in sat(\phi)$ then $\phi \vdash M$.*

▶ Remark 47. The existence of a set $sat(\phi)$ in the above definition means that any set satisfying the conditions of Definition 46 is sufficient. In Definition 41 we give a particular such set for contracting convergent TRSs which satisfies the conditions of Definition 46, as shown below.

In a locally stable TRS, any deduction problem reduces to check finitely many terms that could be possible recipes of the input, and any static equivalence problem reduces to checking finitely many equations between bounded terms satisfying the name restriction [1].

For any contracting convergent TRS, Lemma 45 establishes all but the last item of Definition 46, and this item has already been shown in [1].

▶ **Lemma 48** ([1]). *For any frame ϕ and any ground term M, if $M \in sat(\phi)$ then $\phi \vdash M$.*

This result is proven in [1] where they also consider the more complicated case of systems with AC-symbols.

▶ **Theorem 49.** *Any contracting convergent TRS is locally stable.*

Proof. The first two conditions follow from Definition 41 where $sat(\phi)$ is given in the particular case of a contracting convergent TRS. Then, the third condition follows from Lemma 45. The final condition follows from Lemma 48. ◀

▶ **Example 50.** Continuing Example 34:

- Since the theory of blind signatures is a contracting convergent TRS, it is locally stable by Theorem 49. Since this theory doesn't contain an AC-symbol it is also locally finite [1]. The theory of blind signatures being both locally stable and locally finite, both deduction and static-equivalence are decidable [1].
- All the theories from Example 34 are thus locally stable and locally finite, thus both knowledge problems are decidable.
- By the same argument the theory from Example 22 is locally stable and both knowledge problems are decidable.
- By the same argument any subterm convergent theory such that the right-hand side is either a strict subterm or a constant is locally stable and both knowledge problems are decidable.

Directly from Theorem 49 and the result in [1], which establish the decidability of deduction and static equivalence for locally stable and finite theories, we obtain the following corollary.

▶ **Corollary 51.** *The deduction and static equivalence problems are both decidable for the class of contracting convergent TRSs.*

5 Relation to the Finite Variant Property

The Finite Variant Property (FVP) is a useful property which is utilized in a number of applications, including protocol analysis. See for example [12, 17]. Before discussing the relation it's useful to introduce the following definition.

▶ **Definition 52** (Boundedness Property). *A convergent TRS, R, has the* boundedness property *if* $\forall t \exists n \forall \sigma \;:\; t(\sigma{\downarrow}) \rightarrow_R^{\leq n} (t\sigma){\downarrow}$. *That is, for any term t there exists a bound, n, on the number of step required to reach the normal form, and this bound is independent of the substitution.*

▶ Remark 53. It's been shown in [11] that a TRS has the FVP iff it has the boundedness property of Definition 52. See also [13, 18] for more background.

One could naturally ask if the graph-embedded or contracting definitions just lead to systems with the FVP. This is not the case but some of the examples above, such as blind signatures, do have the FVP. This is not surprising, given that the FVP can be useful for showing things like termination. Therefore, a more interesting question could be: are there interesting examples from the protocol analysis literature for which deduction and static equivalence are decidable, do not have the FVP, but are representable by contracting convergent TRSs? Here we answer positively this question.

▶ **Example 54.** Consider again the theory of Addition, R_{add}, from Example 22. R_{add} is a contracting convergent TRS, is locally stable, and contains no AC-symbols, thus deduction and static equivalence are decidable. However, R_{add} does not have the FVP, we can see this by considering the rule $plus(x, s(y)) \rightarrow plus(s(x), y)$ and the boundedness property. Notice that for any finite bound n one can select a normal form substitution, σ, such that $plus(x, s(y))\sigma \rightarrow_{R_{add}}^{\geq n} (plus(x, s(y))\sigma){\downarrow}$. Namely, $\sigma = \{y \mapsto s^{n+1}(z)\}$. Since R_{add} does not have the boundedness property it can't have the FVP [11]. Yet, R_{add} is a contracting convergent TRS. Notice that the second and third rules are already subterm. The first rule is obtained by applying Definition 13. Therefore, R_{add} satisfies Corollary 51.

6 Conclusions and Future Work

In this paper, we have introduced the idea of graph-embedded term rewrite systems and shown their applicability in protocol analysis for identifying protocols with the local stability property. This in turn allows for the identification of protocols with decidable deduction and static-equivalence problems. However, this could be just the first step as there are many additional questions about the use of graph-embedded systems applied to the protocol analysis domain and also outside that domain.

With respect to the current paper, a natural question arises. While the knowledge problems are undecidable for graph-embedded convergent systems in general and that they are decidable for contracting, graph-embedded convergent systems, there is a gap between the two classes of systems. That is, how much can the contracting subclass be extended before the undecidable barrier is encountered? In this direction, we could try to weaken the rule application strategy used in Definition 33.

With respect to additional security protocol applications there are several interesting areas that could be explored:

- It would be interesting to consider termination conditions of various procedures [1, 12, 15, 17] with respect to graph-embedded systems. That is, do graph-embedded systems provide any help with obtaining termination guarantees?
- The cap problem, developed in [5] could be viewed as a particular form of deduction where one wants to determine whether a given secret constant is deducible or not. We conjecture that the cap problem is decidable for contracting convergent TRSs and undecidable for general graph-embedded convergent TRSs. A proof of this would be useful.
- It would be useful to consider additional properties that have been developed for use in protocol analysis. For example, *layer-convergence* is a property developed in [9] where it's shown that the YAPA procedure for protocol analysis will not fail on theories with this property. While termination is not ensured, this does provide a useful condition for knowing if the procedure can be used. Currently, we don't know how the graph-embedded property compares to layer-convergence. Note, checking for the graph-embedded property is relatively easy. Thus, if a restricted form of graph embedding could be shown to be related to layer-convergence, then such a graph embedding could be a useful way to identify layer-convergent systems. This in turn would be useful for identifying systems for which the YAPA procedure could be applied.

With respect to graph theory ideas, we are also interested in knowing if additional graph theory ideas could be useful in symbolic security protocol analysis:

- Of course not absolutely all theories considered in [1, 12, 15, 17] are graph-embedded. It would be interesting to know if such systems could be considered via graph minor concepts?
- In addition, are graph minor relations such as topological minors useful? Given standard graph theory this would seem not to be the case. The containment of the topological minor relation in the graph minor relation would appear to rule this out. However, this may not be completely true in the term graph domain, where we must obey the standard well-formed term requirements.

References

1 Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. *Theor. Comput. Sci.*, 367(1-2):2–32, 2006.

2 Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In Chris Hankin and Dave Schmidt, editors, *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, pages 104–115. ACM, 2001.

3 Siva Anantharaman, Hai Lin, Christopher Lynch, Paliath Narendran, and Michaël Rusinowitch. Unification modulo homomorphic encryption. *J. Autom. Reason.*, 48(2):135–158, 2012.

4 Siva Anantharaman, Paliath Narendran, and Michael Rusinowitch. Intruders with caps. Research report, Laboratoire d'Informatique Fondamentale d'Orléans, 2007. URL: `https://hal.science/hal-00144178`.

5 Siva Anantharaman, Paliath Narendran, and Michaël Rusinowitch. Intruders with caps. In Franz Baader, editor, *Term Rewriting and Applications, 18th International Conference, RTA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4533 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2007.

6 Mauricio Ayala-Rincón, Maribel Fernández, and Daniele Nantes-Sobrinho. Intruder deduction problem for locally stable theories with normal forms and inverses. *Theor. Comput. Sci.*, 672:64–100, 2017.

7 Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

8 Franz Baader and Wayne Snyder. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 445–532. Elsevier and MIT Press, 2001.

9 Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune. YAPA: A generic tool for computing intruder knowledge. *ACM Trans. Comput. Log.*, 14(1):4, 2013.

10 Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *2004 IEEE Symposium on Security and Privacy (S&P 2004), 9-12 May 2004, Berkeley, CA, USA*, pages 86–100. IEEE Computer Society, 2004.

11 Christopher Bouchard, Kimberly A. Gero, Christopher Lynch, and Paliath Narendran. On forward closure and the finite variant property. In Pascal Fontaine, Christophe Ringeissen, and Renate A. Schmidt, editors, *Frontiers of Combining Systems - 9th International Symposium, FroCoS 2013, Nancy, France, September 18-20, 2013. Proceedings*, volume 8152 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2013.

12 Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. *ACM Trans. Comput. Log.*, 17(4):23:1–23:32, 2016.

13 Hubert Comon-Lundh and Stéphanie Delaune. The finite variant property: How to get rid of some algebraic properties. 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 294–307. Springer, 2005.

14 Véronique Cortier, Stéphanie Delaune, and Pascal Lafourcade. A survey of algebraic properties used in cryptographic protocols. *J. Comput. Secur.*, 14(1):1–43, 2006.

15 Ştefan Ciobâcă, Stéphanie Delaune, and Steve Kremer. Computing knowledge in security protocols under convergent equational theories. *J. Autom. Reasoning*, 48(2):219–262, 2012.

16 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, third edition, 2006.

17 Jannik Dreier, Charles Duménil, Steve Kremer, and Ralf Sasse. Beyond subterm-convergent equational theories in automated verification of stateful protocols. In Matteo Maffei and Mark Ryan, editors, *Principles of Security and Trust - 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10204 of *Lecture Notes in Computer Science*, pages 117–140. Springer, 2017.

18 Santiago Escobar, Ralf Sasse, and José Meseguer. Folding variant narrowing and optimal variant termination. *J. Log. Algebr. Program.*, 81(7-8):898–928, 2012.

A Technical Appendix

Let us first introduce some additional notations that are useful in the proofs of our results. Given a set $\bar{S}$ of ground term, a frame $\phi = \nu\tilde{n}.\sigma$ and a TRS R, a *context instantiated with terms in* $\bar{S}$ is a ground term of the form $u\alpha$ where u is a term including no names in $\tilde{n}$, and α a substitution such that $Dom(\alpha) = Var(u)$ and $Ran(\alpha) \subseteq \bar{S}$. The term u is called the *context part of* t. The term t is called a *small* context if the size of its context part u satisfies $|u| \leq c_R$. A term denoted by $u[S_1, \dots, S_n]$ for $S_1, \dots, S_n \in \bar{S}$ corresponds to the context instantiated with terms in $\bar{S}$ defined by $u[x_1, \dots, x_n]\{x_1 \mapsto S_1, \dots, x_n \mapsto S_n\}$, where the set of variables in $u[x_1, \dots, x_n]$ is assumed to be $\{x_1, \dots, x_n\}$.

A.1 Additional Lemmas

The following two technical lemmas are useful to prove the closure property stated by Lemma 45 for any contracting convergent TRS.

A rewrite step applied at the root position is denoted by $\xrightarrow{\epsilon}$, while a rewrite step applied at some non-rooted position is denoted by $\xrightarrow{\neq\epsilon}$.

▶ **Lemma 55.** *Let R be a contracting convergent TRS. Let $l \xrightarrow{\neq\epsilon}{}^{+}_{R_{gemb}} \cdot \approx r$ such that l is a subterm of the left-hand side l' of a rule $l' \to r$ in R where $depth(r) = 1$. For any variable x in r occurring in l at a position p such that $|p| > 1$, and any substitution φ, the following is true: for any positions q, q' such that $\epsilon < q < q' \leq p$, $q' = q.i$, $l(q)$ is a function symbol f, if $l|_q\varphi \in sat(\phi)$ then there exists a* projecting *rule $C[f(x_1, \dots, x_n)] \to x_i$ in R.*

Proof. The proof is by induction on $dp(l) = \max_{\{p \ | \ l(p) \in Var(r)\}} |p|$. For the base case, one can check that the property holds for $dp(l) = 2$ due the particular form of r. For the induction step, consider a derivation

$$\begin{aligned} & l[f(y_1, \dots, y_{j-1}, g(\dots, x_i, \dots), y_{j+1}, \dots, y_n)] \\ & \qquad \xrightarrow{\neq\epsilon}{}^{+}_{R_{gemb}} \quad l[f(v_1, \dots, v_{j-1}, v_j, v_{j+1}, \dots, v_n)] \\ & \qquad \xrightarrow{\neq\epsilon}{}^{*}_{R_{gemb}} \quad r \end{aligned}$$

where:

- the $\xrightarrow{\neq\epsilon}{}^{+}_{R_{gemb}}$ derivation consists of a single rule (1) or a single rule (4) followed by the repeated application of rule (2) to retrieve a well-formed term of arity n;
- x_i occurs in l at a position p such that $|p| = dp(l)$ and $x_i \in Var(r)$.

Let $l' = l[f(v_1, \dots, v_{j-1}, v_j, v_{j+1}, \dots, v_n)]$. We have $dp(l') < dp(l)$ and so we can assume that the property holds for l'. Moreover, v_j occurs in r, otherwise it would contradict that x_i occurs in r. By the induction hypothesis, there must exist a projecting rule to get v_j from $f(v_1, \dots, v_{j-1}, v_j, v_{j+1}, \dots, v_n)$. This projection rule can be reused for the redex $f(y_1, \dots, y_{j-1}, g(\dots, x_i, \dots), y_{j+1}, \dots, y_n)$ of l. By Definition 33, there exists a projecting rule to get x_i from the subterm $g(\dots, x_i, \dots)$ of l. For all the strict subterms of l above the redex which are on the path to the root of l, we can reuse all the projecting rules available for all the strict subterms of l' above $f(v_1, \dots, v_{j-1}, v_j, v_{j+1}, \dots, v_n)$ which are on the path to the root of l'. Consequently, there is a projecting rule for all the strict subterms of l above x_i at position p which are on the path to the root of l. Thus, the property holds for l. ◀

▶ **Lemma 56.** *Let $l \approx r$ be a rule of a contracting convergent TRS such that $depth(l) = depth(r) = 2$. For any variable x in r occurring at a position p of l in a non-variable direct subterm $l|_q$ of l which is not a direct subterm of r, and any substitution φ, the following is true: if $l|_q\varphi \in sat(\phi)$, then $l|_p\varphi \in sat(\phi)$.*

Proof. Thanks to Definition 33. ◀

A.2 Proof of Lemma 45

Proof. Let us analyse the different forms of derivation that may occur in the definition a rule of a contracting TRS.

First, if $l \xrightarrow{\epsilon}_{R_{gemb}} r$, then r is direct subterm of l. According to [1], for any term t corresponding to a small context instantiated by terms in $sat(\phi)$ such that $t = l\varphi$, the term $r\varphi$ remains a small context instantiated by terms in $sat(\phi)$. Then, we prove by induction on the length of the derivation that the same property also holds for $\xrightarrow{\epsilon}^*_{R_{gemb}}$.

Second, consider a derivation $l \xrightarrow{\neq\epsilon}^+_{R_{gemb}} r$ where $depth(r) \leq 1$. The case $depth(r) = 0$ is easy since it corresponds to the classical subterm case (see above). So, let us assume $r = h(x_1, \ldots, x_n)$ and a small context instantiated by terms in $sat(\phi)$, say t, such that $t = l\varphi$. Let i be any integer in $\{1, \ldots, n\}$. Any x_i occurs in l either at depth at most 1 or at some depth strictly greater than 1.

- If x_i occurs in l at depth at most 1, then $x_i\varphi$ is a small context instantiated by terms in $sat(\phi)$.
- if x_i occurs in l at some depth strictly greater than 1, then $x_i\varphi$ is a small context instantiated by terms in $sat(\phi)$ by Lemma 55.

Then, $r\varphi = h(x_1\sigma, \ldots, x_n\sigma)$ is a small context instantiated by terms in $sat(\phi)$. Indeed, by construction, the context part of $r\varphi$ cannot be greater than the context part of $l\sigma$.

Third, consider the case $l \approx r$ where $depth(r) \leq 2$. By definition of $\approx$, we have $depth(l) = depth(r)$. The case $depth(r) \leq 1$ being easy, let us assume $depth(l) = depth(r) = 2$, $r = h(r_1, \ldots, r_n)$, and a small context instantiated by $sat(\phi)$, say t, such that $t = l\varphi$. Let i be any integer in $\{1, \ldots, n\}$.

- If r_i is a variable occurring at depth at most 1 in l, then $r_i\varphi$ is a small context instantiated by terms in $sat(\phi)$.
- If r_i is a variable occurring at depth 2 in l, then $r_i\varphi$ is a small context instantiated by terms in $sat(\phi)$ by Lemma 56.
- If r_i is a non-variable term occurring as a direct subterm l_j of l for some $j \in [1, n]$, then $r_i\varphi = l_j\varphi$ is a small context instantiated by terms in $sat(\phi)$.
- If r_i is a non-variable term $f(\bar{x})$ not occurring as a direct subterm of l, then there are two cases for any variable $x \in \bar{x}$: if x occurs at depth at most 1 in l, then $x\varphi$ is a small context instantiated by terms in $sat(\phi)$; otherwise x also occurs at depth 2 in l and $x\varphi$ is a small context instantiated by terms in $sat(\phi)$ by Lemma 56.

Then, $r\varphi = h(r_1\sigma, \ldots, r_n\sigma)$ is a small context instantiated by terms in $sat(\phi)$. Indeed, by definition of $\approx$, the context part of $r\varphi$ cannot be greater than the context part of $l\sigma$. ◀

Labelled Tableaux for Linear Time Bunched Implication Logic

Didier Galmiche ✉
Université de Lorraine, CNRS, LORIA, Nancy, France

Daniel Méry ✉
Université de Lorraine, CNRS, LORIA, Nancy, France

Abstract

In this paper, we define the logic of Linear Temporal Bunched Implications (LTBI), a temporal extension of the Bunched Implications logic BI that deals with resource evolution over time, by combining the BI separation connectives and the LTL temporal connectives. We first present the syntax and semantics of LTBI and illustrate its expressiveness with a significant example. Then we introduce a tableau calculus with labels and constraints, called $\mathsf{T}_{\mathsf{LTBI}}$, and prove its soundness w.r.t. the Kripke-style semantics of LTBI. Finally we discuss and analyze the issues that make the completeness of the calculus not trivial in the general case of unbounded timelines and explain how to solve the issues in the more restricted case of bounded timelines.

2012 ACM Subject Classification Theory of computation → Proof theory

Keywords and phrases Temporal Logic, Bunched Implication Logic, Labelled Deduction, Tableaux

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.31

Funding Work supported by the ANR NARCO Project (ANR Grant 21-CE48-0011).

1 Introduction

The notion of resource is a fundamental concept in various fields, especially in computer science. For instance, resources play a central role in designing systems such as computer networks or programs that access memory and manipulate data structures using pointers [9]. It is well known that Linear Logic [8] emphasizes an aspect of resource management that is closely related with resource consumption, whereas the Logic of Bunched Implications (BI) [13, 15] focuses more on aspects related with resource sharing and separation [7]. Recent works consider modal and/or epistemic extensions of BI and Boolean BI (BBI) in order to deal with more dynamic aspects of resource management [3, 4].

In this paper, we introduce the logic of Linear Temporal Bunched Implications (LTBI), a temporal extension of BI that deals with resource evolution over time. LTBI extends BI with operators borrowed from Linear Temporal Logic (LTL) to handle temporal aspects of computer systems [16]. Both temporal and separation logics have proven themselves successful in the design and formal verification of computer systems. Temporal logics are also well-known for their ability to state and verify safety and liveness properties (e.g., using Buchi automata [11]) and have a wide range of applications including model checking, concurrent programming, and reactive systems [2]. It is therefore interesting to study a logic for which the spatial connectives of BI cohabit with the temporal modalities of LTL.

Let us remark that a temporal extension of BI, called tBI, has been introduced in [10]. This extension derives an enriched sequent calculus from LBI (the standard sequent calculus of BI) and gives various embedding of tBI into BI. In this paper, we follow another approach based on labelled tableaux, in the spirit of [3, 4]. Although tBI might at first glance seem very similar to our logic LTBI, they bear significant differences that we discuss in details in Section 5 (after the required technical notions have been introduced).

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 31; pp. 31:1–31:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The paper is organized as follows: in Section 2 we describe the syntax and semantics of our LTBI logic that mixes the separation connectives of BI [7] with the temporal connectives $\Diamond$, $\Box$, $\circ$ of LTL. We also illustrate the expressiveness of LTBI with a significant example. In Section 3, we introduce $\mathsf{T}_{\mathsf{LTBI}}$, our labelled tableau calculus for LTBI in the spirit of [7, 3]. We then illustrate how it works with some examples. In Section 4 we prove the soundness of the $\mathsf{T}_{\mathsf{LTBI}}$ calculus. Finally, Section 5 ends the paper with a discussion of the several completeness issues that arise when trying to keep the labels constraints isomorphic to the standard linear order of the natural numbers.

2 Linear Temporal Bunched Implication Logic

Separation logics like BI and its variants are well suited to state (static) spatial properties about resources [6, 7]. DBI [3], a recent extension of BI with S4 modalities $\Diamond$ and $\Box$, opens the way for more dynamic aspects of resource management, but only to some extent. In this section we introduce Linear Temporal BI (LTBI) as a combination of BI and LTL [2, 16] interpreted on a discrete timeline.

2.1 Syntax and Semantics of LTBI

LTBI is an extension of BI [7, 14] with the three main LTL unary connectives $\Box$, $\Diamond$ and $\circ$. We do not consider the binary connectives U and R ("until" and "release") in this paper and leave them for future work.

▶ **Definition 1.** *Let* $\mathbf{P}$ *be a countable set of propositional letters. The set* $\mathbf{F}$ *of* LTBI *formulas is given by the following grammar:*

$$\mathrm{A} ::= \mathbf{P} \mid \top \mid \bot \mid \mathrm{A} \wedge \mathrm{A} \mid \mathrm{A} \vee \mathrm{A} \mid \mathrm{A} \to \mathrm{A} \mid \mathrm{I} \mid \mathrm{A} * \mathrm{A} \mid \mathrm{A} \mathrel{-\!\!*} \mathrm{A} \mid \Box\mathrm{A} \mid \Diamond\mathrm{A} \mid \circ\mathrm{A}$$

Additive negation is defined as usual as $\mathrm{A} \to \bot$.

In order to define a Kripke-style semantics for LTBI, we first introduce the notions of linear resource frames (LRF), interpretation and models.

▶ **Definition 2.** *An* LTBI-frame *is a structure* $\mathcal{R} = (\mathbf{R}, \star, \epsilon, \leqslant^{\mathfrak{r}}, \pi, \mathbf{S}, \leqslant^{\mathfrak{s}}, s_0)$, *where:*

- $(\mathbf{R}, \star, \epsilon, \leqslant^{\mathfrak{r}}, \pi)$ *is a* resource monoid, *i.e., a partially ordered commutative monoid of elements, called* resources, *such that:*
  - ϵ *is the unit of* $\star$, *i.e.* $\epsilon \star r = r \star \epsilon = r$,
  - π *is the greatest element of* $\mathbf{R}$ *w.r.t.* $\leqslant^{\mathfrak{r}}$ *and* $\forall\, r \in \mathbf{R}.\, r \star \pi = \pi$,
  - $\forall\, r, r', r'' \in \mathbf{R}.\, r \leqslant^{\mathfrak{r}} r'$ *implies* $r \star r'' \leqslant^{\mathfrak{r}} r' \star r''$.
- $(\mathbf{S}, \leqslant^{\mathfrak{s}}, s_0)$ *is a* discrete timeline, *i.e., a subset of* $\mathbb{N}$ *totally ordered by* $\leqslant^{\mathfrak{s}}$ *taken as the restriction to* $\mathbf{S}$ *of the standard order on* $\mathbb{N}$, *and such that* s_0 *is the least element of* $\mathbf{S}$ *w.r.t.* $\leqslant^{\mathfrak{s}}$. *The elements of* $\mathbf{S}$ *are called* states.

For all $s \in \mathbf{S}$, *we define* $N(s)$ *as the set* $\{\, s' \mid s' \in \mathbf{S}$ *and* $s <^{\mathfrak{s}} s' \,\}$. *We then write* $\mathfrak{n}$ *for the function* "next" *induced on* $\mathbf{S}$ *by* $\leqslant^{\mathfrak{s}}$ *and such that for all* $s \in \mathbf{S}$, $\mathfrak{n}(s)$ *is the least element of* $N(s)$ *if* $N(s)$ *is not empty and undefined otherwise.*

▶ **Definition 3.** *An* LTBI-valuation *is a partial function* $[\cdot] : \mathbf{P} \to \wp(\mathbf{R} \times \mathbf{S})$ *that satisfies the following conditions:*

$(\mathcal{M}_{\mathrm{K}})$ $\forall\, \mathrm{p} \in \mathbf{P}.\, \forall\, s \in \mathbf{S}.\, \forall\, r, r' \in \mathbf{R}.$ *if* $r \leqslant^{\mathfrak{r}} r'$ *and* $(r, s) \in [\mathrm{p}]$ *then* $(r', s) \in [\mathrm{p}]$,
$(\mathcal{M}_{\pi})$ $\forall\, \mathrm{p} \in \mathbf{P}.\, \forall\, s \in \mathbf{S}.\, (\pi, s) \in [\mathrm{p}]$.

▶ **Definition 4.** *An* LTBI-model *is a triple* $\mathcal{M} = (\mathcal{R}, [\cdot], \Vdash)$*, where* $\mathcal{R}$ *is an* LTBI*-frame,* $[\cdot]$ *is an* LTBI*-valuation and* $\Vdash \subseteq \mathbf{R} \times \mathbf{S} \times \mathbf{F}$ *is the smallest* forcing relation *such that:*

- $(r, s) \Vdash \mathrm{p}$ *iff* $(r, s) \in [\mathrm{p}]$
- $(r, s) \Vdash \mathrm{I}$ *iff* $\epsilon \leqslant^{\mathfrak{r}} r$
- $(r, s) \Vdash \bot$ *iff* $\pi \leqslant^{\mathfrak{r}} r$
- $(r, s) \Vdash \top$ *always*
- $(r, s) \Vdash \mathrm{A} \vee \mathrm{B}$ *iff* $(r, s) \Vdash \mathrm{A}$ *or* $(r, s) \Vdash \mathrm{B}$
- $(r, s) \Vdash \mathrm{A} \wedge \mathrm{B}$ *iff* $(r, s) \Vdash \mathrm{A}$ *and* $(r, s) \Vdash \mathrm{B}$
- $(r, s) \Vdash \mathrm{A} \to \mathrm{B}$ *iff* $\forall r' \in \mathbf{R}.$ *if* $r \leqslant^{\mathfrak{r}} r'$ *and* $(r', s) \Vdash \mathrm{A}$ *then* $(r', s) \Vdash \mathrm{B}$
- $(r, s) \Vdash \mathrm{A} * \mathrm{B}$ *iff* $\exists r', r'' \in \mathbf{R}.\, r' \star r'' \leqslant^{\mathfrak{r}} r, (r', s) \Vdash \mathrm{A}$ *and* $(r'', s) \Vdash \mathrm{B}$
- $(r, s) \Vdash \mathrm{A} \mathrel{-\!\!*} \mathrm{B}$ *iff* $\forall r', r'' \in \mathbf{R}.$ *if* $(r', s) \Vdash \mathrm{A}$ *and* $r' \star r \leqslant^{\mathfrak{r}} r''$ *then* $(r'', s) \Vdash \mathrm{B}$
- $(r, s) \Vdash \Box\mathrm{A}$ *iff* $\forall s' \in \mathbf{S}.$ *if* $s \leqslant^{\mathfrak{s}} s'$ *then* $(r, s') \Vdash \mathrm{A}$
- $(r, s) \Vdash \Diamond\mathrm{A}$ *iff* $\exists s' \in \mathbf{S}.\, s \leqslant^{\mathfrak{s}} s'$ *and* $(r, s') \Vdash \mathrm{A}$
- $(r, s) \Vdash \circ\mathrm{A}$ *iff* $\exists s' \in \mathbf{S}.\, s' = \mathfrak{n}(s)$ *and* $(r, s') \Vdash \mathrm{A}$

▶ **Definition 5.** *A formula* A *is* satisfied in an LTBI-model $\mathcal{M}$*, written* $\mathcal{M} \vDash \mathrm{A}$*, iff* $(\epsilon, s) \Vdash \mathrm{A}$ *for all* $s \in \mathbf{S}$*. A formula* A *is* valid*, written* $\vDash\mathrm{A}$*, iff it is satisfied in all* LTBI*-models.*

It is routine to show that conditions $\mathcal{M}_{\mathrm{K}}$ and $\mathcal{M}_{\pi}$ of Definition 3 extend from propositional letters to arbitrary formulas, as stated in the following Lemma.

▶ **Lemma 6.** *For all* LTBI*-models* $\mathcal{M}$*:*
$(\mathcal{M}_{\mathbf{K}})$ $\forall \mathrm{A} \in \mathbf{F}. \forall s \in \mathbf{S}. \forall r, r' \in \mathbf{R}.$ *if* $r \leqslant^{\mathfrak{r}} r'$ *and* $(r, s) \Vdash \mathrm{A}$ *then* $(r', s) \Vdash \mathrm{A}$,
$(\mathcal{M}_{\pi})$ $\forall \mathrm{A} \in \mathbf{F}. \forall s \in \mathbf{S}. (\pi, s) \Vdash \mathrm{A}$.

Let us remark that the resource semantics we use for LTBI is based on total (and not partial) resource monoids to avoid tricky additional definedness conditions. The introduction of a greatest element π at which all formulas are satisfied is therefore required in the presence of $\bot$ (as explained in [7], for example, to enforce the validity of BI formulas such as $\mathrm{A} * (\mathrm{A} \mathrel{-\!\!*} \bot)$ where A is a theorem of intuitionistic logic).

2.2 Expressiveness of LTBI

To illustrate what kind of properties LTBI is able to express, let us consider the timeline $(\mathbf{S} = [2023 - 2025], \leqslant^{\mathfrak{s}}, 2023)$ and the resource monoid $(\mathbf{R} = \mathbb{N} \cup \{\, \infty \,\}, +, 0, \leqslant^{\mathfrak{r}}, \infty)$, where $\leqslant^{\mathfrak{r}}$ and $+$ are the extensions of the standard order and of the standard addition on natural numbers such that $r \leqslant^{\mathfrak{r}} \infty$ and $r + \infty = \infty$ for all $r \in \mathbf{R}$.

Now, let $G = \{\, g_1, g_2, g_3 \,\}$ be a set of goods the price of which (in euros) evolves over the years according to the pricing function $pr : G \times \mathbf{S} \to \mathbb{N}$ given in Table 1.

We can then define the affordability predicate on multisets of goods as follows:

$$\forall (r, s) \in \mathbf{R} \times \mathbf{S}. \quad (r, s) \Vdash Af(gs) \ \text{ iff } \ pr(gs, s) \overset{\text{def}}{=} \sum_{g \in gs} pr(g, s) \leqslant r$$

We write $x_1, \ldots, x_n$ as a shorthand for the multiset $\{\, x_1, \ldots, x_n \,\}$. Therefore, $Af(\{\, g, g' \,\})$ is more shortly written as $Af(g, g')$. It is easy to see that

$$\forall (r, s) \in \mathbf{R} \times \mathbf{S}. \forall g, g' \in G. \quad (r, s) \Vdash Af(g, g') \ \text{ iff } \ (r, s) \Vdash Af(g) * Af(g')$$

As an example, let us suppose that each year, we get an amount of money that we are required to spend buying goods on some dedicated website. LTBI allows us to state properties about our ability to buy goods depending on the year and on the amount of money available. For instance,

$$(3000, 2023) \Vdash Af(g_1) \wedge (Af(g_2) * Af(g_3))$$

Table 1 Prices of three goods over the years.

	Prices (€)		
good	**2023**	**2024**	**2025**
g_1	2000	2100	2200
g_2	300	250	350
g_3	1700	1800	1500

intuitively means that in 2023 (the current year), with 3000 euros, we can choose to buy g_1 and we can also choose to split our money into two disjoint amounts, the first one to buy g_2 and the second one to buy g_3. Let us remark that although the two options are available to us simultaneously, it does not necessarily imply that we could afford to buy all three goods simultaneously. Indeed, with an amount of 3000 euros, we would have to make a choice since $pr(\{ g_1, g_2, g_3 \}, 2023) = 4000$. Therefore, $(3000, 2023) \nVdash Af(g_1, g_2, g_3)$.

Using the temporal modalities, we can state more complex propositions that take into account the evolution of prices over the years. For instance,

$$(3000, 2023) \Vdash \Box Af(g_2) * (\Diamond Af(g_3) \wedge (Af(g_1) * \circ Af(g_2)))$$

states that in 2023, we can split 3000 euros into two disjoint amounts of money, the first one keeping g_2 affordable every year from 2023 until 2025, the second one bringing us two choices. The first choice ensures that g_3 should become affordable at least one year during between 2023 and 2025. The second choice tells us that we could split our second amount of money once again into two new disjoint amounts, one making g_3 affordable currently (in 2023), the other making g_2 affordable only one year later (in 2024).

3 An LTBI Labelled Tableau Calculus

The labelled tableau calculus for LTBI, called $\mathsf{T}_{\mathsf{LTBI}}$, is in the spirit of the ones for BI [7] and DBI [3] and relies on the introduction of labels and constraints. $\mathsf{T}_{\mathsf{LTBI}}$ deals with two kinds of labels, namely resource labels and state labels.

We shall see that the latter require a careful and specific treatment in order to keep them isomorphic to natural numbers.

3.1 Labels and Constraints

We define a set of state labels and constraints that deals with temporality in order to capture the notion of resource evolution.

▶ **Definition 7** (Resource labels and constraints). *The set L_r of* resource labels *is built from the countable set $\gamma_r = \{ \epsilon_{\mathrm{L}}, \mathrm{c}_1, \mathrm{c}_2, \dots \}$ of* resource constants *and label composition $\circ$ according to the grammar $X ::= \gamma_r \mid X \circ X$. A* resource constraint *is an expression of the form $\mathrm{x} \leqslant^{\mathrm{r}}_{\mathrm{L}} \mathrm{y}$, where x and y are resource labels.*

Label composition is interpreted as an associative and commutative operation on L_r that admits ϵ_{L} as its neutral element. We shall frequently write $\mathrm{x}\,\mathrm{y}$ instead of $\mathrm{x} \circ \mathrm{y}$ for convenience. We say that x is a sublabel of y iff there exists $\mathrm{z} \in L_r$ such that $\mathrm{x} \circ \mathrm{z} = \mathrm{y}$ and $E(\mathrm{x})$ denotes the set of sublabels of a label x.

▶ **Definition 8** (State labels and constraints). *The set L_s of* state labels *is built from the countable set $\gamma_s = \{\gamma_0, \gamma_1, \gamma_2, \dots\}$ of* state constants *and the successor symbol η according to the grammar $X ::= \gamma_s \mid \eta X$. Given two state labels τ and τ', a* state constraint *is an expression of the form $\tau \leqslant^{\mathfrak{s}}_{\mathrm{L}} \tau'$, $\tau <^{s}_{\mathrm{L}} \tau'$, $\tau =^{\mathfrak{s}}_{\mathrm{L}} \tau'$ or $\tau \neq^{\mathfrak{s}}_{\mathrm{L}} \tau'$.*

▶ **Definition 9** (Domain and alphabet). *Let C_r be a set of resource constraints. The* domain *of C_r, denoted $D_r(C_r)$, is the set of all the sublabels occurring in C_r. More formally, $D_r(C_r) = \bigcup_{\mathrm{x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{y} \in C_r} (E(\mathrm{x}) \cup E(\mathrm{y}))$. The* alphabet *(or* basis*) of C_r is the set $A_r(C_r) = \gamma_r \cap D_r(C_r)$. $D_s(C_s)$ and $A_s(C_s)$, where C_s is a set of state constraints, are defined similarly.*

▶ **Definition 10** (Closure of resource constraints). *Let C_r be a set of resource constraints, the closure $C_r^\bullet$ is the smallest set such that $C_r \subseteq C_r^\bullet$ that is closed under the following rules:*

$$\frac{\mathrm{x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{y} \qquad \mathrm{y} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{z}}{\mathrm{x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{z}} \quad \frac{\mathrm{x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{y}}{\mathrm{x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x}} \quad \frac{\mathrm{x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{y}}{\mathrm{y} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{y}} \quad \frac{\mathrm{x\,y} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x\,y}}{\mathrm{x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x}} \quad \frac{\mathrm{z\,y} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{z\,y} \qquad \mathrm{x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{y}}{\mathrm{z\,x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{z\,y}}$$

These rules reflect the properties of transitivity and reflexivity of $\leqslant^{\mathfrak{r}}_{\mathrm{L}}$ and the compatibility of $\circ$ w.r.t. $\leqslant^{\mathfrak{r}}_{\mathrm{L}}$. Since none of these rules introduce any new resource constant, we have $A_r(C_r) = A_r(C_r^\bullet)$.

▶ **Definition 11** (Closure of state constraints). *Let C_s be a set of state constraints, the closure $C_s^\bullet$ is the smallest set such that $C_s \subseteq C_s^\bullet$ that reflects in $\leqslant^{\mathfrak{s}}_{\mathrm{L}}, <^{s}_{\mathrm{L}}, =^{\mathfrak{s}}_{\mathrm{L}}, \neq^{\mathfrak{s}}_{\mathrm{L}}$ the properties of $\leqslant, <, =, \neq$ in $\mathbb{N}$ and such that η syntactically reflects the properties of the "next" function $\mathfrak{n}$.*

▶ **Proposition 12.** *Let C_r be a set of resource constraints:*

1. *If $\mathrm{z\,x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{y} \in C_r^\bullet$, then $\mathrm{x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x} \in C_r^\bullet$*
2. *If $\mathrm{x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{z\,y} \in C_r^\bullet$, then $\mathrm{y} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{y} \in C_r^\bullet$*

Proof. From $\mathrm{z\,x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{y}$ we get $\mathrm{z\,x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{z\,x}$ (reflexivity), then $\mathrm{x\,z} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x\,z}$ (commutativity) and then $\mathrm{x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x}$ (compatibility). The other case is similar. ◀

3.2 Rules of the $\mathsf{T}_{\mathsf{LTBI}}$ Tableau Calculus

▶ **Definition 13** (Labelled Formula). *A* labelled formula *is a quadruple $(\mathbb{S}, \mathrm{A}, \mathrm{x}, \tau)$, denoted $\mathbb{S}\ \mathrm{A} : (\mathrm{x}, \tau)$, where $\mathbb{S} \in \{\mathbb{T}, \mathbb{F}\}$ is a* sign*, $\mathrm{A} \in \mathbf{F}$ is a formula, and $(\mathrm{x}, \tau) \in L_r \times L_s$ is a* label.

▶ **Definition 14** (CTSS). *A* constrained temporal set of statements *(CTSS) is a triple noted $\langle \mathcal{F}, C_r, C_s \rangle$, where $\mathcal{F}$ is a set of labelled formulas, C_r is a set of resource constraints and C_s is a set of state constraints. A CTSS is required to satisfy the following condition:*

$(CTSS_R)$ *for all $\mathbb{S}\ \mathrm{A} : (\mathrm{x}, \tau) \in \mathcal{F}, \mathrm{x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x} \in C_r$ and $\tau \leqslant^{\mathfrak{s}}_{\mathrm{L}} \tau \in C_s$.*

A CTSS is finite if all of its three components are finite.

▶ **Definition 15** (Inconsistent Label). *Let $\langle \mathcal{F}, C_r, C_s \rangle$ be a CTSS. The label (x, τ) is* inconsistent *if there exist two resource labels y and z such that $\mathrm{y} \circ \mathrm{z} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x} \in C_r^\bullet$ and $\mathbb{T}\ \bot : (\mathrm{y}, \tau) \in \mathcal{F}$. A label is consistent if it is not inconsistent.*

▶ **Proposition 16.** *Let $\langle \mathcal{F}, C_r, C_s \rangle$ be a CTSS. The following properties hold:*

1. *If $\mathrm{y} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x} \in C_r^\bullet$ and (x, τ) is consistent, then (y, τ) is a consistent label.*
2. *If $\mathrm{x} \circ \mathrm{y} \in D_r(C_r^\bullet)$ and $(\mathrm{x} \circ \mathrm{y}, \tau)$ is consistent, then (x, τ) and (y, τ) are consistent.*

Proof. Assume that (y, τ) is inconsistent, then there are two resource labels z, z' and a state label τ such that $\mathrm{z} \circ \mathrm{z}' \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{y} \in C_r^\bullet$ and $\mathbb{T}\ \bot : (\mathrm{z}, \tau) \in \mathcal{F}$. By transitivity with $\mathrm{y} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x} \in C_r^\bullet$ we get $\mathrm{z} \circ \mathrm{z}' \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x} \in C_r^\bullet$, meaning that (x, τ) is inconsistent, which contradicts our assumption. The other proof is similar. ◀

$$\frac{\mathbb{T}\ \mathrm{A}\land\mathrm{B}:(\mathrm{x},\tau)}{\begin{array}{c}\mathbb{T}\ \mathrm{A}:(\mathrm{x},\tau)\\ \mathbb{T}\ \mathrm{B}:(\mathrm{x},\tau)\end{array}} \qquad \begin{array}{c}\mathbb{F}\ \mathrm{A}\land\mathrm{B}:(\mathrm{x},\tau)\\ \begin{array}{c|c}\mathbb{F}\ \mathrm{A}:(\mathrm{x},\tau) & \mathbb{F}\ \mathrm{B}:(\mathrm{x},\tau)\end{array}\end{array} \qquad \frac{\mathbb{F}\ \mathrm{A}\lor\mathrm{B}:(\mathrm{x},\tau)}{\begin{array}{c}\mathbb{F}\ \mathrm{A}:(\mathrm{x},\tau)\\ \mathbb{F}\ \mathrm{B}:(\mathrm{x},\tau)\end{array}}$$

$$\begin{array}{c}\mathbb{T}\ \mathrm{A}\lor\mathrm{B}:(\mathrm{x},\tau)\\ \begin{array}{c|c}\mathbb{T}\ \mathrm{A}:(\mathrm{x},\tau) & \mathbb{T}\ \mathrm{B}:(\mathrm{x},\tau)\end{array}\end{array} \qquad \begin{array}{c}\mathbb{T}\ \mathrm{A}\to\mathrm{B}:(\mathrm{x},\tau)\\ \begin{array}{c|c}\mathbb{R}\ \mathrm{x}\leqslant^{\mathfrak{r}}_{\mathrm{L}}\mathrm{y} & \mathbb{R}\ \mathrm{x}\leqslant^{\mathfrak{r}}_{\mathrm{L}}\mathrm{y}\\ \mathbb{F}\ \mathrm{A}:(\mathrm{y},\tau) & \mathbb{T}\ \mathrm{B}:(\mathrm{y},\tau)\end{array}\end{array} \qquad \frac{\mathbb{F}\ \mathrm{A}\to\mathrm{B}:(\mathrm{x},\tau)}{\begin{array}{c}\mathbb{A}\ \mathrm{x}\leqslant^{\mathfrak{r}}_{\mathrm{L}}\mathrm{a}\\ \mathbb{T}\ \mathrm{A}:(\mathrm{a},\tau)\\ \mathbb{F}\ \mathrm{B}:(\mathrm{a},\tau)\end{array}}$$

$$\frac{\mathbb{T}\ \mathrm{I}:(\mathrm{x},\tau)}{\mathbb{A}\ \epsilon_{\mathrm{L}}\leqslant^{\mathfrak{r}}_{\mathrm{L}}\mathrm{x}} \qquad \frac{\mathbb{T}\ \mathrm{A}*\mathrm{B}:(\mathrm{x},\tau)}{\begin{array}{c}\mathbb{A}\ \mathrm{ab}\leqslant^{\mathfrak{r}}_{\mathrm{L}}\mathrm{x}\\ \mathbb{T}\ \mathrm{A}:(\mathrm{a},\tau)\\ \mathbb{T}\ \mathrm{B}:(\mathrm{b},\tau)\end{array}} \qquad \begin{array}{c}\mathbb{F}\ \mathrm{A}*\mathrm{B}:(\mathrm{x},\tau)\\ \begin{array}{c|c}\mathbb{R}\ \mathrm{y\,z}\leqslant^{\mathfrak{r}}_{\mathrm{L}}\mathrm{x} & \mathbb{R}\ \mathrm{y\,z}\leqslant^{\mathfrak{r}}_{\mathrm{L}}\mathrm{x}\\ \mathbb{F}\ \mathrm{A}:(\mathrm{y},\tau) & \mathbb{F}\ \mathrm{B}:(\mathrm{z},\tau)\end{array}\end{array} \qquad \frac{\mathbb{F}\ \mathrm{A}\mathrel{-\!\!*}\mathrm{B}:(\mathrm{x},\tau)}{\begin{array}{c}\mathbb{A}\ \mathrm{x\,a}\leqslant^{\mathfrak{r}}_{\mathrm{L}}\mathrm{b}\\ \mathbb{T}\ \mathrm{A}:(\mathrm{a},\tau)\\ \mathbb{F}\ \mathrm{B}:(\mathrm{b},\tau)\end{array}}$$

$$\begin{array}{c}\mathbb{T}\ \mathrm{A}\mathrel{-\!\!*}\mathrm{B}:(\mathrm{x},\tau)\\ \begin{array}{c|c}\mathbb{R}\ \mathrm{x\,y}\leqslant^{\mathfrak{r}}_{\mathrm{L}}\mathrm{z} & \mathbb{R}\ \mathrm{x\,y}\leqslant^{\mathfrak{r}}_{\mathrm{L}}\mathrm{z}\\ \mathbb{F}\ \mathrm{A}:(\mathrm{y},\tau) & \mathbb{T}\ \mathrm{B}:(\mathrm{z},\tau)\end{array}\end{array} \qquad \frac{\mathbb{T}\ \circ\mathrm{A}:(\mathrm{x},\tau)}{\begin{array}{c}\mathbb{A}\ \tau<^{s}_{\mathrm{L}}\eta\tau\\ \mathbb{T}\ \mathrm{A}:(\mathrm{x},\eta\tau)\end{array}} \qquad \frac{\mathbb{F}\ \circ\mathrm{A}:(\mathrm{x},\tau)}{\begin{array}{c}\mathbb{R}\ \tau<^{s}_{\mathrm{L}}\eta\tau\\ \mathbb{F}\ \mathrm{A}:(\mathrm{x},\eta\tau)\end{array}}$$

$$\frac{\mathbb{T}\ \Box\mathrm{A}:(\mathrm{x},\tau)}{\begin{array}{c}\mathbb{R}\ \tau\leqslant^{\mathfrak{s}}_{\mathrm{L}}\alpha\\ \mathbb{T}\ \mathrm{A}:(\mathrm{x},\alpha)\end{array}} \qquad \frac{\mathbb{F}\ \Box\mathrm{A}:(\mathrm{x},\tau)}{\begin{array}{c}\mathbb{A}\ \tau\leqslant^{\mathfrak{s}}_{\mathrm{L}}\upsilon\\ \mathbb{F}\ \mathrm{A}:(\mathrm{x},\upsilon)\end{array}} \qquad \frac{\mathbb{T}\ \Diamond\mathrm{A}:(\mathrm{x},\tau)}{\begin{array}{c}\mathbb{A}\ \tau\leqslant^{\mathfrak{s}}_{\mathrm{L}}\alpha\\ \mathbb{T}\ \mathrm{A}:(\mathrm{x},\alpha)\end{array}} \qquad \frac{\mathbb{F}\ \Diamond\mathrm{A}:(\mathrm{x},\tau)}{\begin{array}{c}\mathbb{R}\ \tau\leqslant^{\mathfrak{s}}_{\mathrm{L}}\upsilon\\ \mathbb{F}\ \mathrm{A}:(\mathrm{x},\upsilon)\end{array}}$$

$$\begin{array}{c}\mathrm{CD}\\ \begin{array}{c|c}\mathbb{R}\ \tau\leqslant^{\mathfrak{s}}_{\mathrm{L}}\upsilon & \mathbb{R}\ \tau\leqslant^{\mathfrak{s}}_{\mathrm{L}}\upsilon\\ \mathbb{A}\ \tau<^{s}_{\mathrm{L}}\upsilon & \mathbb{A}\ \tau=^{\mathfrak{s}}_{\mathrm{L}}\upsilon\end{array}\end{array} \qquad \begin{array}{c}\mathrm{LR}\\ \begin{array}{c|c}\mathbb{R}\ \tau\leqslant^{\mathfrak{s}}_{\mathrm{L}}\upsilon & \mathbb{R}\ \tau\leqslant^{\mathfrak{s}}_{\mathrm{L}}\upsilon\\ \mathbb{R}\ \tau\leqslant^{\mathfrak{s}}_{\mathrm{L}}\zeta & \mathbb{R}\ \tau\leqslant^{\mathfrak{s}}_{\mathrm{L}}\zeta\\ \mathbb{A}\ \upsilon\leqslant^{\mathfrak{s}}_{\mathrm{L}}\zeta & \mathbb{A}\ \zeta\leqslant^{\mathfrak{s}}_{\mathrm{L}}\upsilon\end{array}\end{array} \qquad \frac{\mathbb{S}\ \mathrm{A}:(\mathrm{c},\tau)}{\begin{array}{c}\mathbb{R}\ \tau=^{\mathfrak{s}}_{\mathrm{L}}\upsilon\\ \mathbb{S}\ \mathrm{A}:(\mathrm{c},\upsilon)\end{array}}$$

Figure 1 Rules of the $\mathsf{T}_{\mathsf{LTBI}}$ calculus.

The rules of $\mathsf{T}_{\mathsf{LTBI}}$ are presented in Figure 1, where a,b denote fresh resource constants and α denotes a fresh state constant. We observe that some of the rules introduce fresh constants and label constraints called *assertions*. For instance, expanding a labelled formula $\mathbb{F}\ \mathrm{A}\to\mathrm{B}:(\mathrm{x},\tau)$ generates a (resource) assertion $\mathbb{A}\ \mathrm{x}\leqslant^{\mathfrak{r}}_{\mathrm{L}}\mathrm{a}$ where a is a fresh resource constant. Similarly, expanding a labelled formula $\mathbb{T}\ \Diamond\mathrm{A}:(\mathrm{x},\tau)$ generates a (state) assertion $\mathbb{A}\ \tau\leqslant^{\mathfrak{s}}_{\mathrm{L}}\alpha$ where α is a fresh state constant. We also observe that some of the rules introduce label constraints on arbitrary labels called *requirements*. For instance, expanding a labelled formula $\mathbb{T}\ \mathrm{A}\to\mathrm{B}:(\mathrm{x},\tau)$ generates a (resource) requirement $\mathbb{R}\ \mathrm{x}\leqslant^{\mathfrak{r}}_{\mathrm{L}}\mathrm{y}$. Similarly, expanding a labelled formula $\mathbb{F}\ \Diamond\mathrm{A}:(\mathrm{x},\tau)$ generates a (state) requirement $\mathbb{R}\ \tau\leqslant^{\mathfrak{s}}_{\mathrm{L}}\upsilon$.

Before we explain how requirements work, let us note that a tableau branch $\mathcal{B}$ corresponds to a CTSS $\langle\mathcal{F},C_r,C_s\rangle$, where $\mathcal{F}$ is the set of all labelled formulas occurring in $\mathcal{B}$ and C_r, C_s are the sets of all resource and state assertions occurring in $\mathcal{B}$ respectively, i.e. $C_r=\{\mathbb{A}\ \mathrm{x}\leqslant^{\mathfrak{r}}_{\mathrm{L}}\mathrm{y}\mid\mathbb{A}\ \mathrm{x}\leqslant^{\mathfrak{r}}_{\mathrm{L}}\mathrm{y}\in\mathcal{B}\}$ and $C_s=\{\mathbb{A}\ \mathrm{x}\ \mathrm{R}^{s}_{\mathrm{L}}\ \mathrm{y}\mid\mathbb{A}\ \mathrm{x}\ \mathrm{R}^{s}_{\mathrm{L}}\ \mathrm{y}\in\mathcal{B}\}$ for $\mathrm{R}^{s}_{\mathrm{L}}\in\{\leqslant^{\mathfrak{s}}_{\mathrm{L}},<^{s}_{\mathrm{L}},=^{\mathfrak{s}}_{\mathrm{L}},\neq^{\mathfrak{s}}_{\mathrm{L}}\}$. Now if we want to expand a labelled formula $\mathbb{T}\ \mathrm{A}\to\mathrm{B}:(\mathrm{x},\tau)$ occurring in $\mathcal{B}$, the label constraint $\mathbb{R}\ \mathrm{x}\leqslant^{\mathfrak{r}}_{\mathrm{L}}\mathrm{y}$ requires us to find a label y such that $\mathrm{x}\leqslant^{\mathfrak{r}}_{\mathrm{L}}\mathrm{y}\in C_r^{\bullet}$, i.e., a label y for which the requirement is derivable from (the closure of) the assertions that already occur in the branch.

The last line of Figure 1 presents the structural rules of $\mathsf{T}_{\mathsf{LTBI}}$. The first one is the case distinction rule CD that disambiguates any label state constraint $\tau\leqslant^{\mathfrak{s}}_{\mathrm{L}}\upsilon$ derivable from the closure of the state assertions (hence the requirement $\mathbb{R}\ \tau\leqslant^{\mathfrak{s}}_{\mathrm{L}}\upsilon$) w.r.t. $<^{s}_{\mathrm{L}}$ and $=^{\mathfrak{s}}_{\mathrm{L}}$. The second one is the linearizing rule LR that arranges any pair of state labels υ and ζ branching from τ into a linear order $\tau \leqslant^{\mathfrak{s}}_{\mathrm{L}} \upsilon \leqslant^{\mathfrak{s}}_{\mathrm{L}} \zeta$ or $\tau \leqslant^{\mathfrak{s}}_{\mathrm{L}} \zeta \leqslant^{\mathfrak{s}}_{\mathrm{L}} \upsilon$. The last one is the equality rewriting rule which is there mostly for convenience to make the closing of a branch easier to check.

▶ **Definition 17.** *A tableau for a formula* A *is a tableau built inductively according to the rules depicted in Figure 1 the root node of which is the labelled formula* $\mathbb{F}\ \mathrm{A} : (\epsilon_{\mathrm{L}}, \gamma_0)$.

Definition 17 implies that a $\mathsf{T}_{\mathsf{LTBI}}$ tableau for a LTBI formula A begins with the initial CTSS $\langle \mathbb{F}\ \mathrm{A} : (\epsilon_{\mathrm{L}}, \gamma_0), \{\, \epsilon_{\mathrm{L}} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \epsilon_{\mathrm{L}} \,\}, \{\, \gamma_0 \leqslant^{\mathfrak{s}}_{\mathrm{L}} \gamma_0 \,\}\rangle$. Moreover, we define a rule application strategy according to the following order of precedence from highest to lowest:

1. The rules $\mathbb{T}\,\mathrm{I}$, $\mathbb{F} \rightarrow$, $\mathbb{T} *$, $\mathbb{F} \mathrel{-\!\!*}$, $\mathbb{T} \Diamond$, $\mathbb{F} \Box$, $\mathbb{T} \circ$ and $\mathbb{F} \circ$, called π_α-rules, take precedence over the other rules.
2. The structural rules CD and LR have middle precedence.
3. The rules $\mathbb{T} \rightarrow$, $\mathbb{F} *$, $\mathbb{T} \mathrel{-\!\!*}$, $\mathbb{F} \Diamond$, $\mathbb{T} \Box$, called π_β-rules, have low precedence.

▶ **Definition 18** (Closing conditions)**.** *A CTSS* $\langle \mathcal{F}, C_r, C_s\rangle$ *is* closed *if it satisfies one of the following conditions:*

1. $\mathbb{T}\ \mathrm{A} : (\mathrm{x}, \tau) \in \mathcal{F}$, $\mathbb{F}\ \mathrm{A} : (\mathrm{y}, \upsilon) \in \mathcal{F}$, $\mathrm{x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{y} \in C_r^{\bullet}$ *and* $\tau =^{\mathfrak{s}}_{\mathrm{L}} \upsilon \in C_s^{\bullet}$.
2. $\mathbb{F}\ \mathrm{I} : (\mathrm{x}, \tau) \in \mathcal{F}$ *and* $\epsilon_{\mathrm{L}} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x} \in C_r^{\bullet}$
3. $\mathbb{F}\ \top : (\mathrm{x}, \tau) \in \mathcal{F}$
4. $\mathbb{F}\ \mathrm{A} : (\mathrm{x}, \tau) \in \mathcal{F}$ *and* (x, τ) *is inconsistent*
5. $\tau =^{\mathfrak{s}}_{\mathrm{L}} \upsilon \in C_s^{\bullet}$ *and* $\tau \neq^{\mathfrak{s}}_{\mathrm{L}} \upsilon \in C_s^{\bullet}$.

A tableau branch is closed if its corresponding CTSS is closed. A CTSS, or a tableau branch, is open if it is not closed. A tableau is closed if all of its branches are closed.

▶ **Definition 19** ($\mathsf{T}_{\mathsf{LTBI}}$-proof)**.** *A* $\mathsf{T}_{\mathsf{LTBI}}$*-proof for a formula* A *is a closed* $\mathsf{T}_{\mathsf{LTBI}}$ *tableau for* A.

▶ **Example 20.** Let us now illustrate in Figure 2 the construction of a $\mathsf{T}_{\mathsf{LTBI}}$ tableau with an example leading to a closed tableau.

We start with $\mathbb{F}\ \Diamond\mathrm{A} \wedge \Diamond\mathrm{B} \rightarrow \Diamond(\mathrm{A} \wedge \Diamond\mathrm{B}) \vee \Diamond(\mathrm{B} \wedge \Diamond\mathrm{A}) : (\epsilon_{\mathrm{L}}, \gamma_0)$. In Step [2], expanding $\mathbb{T}\ \Diamond\mathrm{A} \wedge \Diamond\mathrm{B} : (\mathrm{c}_1, \gamma_0)$ introduces $\mathbb{T}\ \Diamond\mathrm{A} : (\mathrm{c}_1, \gamma_0)$ and $\mathbb{T}\ \Diamond\mathrm{B} : (\mathrm{c}_1, \gamma_0)$. After Steps [3, 4], we obtain two assertions $\mathbb{A}\ \gamma_0 \leqslant^{\mathfrak{s}}_{\mathrm{L}} \gamma_1$ and $\mathbb{A}\ \gamma_0 \leqslant^{\mathfrak{s}}_{\mathrm{L}} \gamma_1$. In Step [5] we expand the signed formula $\mathbb{F}\ \Diamond(\mathrm{A} \wedge \Diamond\mathrm{B}) \vee \Diamond(\mathrm{B} \wedge \Diamond\mathrm{A}) : (\mathrm{c}_1, \gamma_0)$ and then generate $\mathbb{F}\ \Diamond(\mathrm{A} \wedge \Diamond\mathrm{B}) : (\mathrm{c}_1, \gamma_0)$ and $\mathbb{F}\ \Diamond(\mathrm{B} \wedge \Diamond\mathrm{A}) : (\mathrm{c}_1, \gamma_0)$.

Before expanding them, we apply the linearizing rule LR in Step [6] and the tableau splits into two branches: the left one with the assertion $\mathbb{A}\ \gamma_1 \leqslant^{\mathfrak{s}}_{\mathrm{L}} \gamma_2$ and the right one with the assertion $\mathbb{A}\ \gamma_2 \leqslant^{\mathfrak{s}}_{\mathrm{L}} \gamma_1$. Now we consider Step [7] in the left branch (with assertion $\mathbb{A}\ \gamma_1 \leqslant^{\mathfrak{s}}_{\mathrm{L}} \gamma_2$) that corresponds to the expansion of $\mathbb{F}\ \Diamond(\mathrm{A} \wedge \Diamond\mathrm{B}) : (\mathrm{c}_1, \gamma_0)$ introducing a requirement $\mathbb{R}\ \gamma_0 \leqslant^{\mathfrak{s}}_{\mathrm{L}} \mathrm{v}_1$ and the labelled formula $\mathbb{F}\ \mathrm{A} \wedge \Diamond\mathrm{B} : (\mathrm{c}_1, \mathrm{v}_1)$ with v_1 a variable to be instantiated from the closure of the assertions in the branch. Here we choose $\mathrm{v}_1 = \gamma_1$ in order to satisfy the requirement.

Then, in Step [8] $\mathbb{F}\ \mathrm{A} \wedge \Diamond\mathrm{B} : (\mathrm{c}_1, \gamma_1)$ splits the leftmost branch into two sub-branches. The first one is closed because it contains both $\mathbb{T}\ \mathrm{A} : (\mathrm{c}_1, \gamma_1)$, and $\mathbb{F}\ \Diamond\mathrm{B} : (\mathrm{c}_1, \gamma_1)$. The second one continues with Step [9] that introduces a requirement $\mathbb{R}\ \gamma_1 \leqslant^{\mathfrak{s}}_{\mathrm{L}} \mathrm{v}_2$ and the labelled formula $\mathbb{F}\ \mathrm{B} : (\mathrm{c}_1, \mathrm{v}_2)$ with v_2 a variable to be instantiated from the closure of the assertions in the branch. Here we choose $\mathrm{v}_2 = \gamma_2$ that satisfies the requirement because $\gamma_1 \leqslant^{\mathfrak{s}}_{\mathrm{L}} \gamma_2$. Then we obtain the labelled formula $\mathbb{F}\ \mathrm{B} : (\mathrm{c}_1, \gamma_2)$ and the branch is closed because it also contains $\mathbb{T}\ \mathrm{B} : (\mathrm{c}_1, \gamma_2)$. The tableau on the right(hand side of Step [6] similarly leads to closed branches.

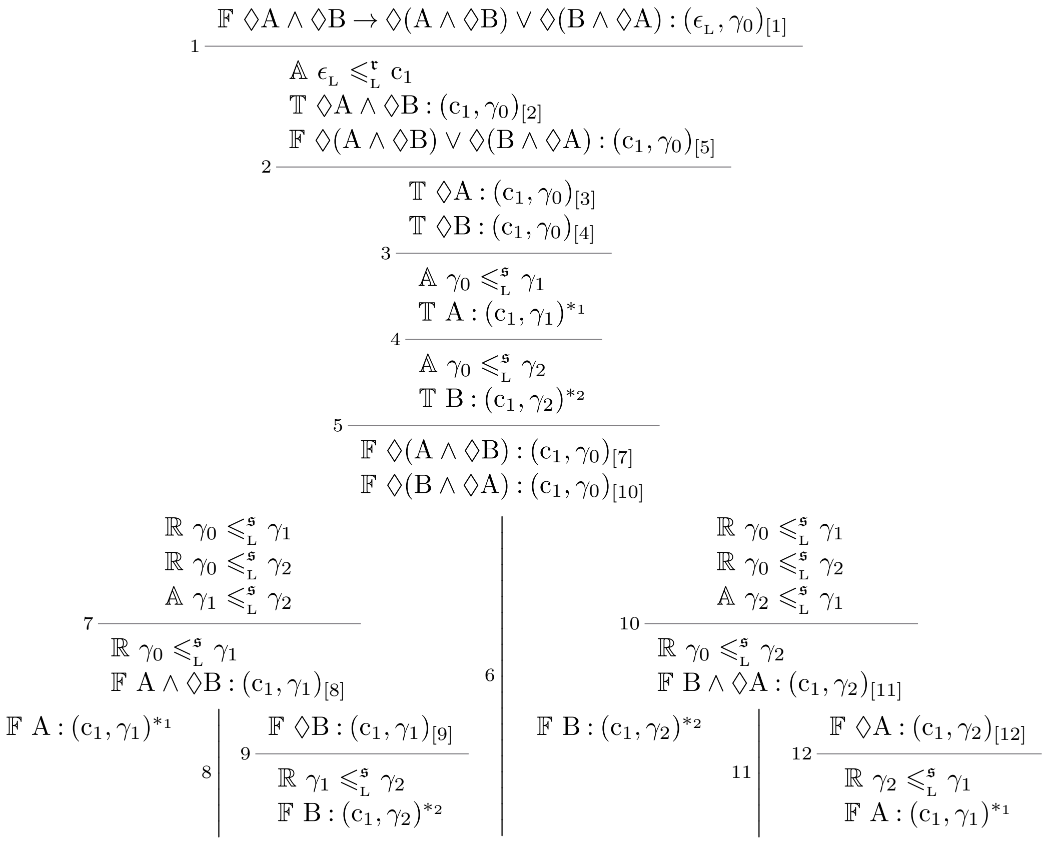

$\mathbb{F}\ \Diamond A \wedge \Diamond B \to \Diamond(A \wedge \Diamond B) \vee \Diamond(B \wedge \Diamond A) : (\epsilon_L, \gamma_0)_{[1]}$
1
$\mathbb{A}\ \epsilon_L \leqslant^{\mathfrak{r}}_{L} c_1$
$\mathbb{T}\ \Diamond A \wedge \Diamond B : (c_1, \gamma_0)_{[2]}$
$\mathbb{F}\ \Diamond(A \wedge \Diamond B) \vee \Diamond(B \wedge \Diamond A) : (c_1, \gamma_0)_{[5]}$
2
$\mathbb{T}\ \Diamond A : (c_1, \gamma_0)_{[3]}$
$\mathbb{T}\ \Diamond B : (c_1, \gamma_0)_{[4]}$
3
$\mathbb{A}\ \gamma_0 \leqslant^{\mathfrak{s}}_{L} \gamma_1$
$\mathbb{T}\ A : (c_1, \gamma_1)^{*_1}$
4
$\mathbb{A}\ \gamma_0 \leqslant^{\mathfrak{s}}_{L} \gamma_2$
$\mathbb{T}\ B : (c_1, \gamma_2)^{*_2}$
5
$\mathbb{F}\ \Diamond(A \wedge \Diamond B) : (c_1, \gamma_0)_{[7]}$
$\mathbb{F}\ \Diamond(B \wedge \Diamond A) : (c_1, \gamma_0)_{[10]}$
$\mathbb{R}\ \gamma_0 \leqslant^{\mathfrak{s}}_{L} \gamma_1$
$\mathbb{R}\ \gamma_0 \leqslant^{\mathfrak{s}}_{L} \gamma_2$
$\mathbb{A}\ \gamma_1 \leqslant^{\mathfrak{s}}_{L} \gamma_2$
7
$\mathbb{R}\ \gamma_0 \leqslant^{\mathfrak{s}}_{L} \gamma_1$
$\mathbb{F}\ A \wedge \Diamond B : (c_1, \gamma_1)_{[8]}$
$\mathbb{F}\ A : (c_1, \gamma_1)^{*_1}$
8
$\mathbb{F}\ \Diamond B : (c_1, \gamma_1)_{[9]}$
9
$\mathbb{R}\ \gamma_1 \leqslant^{\mathfrak{s}}_{L} \gamma_2$
$\mathbb{F}\ B : (c_1, \gamma_2)^{*_2}$
6
$\mathbb{R}\ \gamma_0 \leqslant^{\mathfrak{s}}_{L} \gamma_1$
$\mathbb{R}\ \gamma_0 \leqslant^{\mathfrak{s}}_{L} \gamma_2$
$\mathbb{A}\ \gamma_2 \leqslant^{\mathfrak{s}}_{L} \gamma_1$
10
$\mathbb{R}\ \gamma_0 \leqslant^{\mathfrak{s}}_{L} \gamma_2$
$\mathbb{F}\ B \wedge \Diamond A : (c_1, \gamma_2)_{[11]}$
$\mathbb{F}\ B : (c_1, \gamma_2)^{*_2}$
11
$\mathbb{F}\ \Diamond A : (c_1, \gamma_2)_{[12]}$
12
$\mathbb{R}\ \gamma_2 \leqslant^{\mathfrak{s}}_{L} \gamma_1$
$\mathbb{F}\ A : (c_1, \gamma_1)^{*_1}$

Figure 2 Closed Tableau for $\Diamond A \wedge \Diamond B \to \Diamond(A \wedge \Diamond B) \vee \Diamond(B \wedge \Diamond A)$.

▶ **Example 21.** Let us now illustrate in Figure 3 the construction of a $\mathsf{T}_{\mathsf{LTBI}}$ tableau with an example leading to a non closed tableau.

We start with $\mathbb{F}\ (\Diamond A * \circ B) \to (\Diamond B * \circ A) : (\epsilon_L, \gamma_0)$. Then, Step [2], $\mathbb{T}\ \Diamond A * \circ B : (c_1, \gamma_0)$ introduces the assertion $\mathbb{A}\ c_2\, c_3 \leqslant^{\mathfrak{r}}_{L} c_1$ and to the labelled formulae $\mathbb{T}\ \Diamond A : (c_2, \gamma_0)$ and $\mathbb{T}\ \circ B : (c_3, \gamma_0)$. In Step [3] we expand the first one and generate an assertion $\mathbb{A}\ \gamma_0 \leqslant^{\mathfrak{s}}_{L} \gamma_1$ and the labelled formula $\mathbb{T}\ A : (c_2, \gamma_1)$. In Step [4] we expand the second one and generate the labelled formula $\mathbb{T}\ B : (c_3, \eta\gamma_0)$. Step [5] deals with the labelled formula $\mathbb{F}\ \Diamond B * \circ A : (c_1, \gamma_0)$ and its expansion rules creates two branches: the left one with the requirement $\mathbb{R}\ y\, z \leqslant^{\mathfrak{r}}_{L} c_1$ and the labelled formula $\mathbb{F}\ \Diamond B : (c_1, \gamma_0)$ and the right one with the requirement $\mathbb{R}\ y\, z \leqslant^{\mathfrak{r}}_{L} c_1$ and the labelled formula $\mathbb{F}\ \circ A : (z, \gamma_0)$.

Let us consider the left branch. The requirement $\mathbb{R}\ y\, z \leqslant^{\mathfrak{r}}_{L} c_1$ can only be satisfied in two cases: (1) $y = c_3$, $z = c_2$ and (2) $y = c_2$, $z = c_3$. Step [6] in the left branch corresponds to the expansion of $\mathbb{F}\ \Diamond B : (y, \gamma_0)$. It generates the requirement $\mathbb{R}\ \gamma_0 \leqslant^{\mathfrak{s}}_{L} v$ and the labelled formula $\mathbb{F}\ B : (y, v)$. In order to be able to close the branch with $\mathbb{T}\ B : (c_3, \eta\gamma_0)$ we have to set $y = c_3$ (with $z = c_2$) and to instantiate the variable v such that $\gamma_0 \leqslant^{\mathfrak{s}}_{L} v$. If we instantiate v with $\eta\gamma_0$ we satisfy the requirement $\mathbb{R}\ \gamma_0 \leqslant^{\mathfrak{s}}_{L} v$ and then the branch is closed.

Let us consider the right branch branch in which the requirement $\mathbb{R}\ yz \leqslant^{\mathfrak{r}}_{L} c_1$ is satisfied with $y = c_3, z = c_2$. Step [7] in the left branch corresponds to the expansion of $\mathbb{F}\ \circ A : (c_2, \gamma_0)$ that generates the labelled formula $\mathbb{F}\ A : (c_2, \eta\gamma_0)$. We observe that we cannot close this branch with the latter labelled formula and $\mathbb{T}\ A : (c_2, \gamma_1)$ because there is no possible equality between γ_1 and $\eta\gamma_0$. Then in case (1) there is an open branch and the tableau is not closed. Developing case (2) also leads to an open branch.

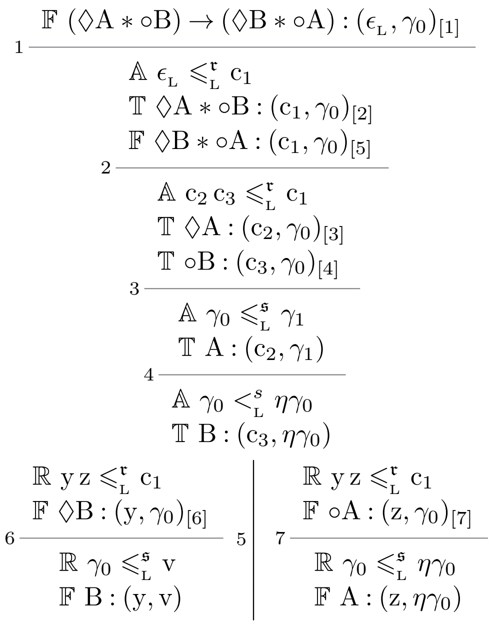

$\mathbb{F}\ (\Diamond A * \circ B) \to (\Diamond B * \circ A) : (\epsilon_{\mathrm{L}}, \gamma_0)_{[1]}$
1
$\mathbb{A}\ \epsilon_{\mathrm{L}} \leqslant^{\mathfrak{r}}_{\mathrm{L}} c_1$
$\mathbb{T}\ \Diamond A * \circ B : (c_1, \gamma_0)_{[2]}$
$\mathbb{F}\ \Diamond B * \circ A : (c_1, \gamma_0)_{[5]}$
2
$\mathbb{A}\ c_2 c_3 \leqslant^{\mathfrak{r}}_{\mathrm{L}} c_1$
$\mathbb{T}\ \Diamond A : (c_2, \gamma_0)_{[3]}$
$\mathbb{T}\ \circ B : (c_3, \gamma_0)_{[4]}$
3
$\mathbb{A}\ \gamma_0 \leqslant^{\mathfrak{s}}_{\mathrm{L}} \gamma_1$
$\mathbb{T}\ A : (c_2, \gamma_1)$
4
$\mathbb{A}\ \gamma_0 <^{s}_{\mathrm{L}} \eta\gamma_0$
$\mathbb{T}\ B : (c_3, \eta\gamma_0)$
$\mathbb{R}\ y z \leqslant^{\mathfrak{r}}_{\mathrm{L}} c_1$
$\mathbb{F}\ \Diamond B : (y, \gamma_0)_{[6]}$
6
$\mathbb{R}\ \gamma_0 \leqslant^{\mathfrak{s}}_{\mathrm{L}} v$
$\mathbb{F}\ B : (y, v)$
5
$\mathbb{R}\ y z \leqslant^{\mathfrak{r}}_{\mathrm{L}} c_1$
$\mathbb{F}\ \circ A : (z, \gamma_0)_{[7]}$
7
$\mathbb{R}\ \gamma_0 \leqslant^{\mathfrak{s}}_{\mathrm{L}} \eta\gamma_0$
$\mathbb{F}\ A : (z, \eta\gamma_0)$

Figure 3 Non-closed Tableau for $(\Diamond A * \circ B) \to (\Diamond B * \circ A)$.

4 Soundness of $\mathsf{T}_{\mathsf{LTBI}}$

In this section, we prove the soundness of $\mathsf{T}_{\mathsf{LTBI}}$ following a method based on the notion of *realizability* of a CTSS that is similar to the one used for various flavours of BI [5].

▶ **Definition 22** (Realization). *A* realization *of a CTSS $\langle \mathcal{F}, C_r, C_s \rangle$ is a triple $(\mathcal{M}, [.]_r, [.]_s)$, where $\mathcal{M}$ is an* LTBI*-model, and $[.]_r$, $[.]_s$ are order preserving homomorphisms from resource and state labels to resources and states respectively. More precisely, we have $[.]_r : D_r(C_r^\bullet) \to \mathbf{R}$ and $[.]_s : D_s(C_s^\bullet) \to \mathbf{S}$, such that:*

- $[\epsilon_{\mathrm{L}}]_r = \epsilon$, $[x \circ y]_r = [x]_r \star [y]_r$, $[\eta\tau]_s = \mathfrak{n}[\tau]_s$
- *If* $\mathbb{T}\ A : (x, \tau) \in \mathcal{F}$*, then* $([x]_r, [\tau]_s) \Vdash A$
- *If* $\mathbb{F}\ A : (x, \tau) \in \mathcal{F}$*, then* $([x]_r, [\tau]_s) \nVdash A$
- *If* $x \leqslant^{\mathfrak{r}}_{\mathrm{L}} y \in C_r$*, then* $[x]_r \leqslant^{\mathfrak{r}} [y]_r$
- *If* $\tau\ \mathrm{R}^{s}_{\mathrm{L}}\ \upsilon \in C_s$*, then* $[\tau]_s\ \mathrm{R}^s\ [\upsilon]_s$*, with* $\mathrm{R}^s \in \{ \leqslant^{\mathfrak{s}}, <^{s}, =^{\mathfrak{s}}, \neq^{\mathfrak{s}} \}$

A CTSS (or branch) is *realizable* if it has a realization. A tableau is *realizable* if it has at least one realizable branch.

▶ **Lemma 23.** *Let $(\mathcal{M}, [.]_r, [.]_s)$ be a realization of a CTSS $\langle \mathcal{F}, C_r, C_s \rangle$. For all $x \leqslant^{\mathfrak{r}}_{\mathrm{L}} y \in C_r^\bullet$ and for all $\tau\ \mathrm{R}^{s}_{\mathrm{L}}\ \upsilon \in C_s^\bullet$, $[x]_r \leqslant^{\mathfrak{r}} [y]_r$ and $[\tau]_s\ \mathrm{R}^s\ [\upsilon]_s$.*

Proof. Straightforward since the closure rules for C_r and C_s preserve compatibility. ◀

▶ **Lemma 24.** *If a $\mathsf{T}_{\mathsf{LTBI}}$ tableau is closed then it is not realizable.*

Proof. If a closed tableau is realizable then it contains at least one branch $\mathcal{B}$ that is realizable in a LTBI-model.

- If the branch is closed with complementary formulas $\mathbb{T}\ A : (x, \tau)$ and $\mathbb{F}\ A : (y, \tau)$ then by Definition 22 we have $x \leqslant^{\mathfrak{r}}_{\mathrm{L}} y$. By Lemma 23, we have $[x]_r \leqslant^{\mathfrak{r}} [y]_r$ and since the branch is realized, by Definition 22, we have $([x]_r, [\tau]_s) \Vdash A$ and $([y]_r, [\tau]_s) \nVdash A$. We thus reach a contradiction since by Lemma 6 (monotonicity) $([y]_r, [\tau]_s) \Vdash A$.

- if the branch is closed because of $\mathbb{F}\ \top : (\mathrm{x}, \tau)$, then $([\mathrm{x}]_r, [\tau]_s) \nVdash \top$, which is a contradiction.
- The other cases are similar. ◀

▶ **Lemma 25.** *All* $\mathsf{T}_{\mathsf{LTBI}}$ *rules preserve realizability.*

Proof. Let $\mathcal{B}$ be a tableau branch and $(\mathcal{M}, [.]_r, [.]_s)$ be a realization of its CTSS $\langle \mathcal{F}, C_r, C_s \rangle$. We proceed by case analysis on the rule that expands $\mathcal{B}$.

The cases for BI connectives are similar to the ones given in [7] for BI tableaux. We thus only consider the modal operators.

- Case $\mathbb{T} \circ$:
  Suppose that the labelled formula $\mathbb{T}\ \circ\mathrm{A} : (\mathrm{x}, \tau)$ has just been expanded in the branch $\mathcal{B}$. Then, $\mathcal{B}$ is extended with a new labelled formula $\mathbb{T}\ \mathrm{A}{:}(\mathrm{x}, \eta\tau)$ and a new assertion $\mathbb{A}\ \tau <^s_{\mathrm{L}} \eta\tau$. Since $\mathcal{B}$ was realizable before the expansion, we have $([\mathrm{x}]_r, [\tau]_s) \Vdash \circ\mathrm{A}$. Therefore, there exists s' such that $s' = \mathfrak{n}[\tau]_s$ and $([\mathrm{x}]_r, s') \Vdash \mathrm{A}$. Since $\mathfrak{n}[\tau]_s = [\eta\tau]_s$ and $[\tau]_s <^s [\eta\tau]_s$, both $\mathbb{T}\ \mathrm{A} : (\mathrm{x}, \eta\tau)$ and $\mathbb{A}\ \tau <^s_{\mathrm{L}} \eta\tau$ are realized.
- Case $\mathbb{F} \circ$:
  Suppose that the labelled formula $\mathbb{F}\ \circ\mathrm{A} : (\mathrm{x}, \tau)$ has just been expanded in the branch $\mathcal{B}$. Then, $\mathcal{B}$ is extended with a new labelled formula $\mathbb{F}\ \mathrm{A} : (\mathrm{x}, \eta\tau)$ and a new requirement $\mathbb{R}\ \tau <^s_{\mathrm{L}} \eta\tau$. A valid application of the expansion rule requires that $\tau <^s_{\mathrm{L}} \eta\tau \in C_s^{\bullet}$. Since $\mathcal{B}$ was realizable before the expansion, we have $([\mathrm{x}]_r, [\tau]_s) \nVdash \circ\mathrm{A}$ and Lemma 23 entails $[\tau]_s <^s [\eta\tau]_s$. Since $\mathfrak{n}[\tau]_s = [\eta\tau]_s$, $([\mathrm{x}]_r, [\tau]_s) \nVdash \circ\mathrm{A}$ implies $([\mathrm{x}]_r, [\eta\tau]_s) \nVdash \mathrm{A}$ by definition. Therefore, both $\mathbb{F}\ \mathrm{A} : (\mathrm{x}, \eta\tau)$ and $\mathbb{R}\ \tau <^s_{\mathrm{L}} \eta\tau$ are realized.
- The other cases are similar. ◀

▶ **Theorem 26** (Soundness)**.** *If there exists a* $\mathsf{T}_{\mathsf{LTBI}}$ *proof for* A*, then* A *is valid.*

Proof. Let $\mathcal{T}$ be a $\mathsf{T}_{\mathsf{LTBI}}$-proof of A. Assume that A is not valid, then there exists a linear resource model $\mathcal{M}$ such that $(\epsilon, s) \nVdash \mathrm{A}$ for some state s. Since the initial CTSS $\langle \{\, \mathbb{F}\ \mathrm{A} : (\epsilon_{\mathrm{L}}, \gamma_0)\,\}, \{\, \epsilon_{\mathrm{L}} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \epsilon_{\mathrm{L}}\,\}, \{\, \gamma_0 \leqslant^{\mathfrak{s}}_{\mathrm{L}} \gamma_0\,\}\rangle$ is trivially realizable by setting $[\gamma_0]_s = s$, Lemma 25 implies that the tableau $\mathcal{T}$ contains at least one realizable branch, which contradicts the fact that $\mathcal{T}$ is a tableau proof. Indeed, if $\mathcal{T}$ is a tableau proof for A, then all of its branches should be closed by definition, and thus not realizable by Lemma 24. Therefore, A is valid. ◀

5 Completeness

In this section we discuss the reasons why the completeness result for $\mathsf{T}_{\mathsf{LTBI}}$ is not trivial and still an open problem.

A usual way of proving the completeness of a labelled tableau calculus is by counter-model construction from an open and completed branch, as we did for BI [7], BBI [12] and various modal extensions of BI [3, 4]. This approach requires the definition of a suitable notion of what it means for a labelled formula to be completely analyzed or fulfilled. Although such a definition can be given for $\mathsf{T}_{\mathsf{LTBI}}$, the completion of an open branch raises several issues.

▶ **Definition 27.** *Let* $\langle \mathcal{F}, C_s, C_r \rangle$ *be the CTSS associated to a tableau branch* $\mathcal{B}$*. A labelled formula* $\mathbb{S}\ \mathrm{C} : (\mathrm{x}, \tau)$ *is* fulfilled *(or* completely analyzed*) in* $\mathcal{B}$*, denoted* $\mathcal{B} \models \mathbb{S}\ \mathrm{C} : (\mathrm{x}, \tau)$*, iff:*

- *Base cases:*
  - $\mathcal{B} \models \mathbb{S}\ \top : (\mathrm{x}, \tau)$ *always*
  - $\mathcal{B} \models \mathbb{S}\ \bot : (\mathrm{x}, \tau)$ *always*
  - $\mathcal{B} \models \mathbb{T}\ \mathrm{I} : (\mathrm{x}, \tau)$ *iff* $\epsilon_{\mathrm{L}} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x} \in C_r^{\bullet}$
  - $\mathcal{B} \models \mathbb{F}\ \mathrm{I} : (\mathrm{x}, \tau)$ *always*
  - $\mathcal{B} \models \mathbb{T}\ \mathrm{p} : (\mathrm{x}, \tau)$ *iff* $\mathbb{T}\ \mathrm{p} : (\mathrm{y}, \tau) \in \mathcal{F}$ *for some* $\mathrm{y} \neq \mathrm{x}$ *such that* $\mathrm{y} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x} \in C_r^{\bullet}$
  - $\mathcal{B} \models \mathbb{F}\ \mathrm{p} : (\mathrm{x}, \tau)$ *iff* $\mathbb{F}\ \mathrm{p} : (\mathrm{y}, \tau) \in \mathcal{F}$ *for some* $\mathrm{y} \neq \mathrm{x}$ *such that* $\mathrm{x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{y} \in C_r^{\bullet}$

- *Induction:*
  - $\mathcal{B} \models \mathbb{T}\ \mathrm{A} \wedge \mathrm{B} : (\mathrm{x}, \tau)$ *iff* $\mathcal{B} \models \mathbb{T}\ \mathrm{A} : (\mathrm{x}, \tau)$ *and* $\mathcal{B} \models \mathbb{T}\ \mathrm{B} : (\mathrm{x}, \tau)$
  - $\mathcal{B} \models \mathbb{F}\ \mathrm{A} \wedge \mathrm{B} : (\mathrm{x}, \tau)$ *iff* $\mathcal{B} \models \mathbb{F}\ \mathrm{A} : (\mathrm{x}, \tau)$ *or* $\mathcal{B} \models \mathbb{F}\ \mathrm{B} : (\mathrm{x}, \tau)$
  - $\mathcal{B} \models \mathbb{T}\ \mathrm{A} \vee \mathrm{B} : (\mathrm{x}, \tau)$ *iff* $\mathcal{B} \models \mathbb{T}\ \mathrm{A} : (\mathrm{x}, \tau)$ *or* $\mathcal{B} \models \mathbb{T}\ \mathrm{B} : (\mathrm{x}, \tau)$
  - $\mathcal{B} \models \mathbb{F}\ \mathrm{A} \vee \mathrm{B} : (\mathrm{x}, \tau)$ *iff* $\mathcal{B} \models \mathbb{F}\ \mathrm{A} : (\mathrm{x}, \tau)$ *and* $\mathcal{B} \models \mathbb{F}\ \mathrm{B} : (\mathrm{x}, \tau)$
  - $\mathcal{B} \models \mathbb{T}\ \mathrm{A} * \mathrm{B} : (\mathrm{x}, \tau)$ *iff* $\mathcal{B} \models \mathbb{T}\ \mathrm{A} : (\mathrm{y}, \tau)$ *and* $\mathcal{B} \models \mathbb{T}\ \mathrm{B} : (\mathrm{z}, \tau)$ *for some* $\mathrm{y\,z} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x} \in C_r^\bullet$
  - $\mathcal{B} \models \mathbb{F}\ \mathrm{A} * \mathrm{B} : (\mathrm{x}, \tau)$ *iff* $\mathcal{B} \models \mathbb{F}\ \mathrm{A} : (\mathrm{y}, \tau)$ *and* $\mathcal{B} \models \mathbb{F}\ \mathrm{B} : (\mathrm{z}, \tau)$ *for all* $\mathrm{y\,z} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x} \in C_r^\bullet$
  - $\mathcal{B} \models \mathbb{T}\ \mathrm{A} \rightarrow \mathrm{B} : (\mathrm{x}, \tau)$ *iff* $\mathcal{B} \models \mathbb{F}\ \mathrm{A} : (\mathrm{y}, \tau)$ *or* $\mathcal{B} \models \mathbb{T}\ \mathrm{B} : (\mathrm{y}, \tau)$ *for all* $\mathrm{x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{y} \in C_r^\bullet$
  - $\mathcal{B} \models \mathbb{F}\ \mathrm{A} \rightarrow \mathrm{B} : (\mathrm{x}, \tau)$ *iff* $\mathcal{B} \models \mathbb{T}\ \mathrm{A} : (\mathrm{y}, \tau)$ *and* $\mathcal{B} \models \mathbb{F}\ \mathrm{B} : (\mathrm{y}, \tau)$ *for some* $\mathrm{x} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{y} \in C_r^\bullet$
  - $\mathcal{B} \models \mathbb{T}\ \mathrm{A} \mathrel{-\!\!*} \mathrm{B} : (\mathrm{x}, \tau)$ *iff* $\mathcal{B} \models \mathbb{F}\ \mathrm{A} : (\mathrm{y}, \tau)$ *or* $\mathcal{B} \models \mathbb{T}\ \mathrm{B} : (\mathrm{z}, \tau)$ *for all* $\mathrm{x\,y} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{z} \in C_r^\bullet$
  - $\mathcal{B} \models \mathbb{F}\ \mathrm{A} \mathrel{-\!\!*} \mathrm{B} : (\mathrm{x}, \tau)$ *iff* $\mathcal{B} \models \mathbb{T}\ \mathrm{A} : (\mathrm{y}, \tau)$ *and* $\mathcal{B} \models \mathbb{F}\ \mathrm{B} : (\mathrm{z}, \tau)$ *for some* $\mathrm{x\,y} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{z} \in C_r^\bullet$
  - $\mathcal{B} \models \mathbb{S}\ \circ\mathrm{A} : (\mathrm{x}, \tau)$ *iff* $\mathcal{B} \models \mathbb{S}\ \mathrm{A} : (\mathrm{y}, \eta\tau)$
  - $\mathcal{B} \models \mathbb{T}\ \Diamond\mathrm{A} : (\mathrm{x}, \tau)$ *iff* $\mathcal{B} \models \mathbb{T}\ \mathrm{A} : (\mathrm{y}, \upsilon)$ *for some* $\tau \leqslant^{\mathfrak{s}}_{\mathrm{L}} \upsilon \in C_s^\bullet$
  - $\mathcal{B} \models \mathbb{F}\ \Diamond\mathrm{A} : (\mathrm{x}, \tau)$ *iff* $\mathcal{B} \models \mathbb{F}\ \mathrm{A} : (\mathrm{y}, \upsilon)$ *for all* $\tau \leqslant^{\mathfrak{s}}_{\mathrm{L}} \upsilon \in C_s^\bullet$
  - $\mathcal{B} \models \mathbb{T}\ \Box\mathrm{A} : (\mathrm{x}, \tau)$ *iff* $\mathcal{B} \models \mathbb{T}\ \mathrm{A} : (\mathrm{y}, \upsilon)$ *for all* $\tau \leqslant^{\mathfrak{s}}_{\mathrm{L}} \upsilon \in C_s^\bullet$
  - $\mathcal{B} \models \mathbb{F}\ \Box\mathrm{A} : (\mathrm{x}, \tau)$ *iff* $\mathcal{B} \models \mathbb{F}\ \mathrm{A} : (\mathrm{y}, \upsilon)$ *for some* $\tau \leqslant^{\mathfrak{s}}_{\mathrm{L}} \upsilon \in C_s^\bullet$

▶ **Definition 28.** *A branch* $\mathcal{B}$ *is* completed *(also* saturated*) if all of its labelled formulas are fulfilled and all possible expansions of the structural rules* CD *and* LR *have been applied.*

It is folklore to define a completion procedure for an open branch by defining a fair strategy for formula expansion (see [5, 4] for details). The actual problem is to turn an open and completed branch into a suitable LTBI counter-model.

5.1 Counter-Model Construction

Let us first illustrate how to construct a counter-model from an open and completed branch using the leftmost open branch of the tableau depicted in Figure 3.

Firstly, we define the set of resources as the set $D_r(C_r^\bullet) \cup \{\, \pi \,\}$ and the composition of resources as:

$$\begin{cases} \mathrm{x} \star \mathrm{y} = \mathrm{xy} \text{ if } \mathrm{xy} \in D_r(C_r^\bullet) \\ \mathrm{x} \star \epsilon_{\mathrm{L}} = \mathrm{x} \\ \mathrm{x} \star \pi = \pi \end{cases}$$

The resource ordering $\leqslant^{\mathfrak{r}}$ is induced by the closure of the resource assertions occurring in the branch, i.e.:

$$\leqslant^{\mathfrak{r}} \;=\; C_r^\bullet \cup \{\, \mathrm{x} \leqslant \pi \mid \mathrm{x} \in D_r(C_r^\bullet),\, \}$$

which, in our example, corresponds to the following transitive and reflevixe closure of the set of relations:

$$\{\, \epsilon_{\mathrm{L}} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{c}_1, \mathrm{c}_2\mathrm{c}_3 \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{c}_1 \,\}$$

augmented with π as the greatest element.

Secondly, the timeline is defined as the set $\{\, 0, 1, 2 \,\}$ with the state labels realized (interpreted) as follows: $[\gamma_0]_s = 0$, $[\eta\gamma_0]_s = 1$, $[\gamma_1]_s = 2$.

Thirdly, the forcing relation is induced by the following LTBI-valuation that matches the positive labelled formulas (those with a sign $\mathbb{T}$) occurring in the branch:

$$\begin{cases} [\mathrm{A}] = \{\, (\pi, 0), (\pi, 1), (\pi, 2), (\mathrm{c}_2, 2) \,\} \\ [\mathrm{B}] = \{\, (\pi, 0), (\pi, 1), (\pi, 2), (\mathrm{c}_3, 2) \,\} \end{cases}$$

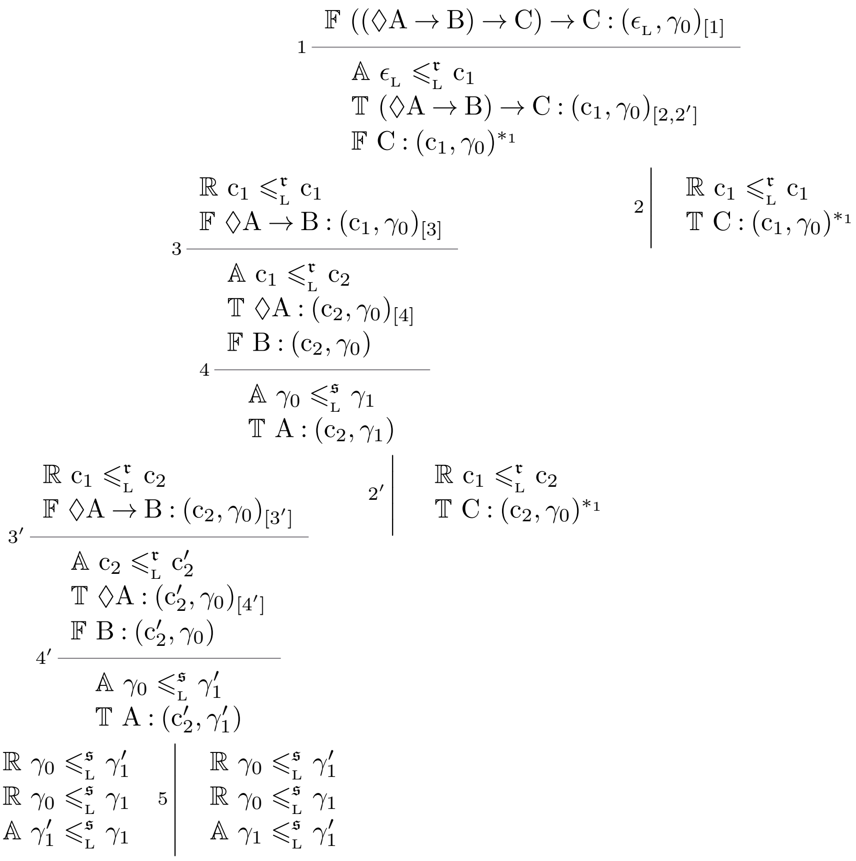

$\mathbb{F}\ ((\Diamond A \to B) \to C) \to C : (\epsilon_L, \gamma_0)_{[1]}$
1
$\mathbb{A}\ \epsilon_L \leqslant_L^{\mathfrak{r}} c_1$
$\mathbb{T}\ (\Diamond A \to B) \to C : (c_1, \gamma_0)_{[2,2']}$
$\mathbb{F}\ C : (c_1, \gamma_0)^{*_1}$
$\mathbb{R}\ c_1 \leqslant_L^{\mathfrak{r}} c_1$
$\mathbb{F}\ \Diamond A \to B : (c_1, \gamma_0)_{[3]}$
2
$\mathbb{R}\ c_1 \leqslant_L^{\mathfrak{r}} c_1$
$\mathbb{T}\ C : (c_1, \gamma_0)^{*_1}$
3
$\mathbb{A}\ c_1 \leqslant_L^{\mathfrak{r}} c_2$
$\mathbb{T}\ \Diamond A : (c_2, \gamma_0)_{[4]}$
$\mathbb{F}\ B : (c_2, \gamma_0)$
4
$\mathbb{A}\ \gamma_0 \leqslant_L^{\mathfrak{s}} \gamma_1$
$\mathbb{T}\ A : (c_2, \gamma_1)$
$\mathbb{R}\ c_1 \leqslant_L^{\mathfrak{r}} c_2$
$\mathbb{F}\ \Diamond A \to B : (c_2, \gamma_0)_{[3']}$
$2'$
$\mathbb{R}\ c_1 \leqslant_L^{\mathfrak{r}} c_2$
$\mathbb{T}\ C : (c_2, \gamma_0)^{*_1}$
$3'$
$\mathbb{A}\ c_2 \leqslant_L^{\mathfrak{r}} c_2'$
$\mathbb{T}\ \Diamond A : (c_2', \gamma_0)_{[4']}$
$\mathbb{F}\ B : (c_2', \gamma_0)$
$4'$
$\mathbb{A}\ \gamma_0 \leqslant_L^{\mathfrak{s}} \gamma_1'$
$\mathbb{T}\ A : (c_2', \gamma_1')$
$\mathbb{R}\ \gamma_0 \leqslant_L^{\mathfrak{s}} \gamma_1'$
$\mathbb{R}\ \gamma_0 \leqslant_L^{\mathfrak{s}} \gamma_1$
$\mathbb{A}\ \gamma_1' \leqslant_L^{\mathfrak{s}} \gamma_1$
5
$\mathbb{R}\ \gamma_0 \leqslant_L^{\mathfrak{s}} \gamma_1'$
$\mathbb{R}\ \gamma_0 \leqslant_L^{\mathfrak{s}} \gamma_1$
$\mathbb{A}\ \gamma_1 \leqslant_L^{\mathfrak{s}} \gamma_1'$

Figure 4 Liberizable Infinite Tableau.

Finally, the reason why we have an actual counter-model can be read directly from the labelled formulas of the completed open branch:

1. We have $(c_2, 2) \Vdash A$ (by definition), which implies $(c_2, 0) \Vdash \Diamond A$.
2. Moreover, we have $(c_3, 1) \Vdash B$ (by definition) and thus we get $(c_3, 0) \Vdash \circ B$.
3. From 1 and 2, we get $(c_2c_3, 0) \Vdash \Diamond A * \circ B$ which implies $(c_1, 0) \Vdash \Diamond A * \circ B$ by Kripke monotonicity (as $c_2c_3 \leqslant c_1$ by definition).
4. Besides, we have $(c_0, 0) \nVdash \Diamond B * \circ A$ because $(x, \tau) \nVdash \circ A$ for all resources x and all states τ (since the timeline has no state 3 and A is only true at $(c_2, 2)$).

The first and third points (construction of a total resource monoid and of a forcing relation) described above work in the general case for any open and completed branch, not just for the tableau depicted in Figure 3. The second point (construction of discrete linear timeline) is however more problematic.

5.2 The Dense Timeline Issue

A first issue in $\mathsf{T}_{\mathsf{LTBI}}$ is that the completion procedure might result in a set of state constraints that, although representing a discrete linear order, might not be isomorphic to any subset of $(\mathbb{N}, \leqslant)$ because it might be dense.

Let us for example consider the tableau depicted in Fig. 4. Its leftmost branch grows infinitely because the $\pi\beta$-formula $\mathbb{T}\ (\Diamond A \to B) \to C$ contains a $\pi\alpha$-subformula $\mathbb{F}\ \Diamond A \to B$ the expansion of which repeatedly generates new resource constants c_2, c_2', c_2'', c_2^i $(i > 2)$ to be fed to the $\pi\beta$-formula for its fulfillment. For instance in Step [3], the resource assertion $\mathbb{A}\ \mathrm{c}_1 \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{c}_2$ is generated, where c_2 is fresh. Then, in Step [4], the state assertion $\mathbb{A}\ \gamma_0 \leqslant^{\mathfrak{s}}_{\mathrm{L}} \gamma_1$ is generated, where γ_1 is fresh. Since the requirement $\mathbb{R}\ \mathrm{c}_1 \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{c}_2$ is met, Step [2] must be reproduced with c_2 instead of c_1, which gives Step [$2'$]. After Step [$2'$], Steps [$3'$] and [$4'$] reproduce Steps [3] and [4] leading to new assertions $\mathbb{A}\ \mathrm{c}_2 \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{c}'_2$ and $\mathbb{A}\ \gamma_0 \leqslant^{\mathfrak{s}}_{\mathrm{L}} \gamma'_1$.

After Step [$4'$], we get two state labels γ_1 and γ'_1 that are not linearly ordered. We therefore use the linearizing rule LR in Step [5] to get (in the leftmost branch) the assertion $\mathbb{A}\ \gamma'_1 \leqslant^{\mathfrak{s}}_{\mathrm{L}} \gamma_1$. Several applications of the case distinction rule CD (not represented in Fig. 4 for conciseness) allow us to get the following ordering of the state labels: $\gamma_0 <^{s}_{\mathrm{L}} \gamma'_1 <^{s}_{\mathrm{L}} \gamma_1$. Repeating the previous steps infinitely many times we can generate a strictly decreasing infinite chain of state labels $(\gamma^i_1)_{i\in\mathbb{N}}$ between γ_0 and γ_1.

The situation described in Fig. 4 well illustrates the fact that our logic LTBI is not a simple and orthogonal combination of BI and LTL connectives, but induces an actual interaction between resource and state labels. Indeed, the infinite chain of state labels γ^i_1 derives from the creation of an infinite chain of resource labels c^i_2.

5.3 Unsoundness of the Liberalized Rules

Tableau branches that might grow infinitely because of the creation of infinitely many fresh labels is a problem that already occurs in tableaux for BI [7]. In the case of BI, such a situation can be remedied using liberalized versions of the tableaux rules that allow the reuse of previously generated labels under specific conditions.

For example, the rule $\mathbb{F}\rightarrow$ would be allowed to expand $\mathbb{F}\ \mathrm{A}\rightarrow\mathrm{B} : (\mathrm{x},\tau)$ to $\mathbb{T}\ \mathrm{A} : (\mathrm{x},\tau), \mathbb{F}\ \mathrm{B} : (\mathrm{x},\tau)$ without generating a fresh (resource) constant whenever the branch already contains a labelled formula $\mathbb{T}\ \mathrm{A} : (\mathrm{y},\tau)$ for which the requirement $\mathbb{R}\ \mathrm{y} \leqslant^{\mathfrak{r}}_{\mathrm{L}} \mathrm{x}$ is met. Under the liberalized version of $\mathbb{F}\rightarrow$, the leftmost branch of the tableau depicted in Fig. 4 would be completed after Step [$3'$] since the introduction of $\mathbb{T}\ \Diamond\mathrm{A} : (\mathrm{c}_2, \gamma_0)$ in Step [3] would allow Step [$3'$] to reuse c_2 instead of generating a fresh c'_2, making Step [$3'$] a redundant copy of Step [3] adding no new information to the branch.

It would be tempting to think that adopting the liberalized rules given for BI in [7] would solve the problem of getting an infinite amount of state labels from the generation of an infinite number of fresh resource labels. Unfortunately, our second issue is that this approach does not work, as illustrated in Fig. 5.

The liberalized rule for $\mathbb{T}\,*$ (resp. $\mathbb{F}\,{-\!\!*}$) in BI tableaux only generates fresh constants for the first instance of a labelled formula $\mathbb{T}\ \mathrm{A}*\mathrm{B} : \mathrm{x}$ (or $\mathbb{F}\ \mathrm{A}\,{-\!\!*}\,\mathrm{B} : \mathrm{x}$) in a tableau branch. Every subsequent instance of the same labelled formula in the same branch is allowed to reuse the constants that have been generated by the expansion of the first instance.

After Step [4], the tableau described in Fig. 5 splits into two branches, the second one being similar to the first one (replacing occurences of A with B) and thus not fully depicted in the figure for conciseness. As easily checked, repeating Steps [2] through [6] makes the leftmost branch of the tableau grow infinitely. The repetitions Step [3^i] of Step [3] generate infinitely many decompositions $\mathrm{c}^i_2\,\mathrm{c}^i_3 (i \in \mathbb{N})$ of the resource constant c_1. In turn, this leads to the repetitions Step [5^i] of Step [5] which generate infinitely many state labels γ^i_1 and state assertions $\mathbb{A}\ \gamma_0 \leqslant^{\mathfrak{s}}_{\mathrm{L}} \gamma^i_1$.

Using the liberalized version of $\mathbb{T}\,*$ in Step [$3'$] as in BI tableaux would result in reusing the constants c_2 and c_3 generated during Step [3] instead of introducing the new constants c'_2 and c'_3. The branch would then be closed, having both $\mathbb{T}\ \mathrm{A} : (\mathrm{c}_2, \gamma_1)$ from Step [$3'$] and $\mathbb{F}\ \mathrm{A} : (\mathrm{c}_2, \gamma_1)$ from Step [5]. Proceeding similarly in the branch that is eluded in Fig. 5, we would finally get a closed $\mathsf{T}_{\mathsf{LTBI}}$ tableau for a formula which is not valid in LTBI. This shows that the liberalized rules for BI are not sound for LTBI.

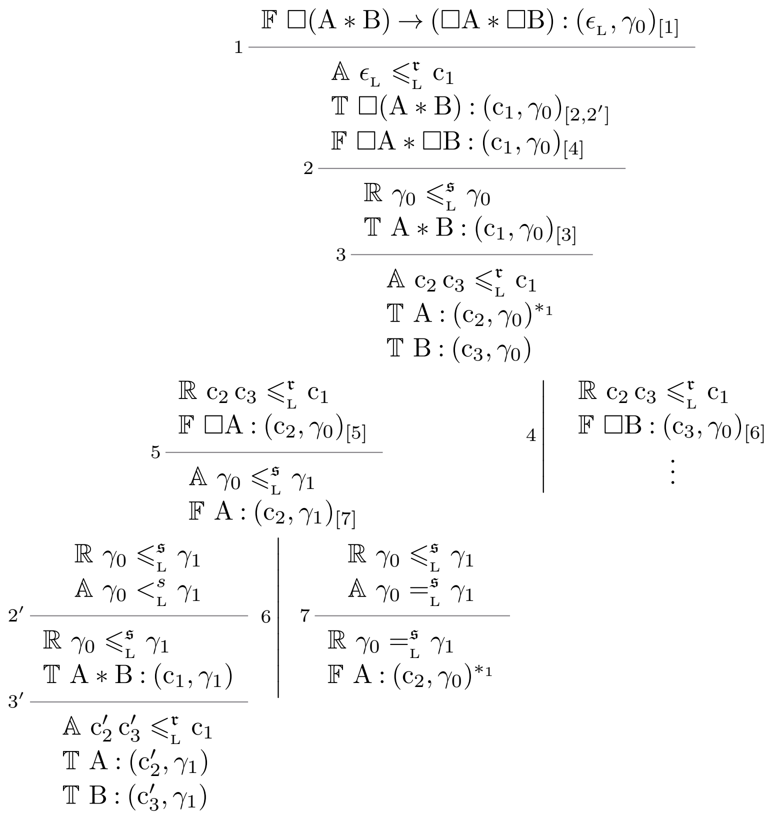

$\mathbb{F}\ \Box(A * B) \to (\Box A * \Box B) : (\epsilon_L, \gamma_0)_{[1]}$
1
$\mathbb{A}\ \epsilon_L \leqslant^{\mathfrak{r}}_L c_1$
$\mathbb{T}\ \Box(A * B) : (c_1, \gamma_0)_{[2,2']}$
$\mathbb{F}\ \Box A * \Box B : (c_1, \gamma_0)_{[4]}$
2
$\mathbb{R}\ \gamma_0 \leqslant^{\mathfrak{s}}_L \gamma_0$
$\mathbb{T}\ A * B : (c_1, \gamma_0)_{[3]}$
3
$\mathbb{A}\ c_2\, c_3 \leqslant^{\mathfrak{r}}_L c_1$
$\mathbb{T}\ A : (c_2, \gamma_0)^{*1}$
$\mathbb{T}\ B : (c_3, \gamma_0)$
$\mathbb{R}\ c_2\, c_3 \leqslant^{\mathfrak{r}}_L c_1$
$\mathbb{F}\ \Box A : (c_2, \gamma_0)_{[5]}$
5
$\mathbb{A}\ \gamma_0 \leqslant^{\mathfrak{s}}_L \gamma_1$
$\mathbb{F}\ A : (c_2, \gamma_1)_{[7]}$
4
$\mathbb{R}\ c_2\, c_3 \leqslant^{\mathfrak{r}}_L c_1$
$\mathbb{F}\ \Box B : (c_3, \gamma_0)_{[6]}$
$\vdots$
$\mathbb{R}\ \gamma_0 \leqslant^{\mathfrak{s}}_L \gamma_1$
$\mathbb{A}\ \gamma_0 <^{\mathfrak{s}}_L \gamma_1$
2′
$\mathbb{R}\ \gamma_0 \leqslant^{\mathfrak{s}}_L \gamma_1$
$\mathbb{T}\ A * B : (c_1, \gamma_1)$
3′
$\mathbb{A}\ c'_2\, c'_3 \leqslant^{\mathfrak{r}}_L c_1$
$\mathbb{T}\ A : (c'_2, \gamma_1)$
$\mathbb{T}\ B : (c'_3, \gamma_1)$
6
$\mathbb{R}\ \gamma_0 \leqslant^{\mathfrak{s}}_L \gamma_1$
$\mathbb{A}\ \gamma_0 =^{\mathfrak{s}}_L \gamma_1$
7
$\mathbb{R}\ \gamma_0 =^{\mathfrak{s}}_L \gamma_1$
$\mathbb{F}\ A : (c_2, \gamma_0)^{*1}$

Figure 5 Unliberizable Infinite Tableau.

5.4 Non-equivalence of LTBI and tBI

In BI tableaux, the soundness of the liberalized rules (as well as the decidability arguments for BI) does not rely on the widespread Kripke resource semantics of BI, but rather on its Beth resource semantics (see [7] for details). The fact that the liberalized rules are unsound for $\mathsf{T}_{\mathsf{LTBI}}$ suggests that replacing the Kripke resource monoid in Definition 2 with a Beth resource monoid would yield a non-equivalent resource semantics for LTBI.

In [10], both a logic called tBI (mixing LTL and BI) for linear bounded timelines and a corresponding purely syntactic sound and complete sequent style proof-system called GtBI are introduced. The semantics of tBI is an extension of the Grothendieck topological resource semantics of BI. The GtBI sequent system is an extension of LBI, the standard bunched sequent calculus of BI. The Grothendieck topological semantics of BI is shown in [7] to be equivalent to its Beth resource semantics w.r.t. provability in LBI, more precisely, for any BI formula A, we have $\models_{\mathsf{Beth}} A \Leftrightarrow \vdash_{\mathsf{LBI}} A \Leftrightarrow \models_{\mathsf{Grot}} A$. Therefore, the unsoundness of the liberalized rules for $\mathsf{T}_{\mathsf{LTBI}}$ proves that even if we would extend GtBI to deal with unbounded timelines, it would be hopeless to try to show the completeness of $\mathsf{T}_{\mathsf{LTBI}}$ by translating proofs of GtBI (with liberalized rules) into closed $\mathsf{T}_{\mathsf{LTBI}}$ tableaux.

More importantly, as stated in Definition 5, the validity of a formula in $\mathsf{T}_{\mathsf{LTBI}}$ only depends on its satisfiability in all time states for the empty resource ϵ, while its validity in tBI depends on its satisfiability in all time states for all resources in the underlying Grothendieck resource monoid. Consequently, although seemingly (syntactically) similar, LTBI and tBI are semantically distinct logics and the results obtained for tBI in [10] do not apply to LTBI.

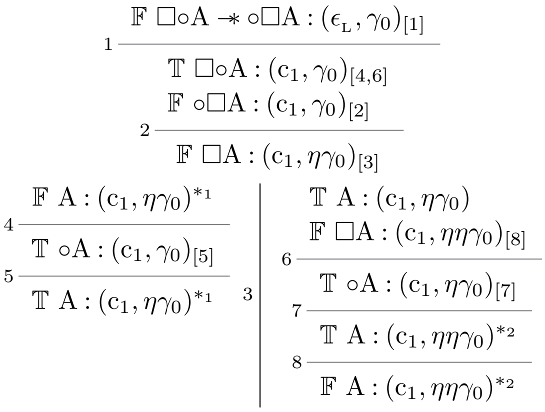

$\mathbb{F}\ \Box\circ A \mathrel{-\!\!*} \circ\Box A : (\epsilon_L, \gamma_0)_{[1]}$
1
$\mathbb{T}\ \Box\circ A : (c_1, \gamma_0)_{[4,6]}$
$\mathbb{F}\ \circ\Box A : (c_1, \gamma_0)_{[2]}$
2
$\mathbb{F}\ \Box A : (c_1, \eta\gamma_0)_{[3]}$
$\mathbb{F}\ A : (c_1, \eta\gamma_0)^{*_1}$
4
$\mathbb{T}\ \circ A : (c_1, \gamma_0)_{[5]}$
5
$\mathbb{T}\ A : (c_1, \eta\gamma_0)^{*_1}$
3
$\mathbb{T}\ A : (c_1, \eta\gamma_0)$
$\mathbb{F}\ \Box A : (c_1, \eta\eta\gamma_0)_{[8]}$
6
$\mathbb{T}\ \circ A : (c_1, \eta\gamma_0)_{[7]}$
7
$\mathbb{T}\ A : (c_1, \eta\eta\gamma_0)^{*_2}$
8
$\mathbb{F}\ A : (c_1, \eta\eta\gamma_0)^{*_2}$

Figure 6 Tableau with Bounded Timeline of Length 3.

5.5 The Bounded Timeline Case

We can solve the completeness issues discussed previously by restricting the semantics of LTBI to bounded timelines. It is well known that LTL with bounded time domains can prove almost all of the typical axioms of unbounded LTL. Moreover, practical uses of LTL almost always consider bounded time domains.

Let us assume a bounded timeline $\mathbf{S} = \mathbf{S}_n = \{\, i < n \mid i \in \mathbb{N} \,\}$ of length $n \in \mathbb{N}^*$. Using the fixpoint definitions of the modal operators, we can derive a new tableau system $\mathsf{T}^{\mathsf{n}}_{\mathsf{LTBI}}$ in which the rules $\mathbb{T}\Diamond$ and $\mathbb{F}\Box$ of $\mathsf{T}_{\mathsf{LTBI}}$ are replaced by the following fixpoint rules:

- when $i < n - 1$:

$$\begin{array}{c} \mathbb{T}\ \Diamond A : (x, \eta^i\gamma_0) \\ \mathbb{T}\ A : (x, \eta^i\gamma_0) \;\Bigg|\; \begin{array}{l} \mathbb{F}\ A : (x, \eta^i\gamma_0) \\ \mathbb{T}\ \Diamond A : (x, \eta^{i+1}\gamma_0) \end{array} \end{array} \qquad \begin{array}{c} \mathbb{F}\ \Box A : (x, \eta^i\gamma_0) \\ \mathbb{F}\ A : (x, \eta^i\gamma_0) \;\Bigg|\; \begin{array}{l} \mathbb{T}\ A : (x, \eta^i\gamma_0) \\ \mathbb{F}\ \Box A : (x, \eta^{i+1}\gamma_0) \end{array} \end{array}$$

- when $i = n - 1$:

$$\frac{\mathbb{T}\ \Diamond A : (x, \eta^i\gamma_0)}{\mathbb{T}\ A : (x, \eta^i\gamma_0)} \qquad \frac{\mathbb{F}\ \Box A : (x, \eta^i\gamma_0)}{\mathbb{F}\ A : (x, \eta^i\gamma_0)}$$

Let us remark that we distinguish two cases (when $i < n-1$ and when $i = n-1$) because in our semantics (as described in Definition 4), the truth of the next modality requires the existence of a successor. A semantics in which the next modality is true whenever interpreted in a time state which is out of the bounds (as in tBI) can be obtained by using only the first pair of rules (the forking rules) in any case. Figure 6 gives an example of a closed bounded tableau of length 3 for the formula $\Box\circ A \mathrel{-\!\!*} \circ\Box A$.

With the fixpoint rules, we claim the following completeness result for bounded tableaux:

▷ Claim 29. $\mathsf{T}^{\mathsf{n}}_{\mathsf{LTBI}}$ is complete for bounded timelines of length n.

Proof (Sketch). We first observe that in $\mathsf{T}_{\mathsf{LTBI}}$ the only rules that can introduce new state labels are the rules $\mathbb{T}\Diamond$ and $\mathbb{F}\Box$. In $\mathsf{T}^{\mathsf{n}}_{\mathsf{LTBI}}$ those rules are replaced with the fixpoint rules that no longer introduce new state labels, but create terms of the form $\eta^i\gamma_0$ from the root state label γ_0. Therefore, once γ_0 is interpreted as 0 and η is interpreted as the successor function, the generated timeline cannot be dense. Finally, since there are only finitely many terms of the form $\eta^i\gamma_0$ with $0 \leqslant i < n$, the tableau branch completion procedure necessarily terminates. Now, if the completion procedure results in an open branch, the counter-model construction procedure described in Section 5.1 yields an actual counter-model for the initial formula at the root of the tableau branch. ◁

6 Conclusion and Perspectives

In this paper we introduced a new resource logic called LTBI that mixes BI and LTL unary connectives. We proposed a labelled tableau proof system $\mathsf{T}_{\mathsf{LTBI}}$ for LTBI and proved its soundness. We discussed the various and non-trivial completeness issues that arise when trying to show the completeness of $\mathsf{T}_{\mathsf{LTBI}}$ in the general case of an unbounded timelime.

A first perspective is to give a detailed proof of the completeness result claimed previously for bounded timelines.

A second perspective is to extend the completeness result to unbounded timelines. Such an extension would necessarily require the definition of a cyclic proof system with some form of induction to decide when the fixpoint rules should stop forking. Closing conditions for sequent style cyclic proof systems have been given in the literature for unbounded LTL and the task is not at all trivial (as explained in [1]). It is presently unclear to us how to adapt such cyclic closing conditions in the context of a labelled tableau calculus and in the presence of BI multiplicative connectives.

A third perspective is to study variants of LTBI, for example variants that incorporate the binary temporal connectives U and R (until and release), or variants where the underlying resource composition is bounded (e.g. $r^n = \pi$ when $n > p$ for some $p \in \mathbb{N}^*$) or satisfies more specific axioms (e.g., $r \star r \leqslant^{\mathfrak{r}} r$).

References

1 Kai Brünnler and Martin Lange. Cut-free sequent systems for temporal logic. *The Journal of Logic and Algebraic Programming*, 76(2):216–225, 2008.

2 Edmund M Clarke and I Anca Draghicescu. Expressibility results for Linear-time and Branching-time Logics. In *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems), LNCS 354*, pages 428–437. Springer, 1988.

3 Jean-René Courtault and Didier Galmiche. A modal BI Logic for Dynamic Resource Properties. In *Int. Symposium on Logical Foundations of Computer Science, LFCS 2013, LNCS 7734*, pages 134–148. Springer, 2013.

4 Jean-René Courtault and Didier Galmiche. A Modal Separation Logic for Resource Dynamics. *Journal of Logic and Computation*, 28(4):733–778, 2018.

5 Didier Galmiche and Daniel Méry. Semantic Labelled Tableaux for Propositional BI without ⊥. *Journal of Logic and Computation*, 13(5):707–753, 2003.

6 Didier Galmiche and Daniel Méry. Tableaux and Resource Graphs for Separation Logic. *Journal of Logic and Computation*, 20(1):189–231, 2010.

7 Didier Galmiche, Daniel Méry, and David Pym. The semantics of BI and Resource Tableaux. *Mathematical Structures in Computer Science*, 15(6):1033–1088, 2005.

8 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1—101, 1987.

9 Samin S Ishtiaq and Peter W. O'Hearn. BI as an Assertion language for Mutable Data Structures. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 14—26, 2001.

10 Norohiro Kamide. Temporal BI: proof system, semantics and translations. *Theoretical Computer Science*, 492:40–69, 2013.

11 Fred Kröger and Stephan Merz. *Temporal Logic and State Systems (Texts in Theoretical Computer Science. an EATCS Series)*. Springer Publishing Company, Incorporated, 2008.

12 Dominique Larchey-Wendling and Didier Galmiche. The Undecidability of Boolean BI through phase Semantics. In *2010 25th Annual IEEE Symposium on Logic in Computer Science*, pages 140−149. IEEE, 2010.

13 Peter W. O'Hearn and David J. Pym. The Logic of Bunched Implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

14 David J. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26. Applied Logic Series. Kluwer Academic Publishers, 2002.

15 John C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. *17th Annual IEEE Symposium on Logic in Computer Science (LICS'02)*, pages 55–74, 2002.

16 Kristin Y. Rozier. Linear Temporal Logic Symbolic Model Checking. *Computer Science Review*, 5(2):163–203, 2011.

Diller-Nahm Bar Recursion

Valentin Blot
Inria, Laboratoire Méthodes Formelles, Université Paris-Saclay, France

Abstract

We present a generalization of Spector's bar recursion to the Diller-Nahm variant of Gödel's Dialectica interpretation. This generalized bar recursion collects witnesses of universal formulas in sets of approximation sequences to provide an interpretation to the double-negation shift principle. The interpretation is presented in a fully computational way, implementing sets via lists. We also present a demand-driven version of this extended bar recursion manipulating partial sequences rather than initial segments. We explain why in a Diller-Nahm context there seems to be several versions of this demand-driven bar recursion, but no canonical one.

2012 ACM Subject Classification Theory of computation → Proof theory

Keywords and phrases Dialectica, Bar recursion

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.32

1 Introduction

Gödel's functional interpretation [5], also known as the Dialectica interpretation (from the name of the journal it was published in) is a translation from intuitionistic arithmetic into the Σ_2^0 fragment of intuitionistic arithmetic in finite types. If π is a proof of arithmetical formula A, then the functional interpretation of π is a proof of a formula $A^D \equiv \exists\vec{x}\forall\vec{y}A_D\left(\vec{x},\vec{y}\right)$ where A_D is quantifier-free. This formula can be understood as asserting that some two-player game has a winning strategy: there exists a strategy $\vec{x}$ such that for all strategy $\vec{y}$, $\vec{x}$ wins against $\vec{y}$, that is, $A_D\left(\vec{x},\vec{y}\right)$ holds. By the witness property, the proof of A^D yields a proof of $\forall\vec{y}A_D\left(\vec{t},\vec{y}\right)$ for some sequence of terms $\vec{t}$ in system T: simply-typed λ-calculus with recursion over natural numbers at all finite types. This sequence $\vec{t}$ of programs is the computational content of π under the Dialectica interpretation.

Since the negative translation of every axiom of arithmetic is provable in intuitionistic arithmetic, the Dialectica interpretation combined with a negative translation provides an interpretation of classical arithmetic. When it comes to classical analysis (classical arithmetic plus the axiom of countable choice) this is not true anymore, as the negative translation of the axiom of choice fails to be an intuitionistic consequence of the axiom of choice. Spector's bar recursion operator [11] provides a Dialectica interpretation of the double-negation shift (DNS) principle, from which one can derive intuitionistically any formula from its negative translation. Applying this to the axiom of countable choice, Spector obtains an interpretation of classical analysis.

Interpreting the contraction rule $A \Rightarrow A \wedge A$ in Gödel's original interpretation requires (besides the $\lambda x.\left\langle x,x\right\rangle$ component) a program that, given a witness M and two potential counterwitnesses x and y of A such that either x or y wins against M, answers with a single counterwitness that wins against M. Doing that relies on the decidability of winningness, which ultimately relies on the decidability of atomic formulas of the source logic (which is true in arithmetic). In order to get rid of this decidability requirement, Diller and Nahm [2] defined a variant of Gödel's interpretation where the programs provide a finite set of counterwitnesses, with the requirement that at least one is correct. In the previous example, the program interpreting the contraction rule answers with the set $\{x; y\}$ and does not have to decide which one is correct. The first contribution of this paper is the extension of Spector's bar

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 32; pp. 32:1–32:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

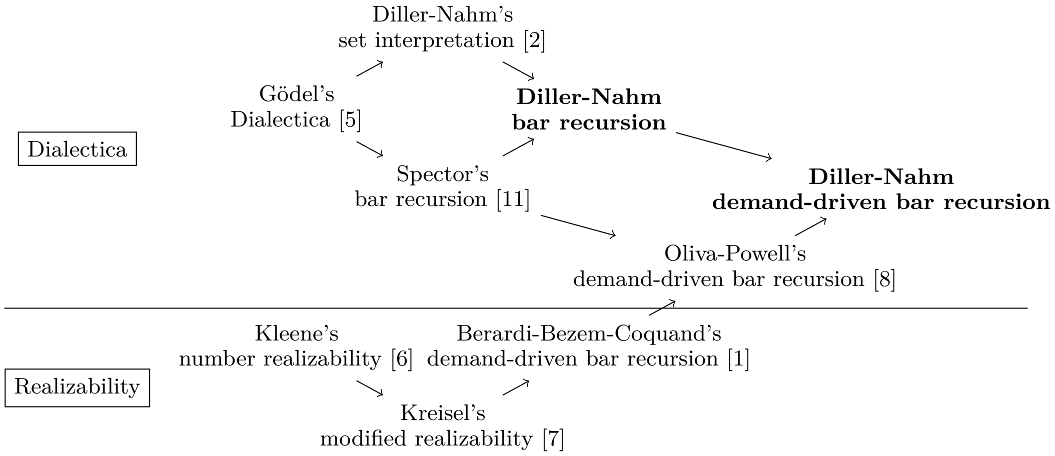


Figure 1 Contributions of the paper, **in bold font**.

recursion to the Diller-Nahm setting. Our operator has a lot in common with the extension of bar recursion to the Herbrand functional interpretation of non-standard arithmetic [3], though there are notable differences which are discussed.

Berardi, Bezem and Coquand [1] adapted Spector's bar recursion from Gödel's Dialectica to Kreisel's modified realizability [7]. Their operator also behaves differently from Spector's original bar recursion as it is demand-driven: it computes the choice sequence in an order that is driven by the environment, rather than in the natural order on natural numbers. This provides a more natural computational interpretation to the axiom of countable choice. More recently, Oliva and Powell [8] adapted Berardi-Bezem-Coquand's operator to Gödel's Dialecica interpretation and obtained a demand-driven bar-recursive interpretation of the axiom of countable choice in this setting. The second contribution of this paper is the definition of a demand-driven bar recursion operator in the Diller-Nahm setting.

Figure 1 summarizes the two contributions of this paper as well as their relationship to the state of the art. An arrow from X to Y means that Y is an extension/refinement/variant of X, and we distinguish elements that take place in the framework of realizability from those that take place in the framework of Dialectica-style functional interpretations.

First, we introduce the logical system and programming language we are working with. Then, we define precisely the Dialectica interpretation of the former into the latter. Finally, we present the two contributions of the paper : first, a variant of Spector's bar recusion in the Diller-Nahm setting, and then a demand-driven version of this operator.

2 Logical System and Programming Language

This section contains the definitions of the logical system and programming language that we will use throughout the paper.

2.1 Logical System

The Dialectica interpretation was originally formulated in a "Hilbert-style" presentation of arithmetic. Moreover, nowadays it is still often presented as a translation on formulas that satisfies a soundness theorem. In this setting, the construction of the witness $\vec{x}$ of the interpretation $\exists\vec{x}\forall\vec{y}A_D\left(\vec{x},\vec{y}\right)$ of a formula A is hidden in the proof of the soundness theorem.

$$\frac{}{\Gamma \vdash p_i : A_i} \qquad\qquad \frac{\Gamma \vdash \pi : \bot}{\Gamma \vdash \bot_A(\pi) : A}$$

$$\frac{\Gamma \vdash \pi : A \Rightarrow \bot}{\Gamma \vdash \neg_i(\pi) : \neg A} \qquad \frac{\Gamma, p_{n+1} : A_{n+1} \vdash \pi : B}{\Gamma \vdash \lambda p_{n+1}.\pi : A_{n+1} \Rightarrow B} \qquad \frac{\Gamma \vdash \pi : A}{\Gamma \vdash \lambda x.\pi : \forall x\, A}\, {\scriptstyle x \notin FV(\Gamma)}$$

$$\frac{\Gamma \vdash \pi : \neg A}{\Gamma \vdash \neg_e(\pi) : A \Rightarrow \bot} \qquad \frac{\Gamma \vdash \pi_1 : A \Rightarrow B \qquad \Gamma \vdash \pi_2 : A}{\Gamma \vdash \pi_1\, \pi_2 : B} \qquad \frac{\Gamma \vdash \pi : \forall x\, A}{\Gamma \vdash \pi\, t : A\,[t/x]}$$

Figure 2 Rules of first-order logic.

In contrast we present here the Dialectica interpretation as a proof translation, in the line of Pédrot [9, 10]. For that, we define precisely the logical system under consideration as a version of natural deduction in a sequent-calculus presentation (with explicit listing of hypotheses). This system is equipped with proof terms in order to ease the definition of the Dialectica interpretation, but these should be seen as purely syntactic artefacts without any notion of β-reduction or cut-elimination. We then give an inductive definition of the program witnessing the interpretation of a formula.

We work in first-order logic parameterized by a signature that is a set of function symbols (ranged over by metaviariable f) and predicate symbols (ranged over by metaviariable P) with arities:

$$t, u ::= x \mid f\,(t_1, \ldots, t_n) \qquad A, B ::= P\,(t_1, \ldots, t_n) \mid \bot \mid \neg A \mid A \Rightarrow B \mid \forall x\, A$$

We use a primitive negation operator rather than a definition in terms of implication and absurdity. This is completely equivalent but will simplify a lot the functional interpretation in the next sections, especially that of the double-negation shift principle. We consider only a reduced set of connectives with no conjunction or existential quantification in order to simplify the presentation, but these could easily be handled. The rules of the system are given in figure 2, where Γ denotes a context of the form $\vec{p} : \vec{A} \equiv p_1 : A_1, \ldots, p_n : A_n$, where we use $\equiv$ throughout the paper for definitions at the meta-level.

Arithmetic

In the Dialectica interpretation, the theory under consideration is that of arithmetic. There are 4 function symbols: $\mathtt{0}$ or arity 0, $\mathtt{S}$ of arity 1, $+$ and $\times$ of arity 2, for which we use infix notation. The only predicate symbol is $=$ of arity 2, for which we use infix notation as well.

The axioms of arithmetic are, as usual, those of equality (reflexivity and Leibniz's scheme), the 4 axioms defining addition and multiplication, non-confusion, injectivity of $\mathtt{S}$ and the induction scheme. They are given in figure 3.

2.2 Programming Language

Gödel's initial interpretation targeted quantifier-free arithmetic at finite types: system T. Nowadays, system T is usually described as primitive recursive functionals, or simply-typed lambda-calculus with natural numbers.

We use a programming language that is a variant of system T where the type of natural numbers is replaced with types σ^* of lists of elements of type σ, for each σ. This type is extensively used for encoding sets that are manipulated in the Diller-Nahm variant of Dialectica. Also, we use it to encode the types of booleans and natural numbers, and the initial segments and partial functions manipulated in our versions of bar recursion.

$$\forall x\,(x = x) \qquad \forall xy\,(x = y \Rightarrow A\,[x/z] \Rightarrow A\,[y/z])$$
$$\forall x\,(x + 0 = x) \qquad \forall xy\,(x + \mathtt{S}\,y = \mathtt{S}\,(x + y))$$
$$\forall x\,(x \times 0 = 0) \qquad \forall xy\,(x \times \mathtt{S}\,y = x \times y + x)$$
$$\forall x\,\neg\,(\mathtt{S}\,x = 0) \qquad \forall xy\,(\mathtt{S}\,x = \mathtt{S}\,y \Rightarrow x = y)$$
$$A\,[\mathtt{0}/x] \Rightarrow \forall x\,(A \Rightarrow A\,[\mathtt{S}\,x/x]) \Rightarrow \forall x\,A$$

Figure 3 Axioms of arithmetic.

$$\frac{}{\Gamma \vdash x : \sigma}\;{\scriptstyle x:\sigma\in\Gamma} \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau}$$

$$\frac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N\rangle : \sigma \times \tau} \qquad \frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash M.1 : \sigma} \qquad \frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash M.2 : \tau} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1}$$

$$\frac{}{\Gamma \vdash \mathtt{nil} : \sigma^*} \qquad \frac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \sigma^*}{\Gamma \vdash M :: N : \sigma^*} \qquad \frac{}{\Gamma \vdash \mathtt{rec} : \tau \to (\sigma \to \sigma^* \to \tau \to \tau) \to \sigma^* \to \tau}$$

Figure 4 Typing rules of the programming language.

For simplicity we also include unit and product types in our programming language. The types and terms are as follows, the typing rules are given in figure 4:

$$\sigma, \tau ::= \sigma \to \tau \mid \sigma \times \tau \mid 1 \mid \sigma^* \qquad \begin{aligned} M, N ::= {}& x \mid \lambda x.M \mid M\,N \mid \langle M, N\rangle \mid M.1 \mid M.2 \mid \langle\rangle \\ & \mid \mathtt{nil} \mid M :: N \mid \mathtt{rec} \end{aligned}$$

Term constructions are, in order: variable, abstraction, application, pairing, first and second projections, unit, empty list, list consing and recursion on lists.

The operational semantics of this language is given by the following β-reduction system:

$$(\lambda x.M)\,N \to_\beta M\,[N/x] \qquad \langle M, N\rangle\,.1 \to_\beta M \qquad \langle M, N\rangle\,.2 \to_\beta N$$

$$\mathtt{rec}\,P\,Q\,\mathtt{nil} \to_\beta P \qquad \mathtt{rec}\,P\,Q\,(M :: N) \to_\beta Q\,M\,N\,(\mathtt{rec}\,P\,Q\,N)$$

As with Gödel's system T, every program of this simple programming language terminates.

As usual we combine variables bound by λ-abstraction, so $\lambda xy.y\,x$ is a notation for $\lambda x.\lambda y.y\,x$. Also, if some variable is bound in an expression, we allow the use of _ in place of the variable if the variable does not appear free in the term. For example, $\lambda x\_.x$ is a notation for $\lambda xy.x$.

We encode Booleans in our programming language as follows:

$$\mathtt{bool} \equiv 1^* \qquad \mathtt{false} \equiv \mathtt{nil} \qquad \mathtt{true} \equiv \langle\rangle :: \mathtt{nil}$$

With these definitions we can derive the following typing rules (a dashed line means that a rule is admissible):

$$\frac{}{\Gamma \vdash \mathtt{false} : \mathtt{bool}} \qquad \frac{}{\Gamma \vdash \mathtt{true} : \mathtt{bool}}$$

We define case analysis on Booleans via case analysis on lists, for which we can derive the expected typing rule:

$$\texttt{if}\,M\,\texttt{then}\,N\,\texttt{else}\,P \equiv \texttt{rec}\,P\,(\lambda\_\_\_.N)\,M$$

$$\dfrac{\Gamma \vdash M : \texttt{bool} \qquad \Gamma \vdash N : \sigma \qquad \Gamma \vdash P : \sigma}{\Gamma \vdash \texttt{if}\,M\,\texttt{then}\,N\,\texttt{else}\,P : \sigma}$$

We also define a few basic operations on Booleans:

$$M \,\&\, N \equiv \texttt{if}\,M\,\texttt{then}\,N\,\texttt{else}\,\texttt{false} \qquad \dfrac{\Gamma \vdash M : \texttt{bool} \qquad \Gamma \vdash N : \texttt{bool}}{\Gamma \vdash M \,\&\, N : \texttt{bool}}$$

$$\neg M \equiv \texttt{if}\,M\,\texttt{then}\,\texttt{false}\,\texttt{else}\,\texttt{true} \qquad \dfrac{\Gamma \vdash M : \texttt{bool}}{\Gamma \vdash \neg M : \texttt{bool}}$$

We also encode natural numbers as follows:

$$\texttt{nat} \equiv 1^* \qquad 0 \equiv \texttt{nil} \qquad \mathsf{S}\,M \equiv \langle\rangle :: M \qquad \dfrac{}{\Gamma \vdash 0 : \texttt{nat}} \qquad \dfrac{\Gamma \vdash M : \texttt{nat}}{\Gamma \vdash \mathsf{S}\,M : \texttt{nat}}$$

Recursion on natural numbers and its expected derivable typing rule are as follows:

$$\texttt{rec}^{\mathbb{N}}\,M\,N \equiv \texttt{rec}\,M\,(\lambda\_.N) \qquad \dfrac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \texttt{nat} \to \sigma \to \sigma}{\Gamma \vdash \texttt{rec}^{\mathbb{N}}\,M\,N : \texttt{nat} \to \sigma}$$

We also define a few basic operations on natural numbers:

$$M = N \equiv \texttt{rec}^{\mathbb{N}}\,\big(\texttt{rec}^{\mathbb{N}}\,\texttt{true}\,(\lambda\_\_.\texttt{false})\big)\,\big(\lambda\_f.\texttt{rec}^{\mathbb{N}}\,\texttt{false}\,(\lambda x\_.f\,x)\big)\,M\,N$$

$$\dfrac{\Gamma \vdash M : \texttt{nat} \qquad \Gamma \vdash N : \texttt{nat}}{\Gamma \vdash M = N : \texttt{bool}}$$

$$M - N \equiv \texttt{rec}^{\mathbb{N}}\,(\lambda x.x)\,\big(\lambda\_f.\texttt{rec}^{\mathbb{N}}\,0\,(\lambda x\_.f\,x)\big)\,N\,M \qquad \dfrac{\Gamma \vdash M : \texttt{nat} \qquad \Gamma \vdash N : \texttt{nat}}{\Gamma \vdash M - N : \texttt{nat}}$$

$$M < N \equiv \mathsf{S}\,M - N = 0 \qquad \dfrac{\Gamma \vdash M : \texttt{nat} \qquad \Gamma \vdash N : \texttt{nat}}{\Gamma \vdash M < N : \texttt{bool}}$$

3 Diller-Nahm interpretation

We define in this section the Diller-Nahm variant [2] of the Dialectica interpretation [5]. We present the interpretation as an explicit translation from proofs to programs and since we work with natural deduction we handle contexts similarly to Pédrot [9, 10].

We define some notations that will make the interpretation easier to read. First, since we need an interpretation for the *ex falso quodlibet* principle, we introduce dummy terms at each type:

$$\Box_{\sigma\to\tau} \equiv \lambda\_.\Box_\tau \qquad \Box_{\sigma\times\tau} \equiv \langle\Box_\sigma, \Box_\tau\rangle \qquad \Box_1 \equiv \langle\rangle \qquad \Box_{\sigma^*} \equiv \texttt{nil}$$

Second, since we work in the Diller-Nahm variant of the Dialectica interpretation in which several witnesses can be collected for a single formula, we extensively use an encoding of sets as lists. Note that we impose no restriction on this encoding, so that a single element can appear several times and the order of elements is of no importance. We use set notations as follows:

$$\{\sigma\} \equiv \sigma^* \qquad \{M_1; M_2; \ldots; M_n\} \equiv M_1 :: M_2 :: \cdots :: M_n :: \texttt{nil}$$

$$\dfrac{\Gamma \vdash M_1 : \sigma \qquad \Gamma \vdash M_2 : \sigma \qquad \cdots \qquad \Gamma \vdash M_n : \sigma}{\Gamma \vdash \{M_1; M_2; \ldots; M_n\} : \{\sigma\}}$$

$$M \cup N \equiv \texttt{rec}\,M\,(\lambda x\_z.x :: z)\,N \qquad \dfrac{\Gamma \vdash M : \{\sigma\} \qquad \Gamma \vdash N : \{\sigma\}}{\Gamma \vdash M \cup N : \{\sigma\}}$$

$$\{N \mid x \in M \wedge P\} \equiv \mathtt{rec}\,\mathtt{nil}\,(\lambda x\_z.\mathtt{if}\,P\,\mathtt{then}\,N :: z\,\mathtt{else}\,z)\,M$$

$$\frac{\Gamma \vdash M : \{\sigma\} \qquad \Gamma, x : \sigma \vdash N : \tau \qquad \Gamma, x : \sigma \vdash P : \mathtt{bool}}{\Gamma \vdash \{N \mid x \in M \wedge P\} : \{\tau\}}$$

$$\{N \mid x \in M\} \equiv \{N \mid x \in M \wedge \mathtt{true}\} \qquad \frac{\Gamma \vdash M : \{\sigma\} \qquad \Gamma, x : \sigma \vdash N : \tau}{\Gamma \vdash \{N \mid x \in M\} : \{\tau\}}$$

$$\bigcup M \equiv \mathtt{rec}\,\mathtt{nil}\,(\lambda x\_z.x \cup z)\,M \qquad \frac{\Gamma \vdash M : \{\{\sigma\}\}}{\Gamma \vdash \bigcup M : \{\sigma\}}$$

$$\forall x \in M,\ N \equiv \mathtt{rec}\,\mathtt{true}\,(\lambda x\_z.N \,\&\, z)\,M \qquad \frac{\Gamma \vdash M : \{\sigma\} \qquad \Gamma, x : \sigma \vdash N : \mathtt{bool}}{\Gamma \vdash \forall x \in M,\ N : \mathtt{bool}}$$

$$[0; M[\ \equiv \mathtt{rec}^{\mathbb{N}}\,\mathtt{nil}\,(\lambda xy.x :: y)\,M \qquad \frac{\Gamma \vdash M : \mathtt{nat}}{\Gamma \vdash [0; M[\ : \{\mathtt{nat}\}}$$

In order to ensure the preservation of computations via the Dialectica interpretation, Pédrot axiomatized abstract multisets via a series of laws that these should satisfy. Here we work with a concrete implementation of these abstract multisets as lists. To simplify the presentation we do not define any reduction on proofs, and therefore we do not aim at preservation of computations. Consequently we do not define our implementation of sets in such a way that they satisfy the monadic and distributive laws of Pédrot. Choosing carefully the implementation should allow to satisfy some of these laws. Some other (in particular commutativity of union) would be more technical to implement in a terminating language.

The Dialectica interpretation provides a computational interpretation of proofs. Therefore, the meaning of such an interpretation is defined up to equivalence of programs: witnesses of formulas are equivalence classes of programs. In our case this equivalence is β-equivalence $=_\beta$, defined as the reflexive symmetric transitive contextual closure of β-reduction $\to_\beta$. When this is clear from the context we will describe an equivalence class by one of its representatives.

Since we work in the context of first-order logic, the whole interpretation is parameterized by an interpretation of the function and predicate symbols. In the general case, the type of programs interpreting first-order terms should be a parameter of the interpretation as well. However, since in the present work we only consider arithmetic and its extensions, we fix this type to be `nat`, the type of natural numbers. The interpretation is therefore parameterized by the interpretations of function and predicate symbols. The interpretation of a function symbol f of arity n is a program:

$$f^\bullet : \mathtt{nat} \to \ldots \to \mathtt{nat} \to \mathtt{nat}\ (n \text{ arguments})$$

and the interpretation of a predicate symbol P of arity n is a set:

$$P^\bullet \subseteq \mathtt{nat}^n_{/=_\beta}$$

of n-tuples of (equivalence classes of) programs of type `nat`, where $\mathtt{nat}_{/=_\beta}$ denotes the set of equivalence classes of terms of type `nat`. This interpretation is extended to all first-order terms as follows, so that $FV\,(t) = FV\,(t^\bullet)$:

$$x^\bullet \equiv x \qquad (f\,(t_1, \ldots, t_n))^\bullet \equiv f^\bullet\,t_1^\bullet \ldots t_n^\bullet$$

Following Pédrot we define for each formula A the type of its witnesses $\underline{A}$ (Pédrot's $\mathbb{W}\,(A)$) and the type of its counterwitnesses $\overline{A}$ (Pédrot's $\mathbb{C}\,(A)$). Since we use a non-dependently-typed language, the types of witnesses and counterwitnesses of atomic predicates $P\,(t_1, \ldots, t_n)$ cannot depend on $t_1^\bullet, \ldots, t_n^\bullet$, even though the truth of these predicates does depend on them.

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
 & \textbf{witnesses} & & & \textbf{counterwitnesses} & \\
\underline{P\left(t_1,\dots,t_n\right)} & \equiv & 1 & \overline{P\left(t_1,\dots,t_n\right)} & \equiv & 1 \\
\underline{\bot} & \equiv & 1 & \overline{\bot} & \equiv & 1 \\
\underline{\neg A} & \equiv & \underline{A} \to \{\overline{A}\} & \overline{\neg A} & \equiv & \underline{A} \\
\underline{A \Rightarrow B} & \equiv & (\underline{A} \to \underline{B}) \times \left(\underline{A} \to \overline{B} \to \{\overline{A}\}\right) & \overline{A \Rightarrow B} & \equiv & \underline{A} \times \overline{B} \\
\underline{\forall x\, A} & \equiv & \mathtt{nat} \to \underline{A} & \overline{\forall x\, A} & \equiv & \mathtt{nat} \times \overline{A}
\end{array}
$$

Figure 5 Types of witnesses and counterwitnesses of formulas.

The interpretation itself therefore does not depend on the interpretations $P^\bullet$ of predicates P, but the correctness of the interpretation will be expressed via an orthogonality relation afterwards. The types of witnesses and counterwitnesses of formulas are defined inductively in figure 5.

We now lift this interpretation of formulas to an interpretation of proofs, so that a proof of the sequent $\vec{p} : \vec{A} \vdash \pi : A$ (where $\vec{p} : \vec{A} \equiv p_1 : A_1, \dots, p_n : A_n$) is interpreted as a collection of programs with the following types, where $FV\left(\vec{A}, A\right) \subseteq \vec{x}$:

$$
\begin{array}{lc}
 & \vec{x} : \mathtt{nat}, \vec{p} : \underline{\vec{A}} \vdash \pi_{p_1} : \overline{A} \to \{\overline{A_1}\} \\
\vec{x} : \mathtt{nat}, \vec{p} : \underline{\vec{A}} \vdash \pi^\bullet : \underline{A} & \vdots \\
 & \vec{x} : \mathtt{nat}, \vec{p} : \underline{\vec{A}} \vdash \pi_{p_n} : \overline{A} \to \{\overline{A_n}\}
\end{array}
$$

The interpretation of proofs is given in figure 6, where some type superscripts have been added to ease the reading. Note that $(\pi_1\, \pi_2)_{p_j}$ involves the union of counterwitnesses comming from both π_1 and π_2.

$$
\begin{array}{rcl@{\quad}rcl}
p_i^\bullet & \equiv & p_i & {p_i}_{p_j} & \equiv & \begin{cases} \lambda q^{\overline{A_i}}.\, \{q\} & \text{if } j = i \\ \lambda\_^{\overline{A_i}}.\, \{\} & \text{if } j \neq i \end{cases} \\
\left(\bot_A\left(\pi\right)\right)^\bullet & \equiv & \Box_{\underline{A}} & \left(\bot_A\left(\pi\right)\right)_{p_j} & \equiv & \lambda\_^{\overline{A}}.\pi_{p_j}\, \langle\rangle \\
\left(\neg_i\left(\pi\right)\right)^\bullet & \equiv & \lambda x.\pi^\bullet.2\, x\, \langle\rangle & \left(\neg_i\left(\pi\right)\right)_{p_j} & \equiv & \lambda q^{\overline{\neg A}}.\pi_{p_j}\, \langle q, \langle\rangle\rangle \\
\left(\neg_e\left(\pi\right)\right)^\bullet & \equiv & \langle \lambda\_.\, \langle\rangle\, , \lambda x.\lambda\_.\pi^\bullet\, x \rangle & \left(\neg_e\left(\pi\right)\right)_{p_j} & \equiv & \lambda q^{\overline{A \Rightarrow \bot}}.\pi_{p_j}\, q.1 \\
\left(\lambda p_{n+1}.\pi\right)^\bullet & \equiv & \left\langle \lambda p_{n+1}.\pi^\bullet, \lambda p_{n+1}.\pi_{p_{n+1}} \right\rangle & \left(\lambda p_{n+1}.\pi\right)_{p_j} & \equiv & \lambda q^{\overline{A_{n+1} \Rightarrow B}}.\, \left(\lambda p_{n+1}.\pi_{p_j}\right) q.1\, q.2 \\
\left(\pi_1\, \pi_2\right)^\bullet & \equiv & \pi_1^\bullet.1\, \pi_2^\bullet & \left(\pi_1\, \pi_2\right)_{p_j} & \equiv & \lambda q^{\overline{B}}.\, \left({\pi_1}_{p_j}\, \langle \pi_2^\bullet, q\rangle\right) \\
 & & & & & \cup \left(\bigcup \left\{ {\pi_2}_{p_j}\, r \,\middle|\, r \in \pi_1^\bullet.2\, \pi_2^\bullet\, q \right\}\right) \\
\left(\lambda x.\pi\right)^\bullet & \equiv & \lambda x.\pi^\bullet & \left(\lambda x.\pi\right)_{p_j} & \equiv & \lambda q^{\overline{\forall x\, A}}.\, \left(\lambda x.\pi_{p_j}\right) q.1\, q.2 \\
\left(\pi\, t\right)^\bullet & \equiv & \pi^\bullet\, t^\bullet & \left(\pi\, t\right)_{p_j} & \equiv & \lambda q^{\overline{A[t/x]}}.\pi_{p_j}\, \langle t^\bullet, q\rangle
\end{array}
$$

Figure 6 Diller-Nahm interpretation of proofs.

$$\begin{aligned}
M \perp_{P(t_1,\ldots,t_n)} N &\text{ iff } (t_1^\bullet,\ldots,t_n^\bullet) \in P^\bullet \\
M \perp_{\perp} N &\text{ is false} \\
M \perp_{\neg A} N &\text{ iff } N \perp_A P \text{ is false for some } P \in M\,N \\
M \perp_{A \Rightarrow B} N &\text{ iff } N.1 \perp_A P \text{ for all } P \in M.2\,N.1\,N.2 \text{ implies } M.1\,N.1 \perp_B N.2 \\
M \perp_{\forall x\, A} N &\text{ iff } M\,N.1 \perp_{A[N.1/x]} N.2
\end{aligned}$$

Figure 7 Definition of the orthogonality relation.

As explained before, the first-order structure is not reflected in this interpretation, as $\langle\rangle$ has the type interpreting any atomic formula, including $\perp$. Similarly, $\lambda\_.\,\{\}$ has the type interpreting $\neg A$ for any formula A. Therefore, typing is not sufficient to ensure correctness of the interpretation, both because of the first-order aspect of our logic and because the Diller-Nahm variant manipulates "false" witnesses and counterwitnesses ($M \perp_{\neg A} N$ only requires $\neg\,(N \perp_A P)$ for *some* $P \in M\,N$). In order to express correctness, we define an orthogonality relation between (equivalence classes of) witnesses and (equivalence classes of) counterwitnesses of formulas that corresponds Gödel's A_D. When viewing the interpretation as games, this orthogonality relation represents the outcome of the game: a witness is orthogonal to a counterwitness whenever the witness wins against the counterwitness. In the following, if $N : \{\sigma\}$ then we write "$M \in N$" as a shorthand for "$N =_\beta \{N_1; \ldots; N_n\}$ and $M =_\beta N_i$ for some i". If $M : \underline{A}$ and $N : \overline{A}$, then $M \perp_A N$ is defined in figure 7.

With this orthogonality relation at hand we can now formulate the correctness theorem of the interpretation:

▶ **Theorem 1.** *If $\vec{p} : \vec{A} \vdash \pi : B$ is derivable in first-order logic and $FV\left(\vec{A}, B\right) \subseteq \vec{x}$, then for all $\vec{X} : \vec{\mathtt{nat}}$, $\vec{P} : \underline{\vec{A}}$ and $Q : \overline{B}$:*

$$\textit{if for all } i \textit{ and for all } R \in \pi_{p_i}\left[\vec{P}, \vec{X}/\vec{p}, \vec{x}\right] Q \textit{ we have } P_i \perp_{A_i} R, \textit{ then } \pi^\bullet\left[\vec{P}, \vec{X}/\vec{p}, \vec{x}\right] \perp_B Q$$

Proof. By induction on π, see Pédrot [10]. ◀

This theorem states that whenever $\vec{P}$ are witnesses of $\vec{A}$ and Q is a counterwitness of B, if for each i P_i wins against every counterwitness of A_i computed by π_{p_i} (using $\vec{P}$ and Q), then the witness of B computed by $\pi^\bullet$ (using $\vec{P}$) wins against Q.

Arithmetic

The interpretations of function symbols of arithmetic are the expected implementations of operations on natural numbers in our programming language, and the only predicate of arithmetic, equality, is interpreted as:

$$=^\bullet \equiv \{(M, M) \mid M : \mathtt{nat}\}$$

The interpretation of the axioms of arithmetic, except for induction, is relatively straightforward and is given in figure 8. The case of induction is more complex. Its interpretation is of the form $\langle \lambda p.\, \langle \lambda q.a, \lambda qr.c\,[r.1, r.2/z, r]\rangle, \lambda ps.d\,[s.1, s.2.1, s.2.2/q, z, r]\rangle$, where we define a, b, c and d as follows:

$\forall x\,(x = x)$	$\lambda\_.\,\langle\rangle$
$\forall xy\,(x = y \Rightarrow A\,[x/z] \Rightarrow A\,[y/z])$	$\lambda\_\_.\,\langle\lambda\_.\,(\lambda p.p)^{\bullet}\,,\lambda\_\_.\,\{\langle\rangle\}\rangle$
$\forall x\,(x + 0 = x)$	$\lambda\_.\,\langle\rangle$
$\forall xy\,(x + \mathtt{S}\,y = \mathtt{S}\,(x + y))$	$\lambda\_.\,\langle\rangle$
$\forall x\,(x \times 0 = 0)$	$\lambda\_.\,\langle\rangle$
$\forall xy\,(x \times \mathtt{S}\,y = x \times y + x)$	$\lambda\_.\,\langle\rangle$
$\forall x\,\neg\,(\mathtt{S}\,x = 0)$	$\lambda\_\_.\,\{\langle\rangle\}$
$\forall xy\,(\mathtt{S}\,x = \mathtt{S}\,y \Rightarrow x = y)$	$\lambda\_\_.\,\langle\lambda\_.\,\langle\rangle\,,\lambda\_\_.\,\{\langle\rangle\}\rangle$

Figure 8 Interpretation of the axioms of arithmetic, except induction.

$$\begin{array}{ll}
p : \underline{A\,[\mathtt{0}/x]}, q : \underline{\forall x\,(A \Rightarrow A\,[\mathtt{S}\,x/x])} & \vdash a : \underline{\forall x\,A} \\
p : \underline{A\,[\mathtt{0}/x]}, q : \underline{\forall x\,(A \Rightarrow A\,[\mathtt{S}\,x/x])}, z : \mathtt{nat}, r : \overline{A} & \vdash b : \mathtt{nat} \to \{\overline{A}\} \\
p : \underline{A\,[\mathtt{0}/x]}, q : \underline{\forall x\,(A \Rightarrow A\,[\mathtt{S}\,x/x])}, z : \mathtt{nat}, r : \overline{A} & \vdash c : \left\{\overline{A\,[\mathtt{0}/x]}\right\} \\
p : \underline{A\,[\mathtt{0}/x]}, q : \underline{\forall x\,(A \Rightarrow A\,[\mathtt{S}\,x/x])}, z : \mathtt{nat}, r : \overline{A} & \vdash d : \left\{\overline{\forall x\,(A \Rightarrow A\,[\mathtt{S}\,x/x])}\right\}
\end{array}$$

$$\begin{aligned}
a &\equiv \mathtt{rec}^{\mathbb{N}}\,p\,(\lambda xy.\,(q\,x)\,.1\,y) \\
b &\equiv \mathtt{rec}^{\mathbb{N}}\,\{r\}\left(\lambda xy.\bigcup\{(q\,(z - (\mathtt{S}\,x)))\,.2\,(a\,(z - (\mathtt{S}\,x)))\,g\,|\,g \in y\}\right) \\
c &\equiv b\,z \\
d &\equiv \bigcup\{\{\langle n, \langle a\,n, g\rangle\rangle\,|\,g \in b\,(z - (\mathtt{S}\,n))\}\,|\,n \in [0; z[\}
\end{aligned}$$

a performs standard recursion on natural numbers, while $b\,0, \ldots, b\,z$ compute sets of counterwitnesses to $A\,[n/x]$ for n from z down to $\mathtt{0}$, starting with counterwitness r for $A\,[z/x]$. The following lemma is the core of correctness of this interpretation:

▶ **Lemma 2.** *Let $P : \underline{A\,[\mathtt{0}/z]}$, $Q : \underline{\forall x\,(A\,[x/z] \Rightarrow A\,[\mathtt{S}\,x/z])}$, $Z : \mathtt{nat}$ and $R : \overline{A}$, and let $A \equiv a\,[P, Q/p, q]$, $B \equiv b\,[P, Q, Z, R/p, q, z, r]$, $C \equiv c\,[P, Q, Z, R/p, q, z, r]$ and $D \equiv d\,[P, Q, Z, R/p, q, z, r]$.*

$$\begin{aligned}
&\textit{if for any } S \in C \textit{ we have } P\perp_{A[\mathtt{0}/z]}S \\
&\quad\textit{and if for any } S \in D \textit{ we have } Q\perp_{\forall x(A[x/z]\Rightarrow A[\mathtt{S}\,x/z])}S \\
&\qquad\qquad\textit{then for any } N \in [0; \mathtt{S}\,Z[\;\textit{ and } S \in B\;(Z - N)\;\textit{ we have } A\,N\perp_{A[N/z]}S
\end{aligned}$$

Proof. By induction on N from 0 to Z. ◀

With this lemma at hand it becomes easy to prove correctness of the interpretation of arithmetic:

▶ **Theorem 3.** *For any axiom A of arithmetic interpreted as $a : \underline{A}$, for all $M : \overline{A}$, $a\perp_A M$.*

4 Double-Negation Shift

This section is devoted to the main contributions of this paper: bar recursive interpretations of the double-negation shift (DNS) principle in the Diller-Nahm variant of the Dialectica interpretation.

The negative translation provides an interpretation of classical logic into intuitionistic logic, at the expense of changing the formula proven: if A is provable in classical logic then $A^{\neg}$ is provable in intuitionistic logic, where $A^{\neg}$ is the negative translation of A defined inductively on the structure of A. This negative translation can be extended to arithmetic because the negative translation of every axiom of arithmetic is derivable in intuitionistic arithmetic. Combined with Friedman's translation [4] one can even eliminate classical logic without changing the formula in the case of Π^0_2 formulas, which means that classical arithmetic is conservative over intuitionistic arithmetic for this class of formulas.

When it comes to richer theories, however, negative translation can fail. In particular, the negative translation of the axiom of choice fails to be an intuitionistic consequence of the axiom of choice. In order to recover the negative translation on such theories, one can use the DNS principle:

$$\forall x \neg\neg A \Rightarrow \neg\neg\forall x\, A$$

Using this principle, $A^{\neg}$ becomes an intuitionistic consequence of A for any formula A. This allows the extension of the negative translation from plain logic to any theory. In particular, negative translation can then be applied to classical analysis using DNS on natural numbers. Combining the negative translation with an extension of the Dialectica interpretation to the DNS principle on natural numbers, one obtains a computational interpretation of classical analysis.

In the past, several versions of bar recursion have been used to provide a Dialectica interpretation of the DNS principle, including Spector's original bar recursion [11], as well as Oliva-Powell's demand-driven variant [8], inspired by Berardi-Bezem-Coquand's demand-driven operator [1]. In the following sections we give new versions of bar recursion that are compatible with the Diller-Nahm variant of the Dialectica interpretation. This extends the Dialectica interpretation to classical theories with non-decidable atomic predicates.

4.1 Diller-Nahm Bar Recursion

In this section we define an adaptation of Spector's original bar recursion to the Diller-Nahm setting. Bar recursion builds incrementally finite aproximations to a witness of $\forall x\, A\,(x)$. These finite aproximations are sequences of witnesses for $A\,(0)\,, \ldots, A\,(n-1)$ for some $n \in \mathbb{N}$. We encode these approximations as lists of programs of type $\underline{A}$, so that if for $i < n$, $a_i : \underline{A}$ is a witness for $A\,(i)$, then $a_{n-1} :: \ldots :: a_1 :: a_0 :: \mathtt{nil}$ is such an approximation. We define the following notation for the length of an approximation:

$$|M| \equiv \mathtt{rec}\,\mathtt{0}\,(\lambda\_\_z.\mathtt{S}\,z)\,M \qquad \frac{\Gamma \vdash M : \sigma^*}{\Gamma \vdash |M| : \mathtt{nat}}$$

We also define the "dummy" completion of an approximation $a_{n-1} :: \ldots :: a_1 :: a_0 :: \mathtt{nil}$ into a full sequence mapping $i < n$ to a_i and $i \geq n$ to $\Box_\sigma$:

$$M^+ \equiv \lambda n.\mathtt{rec}\,\Box_\sigma\,(\lambda xyz.\mathtt{if}\,n = |y|\;\mathtt{then}\,x\,\mathtt{else}\,z)\,M \qquad \frac{\Gamma \vdash M : \sigma^*}{\Gamma \vdash M^+ : \mathtt{nat} \to \sigma}$$

Finally, each relation between $\mathtt{nat}$ and σ can be turned into a function from $\mathtt{nat}$ to $\{\sigma\}$:

$$\widehat{M} \equiv \lambda x.\, \{y.2 \,|\, y \in M \wedge y.1 = x\} \qquad \begin{array}{c} \Gamma \vdash M : \{\mathtt{nat} \times \sigma\} \\ \hdashline \Gamma \vdash \widehat{M} : \mathtt{nat} \to \{\sigma\} \end{array}$$

Computing the required type for a program interpreting the DNS principle gives:

$$\begin{aligned} \underline{\forall x\, \neg\neg A \Rightarrow \neg\neg\forall x\, A} \equiv {} & (\underline{\forall x\, \neg\neg A} \to \underline{\neg\forall x\, A} \to \{\mathtt{nat} \to \underline{A}\}) \\ & \times \big(\underline{\forall x\, \neg\neg A} \to \underline{\neg\forall x\, A} \to \big\{\mathtt{nat} \times \big(\underline{A} \to \big\{\overline{A}\big\}\big)\big\}\big) \end{aligned}$$

$$\underline{\forall x\, \neg\neg A} \equiv \mathtt{nat} \to \big(\underline{A} \to \big\{\overline{A}\big\}\big) \to \{\underline{A}\} \qquad \underline{\neg\forall x\, A} \equiv (\mathtt{nat} \to \underline{A}) \to \big\{\mathtt{nat} \times \overline{A}\big\}$$

The interpretation is therefore a pair of two programs a and b with respective types:

$$\begin{aligned} a &: \big(\mathtt{nat} \to \big(\underline{A} \to \big\{\overline{A}\big\}\big) \to \{\underline{A}\}\big) \to \big((\mathtt{nat} \to \underline{A}) \to \big\{\mathtt{nat} \times \overline{A}\big\}\big) \to \{\mathtt{nat} \to \underline{A}\} \\ b &: \big(\mathtt{nat} \to \big(\underline{A} \to \big\{\overline{A}\big\}\big) \to \{\underline{A}\}\big) \to \big((\mathtt{nat} \to \underline{A}) \to \big\{\mathtt{nat} \times \overline{A}\big\}\big) \to \big\{\mathtt{nat} \times \big(\underline{A} \to \big\{\overline{A}\big\}\big)\big\} \end{aligned}$$

Unfolding the definitions we see that in order to satisfy correctness, a and b should satisfy for any arguments $P : \mathtt{nat} \to \big(\underline{A} \to \big\{\overline{A}\big\}\big) \to \{\underline{A}\}$ and $Q : (\mathtt{nat} \to \underline{A}) \to \big\{\mathtt{nat} \times \overline{A}\big\}$:

$$\begin{aligned} & \text{if for any } M \in b\,P\,Q,\, P \perp_{\forall x \neg\neg A} M \\ & \qquad\qquad\qquad \text{then there is some } M \in a\,P\,Q \text{ such that for all } N \in Q\,M,\, M \perp_{\forall x\, A} N \end{aligned}$$

In other words, we should put in $b\,P\,Q$ counterwitnesses of $\forall x \neg\neg A$ such that whenever P wins against all of them, we can use that hypothesis to ensure that $a\,P\,Q$ contains at least an element M that wins against every element of $Q\,M$. The elements of $a\,P\,Q$ will be "dummy" completions of finite approximations, that is, they will be of the form S^+ for $S : \underline{A}^*$. Given such a finite approximation S, if $Q\,S^+$ contains some $\langle N, R\rangle$ then $S^+\,N$ should win against R. But if $N \geq |S|$ then $S^+\,N$ is the dummy value $\square_{\underline{A}}$, meaning that S is not a sufficiently precise approximation in order to be correct. The idea is then to extend S with a value at point $|S|$, using $P\,|S|$ (which is a witness for $\neg\neg A\,[|S|\,/x]$) and continue the process. Computation stops when a sufficiently precise approximation is found, that is, an approximation S such that for all $\langle N, R\rangle \in Q\,S^+$, we have $N < |S|$.

Our variant $\mathtt{dnbr}$ of bar recursion is an operator that, given an approximation in $\underline{A}^*$, computes extensions of this approximation in such a way that at least one of these extensions is correct. Then a is obtained by running this operator on the empty approximation. In order to distinguish sufficiently precise approximations we use the notation:

$$S \Subset Q \equiv \forall z \in Q\,S^+,\; z.1 < |S| \qquad \begin{array}{c} \Gamma \vdash Q : (\mathtt{nat} \to \sigma) \to \{\mathtt{nat} \times \tau\} \qquad \Gamma \vdash S : \sigma^* \\ \hdashline \Gamma \vdash S \Subset Q : \mathtt{bool} \end{array}$$

In order to facilitate the reading we also use the notation:

$$\mathtt{let}\, f = M \,\mathtt{in}\, N \equiv (\lambda f.N)\, M \qquad \begin{array}{c} \Gamma \vdash M : \sigma \qquad \Gamma, f : \sigma \vdash N : \tau \\ \hdashline \Gamma \vdash \mathtt{let}\, f = M \,\mathtt{in}\, N \equiv (\lambda f.N)\, M : \tau \end{array}$$

We now extend our programming language with this operator and its reduction rule:

$$\begin{aligned} & \mathtt{dnbr} : \big(\mathtt{nat} \to \big(\underline{A} \to \big\{\overline{A}\big\}\big) \to \{\underline{A}\}\big) \to \big((\mathtt{nat} \to \underline{A}) \to \big\{\mathtt{nat} \times \overline{A}\big\}\big) \to \underline{A}^* \to \{\underline{A}^*\} \\ & \mathtt{dnbr}\, P\, Q\, S \to_\beta \{S\} \cup \mathtt{if}\, S \Subset Q \,\mathtt{then}\, \{\} \\ & \qquad\qquad\qquad\qquad \mathtt{else}\,\mathtt{let}\, f = \lambda x.\mathtt{dnbr}\, P\, Q\, (x :: S) \;\mathtt{in} \\ & \qquad\qquad\qquad\qquad\quad \bigcup \Big\{ f\, x \,\Big|\, x \in P\,|S|\, \Big(\lambda y. \bigcup \Big\{ \widehat{Q\, t^+}\, |S| \,\Big|\, t \in f\, y \wedge t \Subset Q \Big\}\Big)\Big\} \end{aligned}$$

Observe that given P, Q and an approximation S as input, if for every $\langle N, R\rangle \in Q\,S^+$ we have $N < |S|$ then $S \Subset Q$ so S is a potentially correct approximation and the recursive calls stop. Otherwise, new recursive calls are made on extensions of S with values obtained from $P\,|S|$.

A fundamental result about this operator is that for any P, Q and S, $\texttt{dnbr}\,P\,Q\,S$ has a normal form. Otherwise, there would be an infinite sequence of programs $M_0, M_1, \dots$ such that for any k there exists $\langle N, R\rangle \in Q\,(M_{k-1} :: \dots :: M_1 :: M_0 :: \texttt{nil})^+$ with $N > k$. But continuity of $\lambda f.\,\{x.1 \,|\, x \in Q\,f\} : (\texttt{nat} \to \underline{A}) \to \{\texttt{nat}\}$ implies that the sequence:

$$\left\{x.1 \,\middle|\, x \in Q\,\texttt{nil}^+\right\}, \left\{x.1 \,\middle|\, x \in Q\,(M_0 :: \texttt{nil})^+\right\}, \dots, \\ \left\{x.1 \,\middle|\, x \in Q\,(M_{k-1} :: \dots :: M_1 :: M_0 :: \texttt{nil})^+\right\}, \dots$$

is ultimately constant, hence the contradiction.

With this operator at hand we can finally define a and b, and therefore the interpretation of the DNS principle in the Diller-Nahm variant of the Dialectica interpretation:

$$a \equiv \lambda pq.\,\left\{s^+ \,\middle|\, s \in \texttt{dnbr}\,p\,q\,\texttt{nil} \wedge s \Subset q\right\}$$
$$b \equiv \lambda pq.\,\left\{\left\langle |s|\,, \lambda x.\bigcup\left\{\widehat{q\,t^+}\,|s| \,\middle|\, t \in \texttt{dnbr}\,p\,q\,(x :: s) \wedge t \Subset q\right\}\right\rangle \,\middle|\, \begin{array}{c} s \in \texttt{dnbr}\,p\,q\,\texttt{nil} \\ \wedge \neg\, s \Subset q \end{array}\right\}$$

Correctness of this interpretation relies on the following lemma:

▶ **Lemma 4.** *Let $P : \texttt{nat} \to \left(\underline{A} \to \left\{\overline{A}\right\}\right) \to \{\underline{A}\}$ and $Q : (\texttt{nat} \to \underline{A}) \to \left\{\texttt{nat} \times \overline{A}\right\}$ and let $D \equiv \texttt{dnbr}\,P\,Q$. If for any $M \in b\,P\,Q$ we have $P \bot_{\forall x \neg\neg A} M$, then there exists a sequence of programs $M_0, \dots, M_{n-1}$ such that if we write $S_i \equiv M_{i-1} :: \dots :: M_0 :: \texttt{nil}$:*

(a) *$D\,\texttt{nil} = D\,S_0 \supseteq D\,S_1 \supseteq \dots \supseteq D\,S_n$*

(b) *for all $i \leq n$, $S \in D\,S_i$ and $N \in Q\,S^+$, if $S \Subset Q$ and $N.1 < i$ then $S_i^+\,N.1 \bot_{A[N.1/x]} N.2$*

(c) *for all $N \in Q\,S_n^+$, $N.1 < n$*

Proof. $S_0 = \texttt{nil}$ trivially satisfies (b). Suppose now $M_0, \dots, M_{k-1}$ already defined, and satisfying (a) and (b). If for all $N \in Q\,S_k^+$, $N.1 < k$, then we choose $n = k$ and we are done since (c) is verified. Otherwise we have:

$$D\,S_k =_\beta \{S_k\} \cup \bigcup\left\{D\,(x :: S_k) \,\middle|\, x \in P\,k\left(\lambda y.\bigcup\left\{\widehat{Q\,t^+}k \,\middle|\, t \in D\,(y :: S_k) \wedge t \Subset Q\right\}\right)\right\}$$

and we have $S_k \in D\,S_k \subseteq C\,\texttt{nil}$, so:

$$\left\langle k, \lambda x.\bigcup\left\{\widehat{Q\,t^+}k \,\middle|\, t \in D\,(x :: S_k) \wedge t \Subset Q\right\}\right\rangle \in b\,P\,Q$$

and therefore by hypothesis:

$$P \bot_{\forall x \neg\neg A} \left\langle k, \lambda x.\bigcup\left\{\widehat{Q\,t^+}k \,\middle|\, t \in D\,(x :: S_k) \wedge t \Subset Q\right\}\right\rangle$$

which means that there must be some:

$$M \in P\,k\left(\lambda x.\bigcup\left\{\widehat{Q\,t^+}k \,\middle|\, t \in D\,(x :: S_k) \wedge t \Subset Q\right\}\right)$$

such that for all:

$$N \in \bigcup\left\{\widehat{Q\,t^+}k \,\middle|\, t \in D\,(M :: S_k) \wedge t \Subset Q\right\}$$

we have $M \perp_{A[k/x]} N$. Define $M_k \equiv M$. We have (a) since $D\, S_{k+1} \equiv D\,(M_k :: S_k) \subseteq D\, S_k$. For (b), let $S \in D\, S_{k+1}$ and $N \in Q\, S^+$ such that $S \Subset Q$ and $N.1 < \mathtt{S}\, k$. If $N.1 < k$ then $S^+_{k+1}\, N.1 =_\beta S^+_k\, N.1$ and we are done by hypothesis on S_k. Otherwise $N.1 =_\beta k$ and $S^+_{k+1}\, N.1 =_\beta M_k$. But then we can prove that:

$$N.2 \in \bigcup \left\{ \widehat{Q\, t^+} k \,\middle|\, t \in D\,(M_k :: S_k) \wedge t \Subset Q \right\}$$

so we obtain $M_k \perp_{A[k/x]} N.2$ by property of M_k. ◀

Finally we can conclude that our interpretation of the DNS principle is correct:

▶ **Theorem 5.** *For any $P : \mathtt{nat} \to (\underline{A} \to \{\overline{A}\}) \to \{\underline{A}\}$ and $Q : (\mathtt{nat} \to \underline{A}) \to \{\mathtt{nat} \times \overline{A}\}$:*

if for any $M \in b\, P\, Q$ we have $P \perp_{\forall x \neg\neg A} M$

then there is some $M' \in a\, P\, Q$ such that for all $N \in Q\, M'$ we have $M' \perp_{\forall x\, A} N$

Proof. Apply the previous lemma and take $M' = S^+_n$. The conclusion follows from properties 2 and 3. ◀

We now discuss the relationship between our operator and Herbrand bar recursion [3] (`hBR`). In the Diller-Nahm interpretation, finite sets appear only in the return type of the "reverse component" of the witnesses of implication. In the Herbrand functional interpretation, handling of standard and non-standard elements requires the use of finite sets in several places. In particular both components of the witnesses of implication are finite sets. Nevertheless, `hBR` is very similar in principle to `dnbr`. But besides technical differences there is also a conceptual difference: `dnbr` handles the argument Q of type $(\mathtt{nat} \to \underline{A}) \to \{\mathtt{nat} \times \overline{A}\}$ more carefully than `hBR`. Indeed, for each t, $Q\, t^+$ is a set of pairs $\langle N, R \rangle$ where R is a potential counterwitness of $A\,[N/x]$. However, since `dnbr` on S extends S at point $|S|$, the only useful counterwitnesses are those of the form $\langle |S|\,, R \rangle$, which are those we get via $\widehat{Q\, t^+}\, |S|$. Conversely, in `hBR`, Q is split in two components: $q \equiv \lambda f.\, \{x.2 \,|\, x \in q\, f\}$ and $\omega \equiv \lambda f.\mathtt{max}\, \{x.1 \,|\, x \in q\, f\}$. Therefore in $q\, f$ the information about the N in $A\,[N/x]$ for which the element of $q\, f$ is a counterwitness is lost. When extending S at point $|S|$, `hBR` considers all the elements of $q\, S^+$, that is, $\{x.2 \,|\, x \in Q\, S^+\}$, instead of only $\{x.2 \,|\, x \in Q\, S^+ \wedge x.1 = |S|\}$ (as `dnbr` does). In that sense `hBR` is less optimal that `dnbr`.

4.2 Diller-Nahm Demand-Driven Bar Recursion

In this last section we present the second contribution of the paper: a demand-driven version of bar recursion in the context of the Diller-Nahm variant of the Dialectica interpretation. The first demand-driven bar recursion was defined by Berardi, Bezem and Coquand in the context of Kreisel's realizability, but this was only recently adapted to the Dialectica interpretation by Oliva and Powell. Here we take inspiration both from Oliva and Powell's operator, and from the version of bar recursion presented in the previous section.

Instead of working on initial segments, demand-driven bar recursion works on partial sequences that may be defined at arbitrary points. We encode these partial functions as lists of points, that is, lists of pairs of a natural number and its image. We define the empty function and the extension of a function with a new value as follows:

$$\epsilon \equiv \mathtt{nil} \qquad \frac{}{\Gamma \vdash \epsilon : (\mathtt{nat} \times \sigma)^*}$$

$$M\,[N \mapsto P] \equiv \langle N, P \rangle :: M \qquad \frac{\Gamma \vdash M : (\mathtt{nat} \times \sigma)^* \qquad \Gamma \vdash N : \mathtt{nat} \qquad \Gamma \vdash P : \sigma}{\Gamma \vdash M\,[N \mapsto P] : (\mathtt{nat} \times \sigma)^*}$$

Similarly to the $\_^{+}$ operator on initial segments, we define the "dummy" completion of a partial function:

$$M^{\dagger} \equiv \lambda n.\mathtt{rec}\,\Box_{\sigma}\,(\lambda x\_z.\mathtt{if}\,x.1 = n\,\mathtt{then}\,x.2\,\mathtt{else}\,z)\,M \qquad \frac{\Gamma \vdash M : (\mathtt{nat} \times \sigma)^{*}}{\Gamma \vdash M^{\dagger} : \mathtt{nat} \to \sigma}$$

We also define the domain of definition of a partial function as follows:

$$\mathtt{dom}\,(M) \equiv \mathtt{rec}\,\{\}\,(\lambda x\_z.\,\{x.1\} \cup z)\,M \qquad \frac{\Gamma \vdash M : (\mathtt{nat} \times \sigma)^{*}}{\Gamma \vdash \mathtt{dom}\,(M) : \{\mathtt{nat}\}}$$

We will also need some more operations on sets:

$$M \setminus N \equiv \{x \,|\, x \in M \wedge \forall y \in N, \neg\,(x = y)\} \qquad \frac{\Gamma \vdash M : \{\mathtt{nat}\} \qquad \Gamma \vdash N : \{\mathtt{nat}\}}{\Gamma \vdash M \setminus N : \{\mathtt{nat}\}}$$

$$M \subseteq N \equiv \forall x \in M, \neg\forall y \in N, \neg\,(x = y) \qquad \frac{\Gamma \vdash M : \{\mathtt{nat}\} \qquad \Gamma \vdash N : \{\mathtt{nat}\}}{\Gamma \vdash M \subseteq N : \mathtt{bool}}$$

Similarly to the $\Subset$ relation on initial segments, sufficiently precise partial functions are the ones satisfying:

$$S \propto Q \equiv \mathtt{dom}\left(Q\,S^{\dagger}\right) \subseteq \mathtt{dom}\,(S) \qquad \frac{\Gamma \vdash Q : (\mathtt{nat} \to \sigma) \to \{\mathtt{nat} \times \tau\} \qquad \Gamma \vdash S : (\mathtt{nat} \times \sigma)^{*}}{\Gamma \vdash S \propto Q : \mathtt{bool}}$$

Finally, we define an operator that picks an element in a non-empty set:

$$\mathtt{choose}\,(M) \equiv M\,?\,\{\mathtt{nil} \mapsto \Box_{\sigma} \,|\, x :: \_ \mapsto x\} \qquad \frac{\Gamma \vdash M : \{\sigma\}}{\Gamma \vdash \mathtt{choose}\,(M) : \sigma}$$

The new demand-driven bar recursion operator is very similar to the previous one, except that if some $N \in Q\,S^{\dagger}$ is such that $N.1$ falls outside the domain of definition of S we do not discard N and extend the function in a linear order as before, but we use this $N.1$ as a hint and extend the current partial function at this point.

We extend our programming language with the following demand-driven bar recursion operator and its reduction rule:

$$\begin{aligned}\mathtt{dnddbr} : \left(\mathtt{nat} \to \left(\underline{A} \to \left\{\overline{A}\right\}\right) \to \{\underline{A}\}\right) \to \left((\mathtt{nat} \to \underline{A}) \to \left\{\mathtt{nat} \times \overline{A}\right\}\right)& \\ \to (\mathtt{nat} \times \underline{A})^{*} \to \left\{(\mathtt{nat} \times \underline{A})^{*}\right\}&\end{aligned}$$

$$\begin{aligned}\mathtt{dnddbr}\,P\,Q\,S \to_{\beta} \{S\} \cup \mathtt{if}\,S \propto Q\,\mathtt{then}\,\{\}& \\ \mathtt{else}\,\mathtt{let}\,f = \lambda zx.\mathtt{dnddbr}\,P\,Q\,(S\,[z \mapsto x])\,\mathtt{in}& \\ \mathtt{let}\,z = \mathtt{choose}\left(\mathtt{dom}\left(Q\,S^{\dagger}\right) \setminus \mathtt{dom}\,(S)\right)\mathtt{in}& \\ \bigcup\left\{f\,z\,x \,\middle|\, x \in P\,z\left(\lambda y.\bigcup\left\{\widehat{Q\,t^{\dagger}}\,z \,\middle|\, t \in f\,z\,y \wedge t \propto Q\right\}\right)\right\}&\end{aligned}$$

Note that while in the previous section the expansions were always performed at $|S|$, here we expand the function at a point chosen in the domain of $Q\,S^{\dagger}$ but outside the domain of S. Since the final goal is to obtain a partial function S such that $S \propto Q$, choosing such a point is more natural and may avoid some useless recursive calls.

As before, the continuity of $\lambda f.\mathtt{dom}\,(Q\,f)$ ensures that $\mathtt{dnddbr}\,P\,Q\,S$ has a normal form for every arguments P, Q and S.

Using this new operator we define the demand-driven interpretation of the DNS principle as follows:

$$a \equiv \lambda pq.\left\{s^{\dagger} \,\middle|\, s \in \mathtt{dnddbr}\,p\,q\,\epsilon \wedge s \propto q\right\}$$

$$b \equiv \lambda pq. \left\{ \begin{array}{l} \texttt{let}\, z = \texttt{choose}\left(\texttt{dom}\left(q\, s^{\dagger}\right) \setminus \texttt{dom}\,(s)\right) \,\texttt{in} \\ \left(\lambda z. \left\langle z, \lambda x. \bigcup \left\{ \widehat{q\, t^{\dagger}}\, z \;\middle|\; \begin{array}{l} t \in \texttt{dnddbr}\, p\, q\, (s\,[z \mapsto x]) \\ \wedge\, t \propto q \end{array} \right\} \right\rangle \right) \end{array} \;\middle|\; \begin{array}{l} s \in \texttt{dnddbr}\, p\, q\, \epsilon \\ \wedge \neg\, s \propto q \end{array} \right\}$$

The following lemma is an adaptation of the one in the previous section:

▶ **Lemma 6.** *Let $P : \texttt{nat} \to \left(\underline{A} \to \left\{\overline{A}\right\}\right) \to \{\underline{A}\}$ and $Q : (\texttt{nat} \to \underline{A}) \to \left\{\texttt{nat} \times \overline{A}\right\}$ and let $D \equiv \texttt{dnddbr}\, P\, Q$. If for any $M \in b\, P\, Q$ we have $P \perp_{\forall x \neg\neg A} M$, then there exists a sequence of programs $N_0, M_0, \ldots, N_{n-1}, M_{n-1}$ such that if we write $S_i \equiv \epsilon\, [N_0 \mapsto M_0] \ldots [N_{i-1} \mapsto M_{i-1}]$:*

(a) $D\, \epsilon = D\, S_0 \supseteq D\, S_1 \supseteq \ldots \supseteq D\, S_n$

(b) *for all $i \leq n$, $S \in D\, S_i$ and $N \in Q\, S^{\dagger}$*
if $S \propto Q$ and $N.1 \in \texttt{dom}\,(S_i)$ then $S_i^{\dagger}\, N.1 \perp_{A[N.1/x]} N.2$

(c) *for all $N \in Q\, S_n^{\dagger}$, $N.1 \in \texttt{dom}\,(S_n)$*

Proof. $S_0 = \epsilon$ trivially satisfies (b). Suppose now $N_0, M_0, \ldots, N_{k-1}, M_{k-1}$ already defined, and satisfying (a) and (b). If for all $N \in Q\, S_k^{\dagger}$, $N.1 \in \texttt{dom}\,(S_k)$, then we choose $n = k$ and we are done since (c) is verified. Otherwise, let $N_k \equiv \texttt{choose}\left(\texttt{dom}\left(Q\, S_k^{\dagger}\right) \setminus \texttt{dom}\,(S_k)\right)$. We have:

$$D\, S_k =_{\beta} \{S_k\} \cup \bigcup \left\{ D\, (S_k\, [N_k \mapsto x]) \;\middle|\; x \in P\, k \left(\lambda x. \bigcup \left\{ \widehat{Q\, t^{\dagger}}\, k \;\middle|\; \begin{array}{l} t \in D\, (S_k\, [N_k \mapsto x]) \\ \wedge\, t \propto Q \end{array} \right\} \right) \right\}$$

and we have $S_k \in D\, S_k \subseteq D\, \epsilon$, so:

$$\left\langle N_k, \lambda x. \bigcup \left\{ \widehat{Q\, t^{\dagger}}\, N_k \;\middle|\; t \in D\, (S_k\, [N_k \mapsto x]) \wedge t \propto Q \right\} \right\rangle \in b\, P\, Q$$

and therefore by hypothesis:

$$P \perp_{\forall x \neg\neg A} \left\langle N_k, \lambda x. \bigcup \left\{ \widehat{Q\, t^{\dagger}}\, N_k \;\middle|\; t \in D\, (S_k\, [N_k \mapsto x]) \wedge t \propto Q \right\} \right\rangle$$

which means that there must be some:

$$M \in P\, N_k \left(\lambda x. \bigcup \left\{ \widehat{Q\, t^{\dagger}} N_k \;\middle|\; t \in D\, (S_k\, [N_k \mapsto x]) \wedge t \propto Q \right\} \right)$$

such that for all:

$$N \in \bigcup \left\{ \widehat{Q\, t^{\dagger}} N_k \;\middle|\; t \in D\, (S_k\, [N_k \mapsto M]) \wedge t \propto Q \right\}$$

we have $M \perp_{A[N_k/x]} N$. Define $M_k \equiv M$. We have (a) since $D\, S_{k+1} \equiv D\, (S_k\, [N_k \mapsto M_k]) \subseteq D\, S_k$. For (b), let $S \in D\, S_{k+1}$ and $N \in Q\, S^{\dagger}$ such that $N.1 \in \texttt{dom}\,(S_{k+1}) = \{N_k\} \cup \texttt{dom}\,(S_k)$. If $N.1 \in \texttt{dom}\,(S_k)$ then $S_{k+1}^{\dagger}\, N.1 =_{\beta} S_k^{\dagger}\, N.1$ and we are done by hypothesis on S_k. Otherwise $N.1 =_{\beta} N_k$ and $S_{k+1}^{\dagger}\, N.1 =_{\beta} M_k$. But then we can prove that:

$$N.2 \in \bigcup \left\{ \widehat{Q\, t^{\dagger}} N_k \;\middle|\; t \in D\, (S_k\, [N_k \mapsto M_k]) \wedge t \propto Q \right\}$$

so we obtain $M_k \perp_{A[N_k/x]} N.2$ by property of M_k. ◀

Correctness of this new interpretation of the DNS principle then follows from this lemma as in the previous section.

Having a demand-driven operator allows for a potentially simpler interpretation, as well as a more natural one. Indeed, there is no reason why the bar recursion operator should rely on a particular ordering of natural numbers (as the non-demand-driven one does) and the intuitive interpretation should be that the operator only computes the witnesses that are necessary, rather than collecting blindly witnesses until there are enough of them. This also opens the possibility of interpreting theories other than arithmetic, where basic objects are not natural numbers and may not have any natural ordering.

It should be noted that while $\texttt{choose}\,(M)$ picks the first element of (the list encoding) M, this property is never used in the proof. The only required property is that if M has at least one element, then $\texttt{choose}\,(M)$ picks an element of M (hence its name). Picking any other element would work as well, and the choice we made seems arbitrary. In the context of the standard Dialectica interpretation, Oliva and Powell did not encounter this because in that case $Q\,S^\dagger$ returns a single pair, and there is only one possible point at which S can be extended. In the Diller-Nahm variant, however, there may be several points in $\texttt{dom}\left(Q\,S^\dagger\right) \setminus \texttt{dom}\,(S)$ and there seems to be no canonical choice. Depending on the case it may be interesting to use some heuristic to choose the point at which S is extended. One may also wonder whether S could be extended at every point in $\texttt{dom}\left(Q\,S^\dagger\right) \setminus \texttt{dom}\,(S)$ simultaneously, but this seems impossible since the correctness proof requires that the sequence of partial functions $S_0, S_1, \ldots$ is totally ordered.

References

1 Stefano Berardi, Marc Bezem, and Thierry Coquand. On the Computational Content of the Axiom of Choice. *Journal of Symbolic Logic*, 63(2):600–622, 1998.

2 Justus Diller and Werner Nahm. Eine Variante zur Dialectica-Interpretation der Heyting-Arithmetik endlicher Typen. *Archiv für mathematische Logik und Grundlagenforschung*, 16:49–66, 1974.

3 Martín Escardó and Paulo Oliva. The herbrand functional interpretation of the double negation shift. *Journal of Symbolic Logic*, 82(2):590–607, 2017.

4 Harvey Friedman. Classically and intuitionistically provably recursive functions. In Gert Müller and Dana Scott, editors, *Higher Set Theory*, volume 669 of *Lecture Notes in Mathematics*, pages 21–27. Springer, 1978.

5 Kurt Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12(3-4):280–287, 1958.

6 Stephen Cole Kleene. On the Interpretation of Intuitionistic Number Theory. *Journal of Symbolic Logic*, 10(4):109–124, 1945.

7 Georg Kreisel. Interpretation of analysis by means of constructive functionals of finite types. In *Constructivity in mathematics: Proceedings of the colloquium held at Amsterdam, 1957*, Studies in Logic and the Foundations of Mathematics, pages 101–128. North-Holland Publishing Company, 1959.

8 Paulo Oliva and Thomas Powell. Bar recursion over finite partial functions. *Annals of Pure and Applied Logic*, 168(5):887–921, 2017.

9 Pierre-Marie Pédrot. A functional functional interpretation. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 77:1–77:10. ACM, 2014.

10 Pierre-Marie Pédrot. *A Materialist Dialectica. (Une Dialectica matérialiste)*. PhD thesis, Paris Diderot University, France, 2015.

11 Clifford Spector. Provably recursive functionals of analysis: a consistency proof of analysis by an extension of principles in current intuitionistic mathematics. In *Recursive Function Theory: Proceedings of Symposia in Pure Mathematics*, volume 5, pages 1–27. American Mathematical Society, 1962.

Dinaturality Meets Genericity: A Game Semantics of Bounded Polymorphism

James Laird
Department of Computer Science, University of Bath, UK

Abstract

We study subtyping and parametric polymorphism, with the aim of providing direct and tractable semantic representations of type systems with these expressive features. The *liveness order* uses the Player-Opponent duality of game semantics to give a simple representation of subtyping: we generalize it to include graphs extracted directly from second-order intuitionistic types, and use the resulting complete lattice to interpret bounded polymorphic types in the style of System $\mathsf{F}_{<:}$, but with a more tractable subtyping relation.

To extend this to a semantics of terms, we use the type-derived graphs as arenas, on which strategies correspond to dinatural transformations with respect to the canonical coercions ("on the nose" copycats) induced by the liveness ordering. This relationship between the interpretation of generic and subtype polymorphism thus provides the basis of the semantics of our type system.

2012 ACM Subject Classification Theory of computation → Program semantics

Keywords and phrases Subtyping, Bounded Polymorphism, Game Semantics, Dinaturality

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.33

1 Introduction and Related Work

Subtype and parametric polymorphism both provide powerful principles for data abstraction. Combining them via bounded quantification increases this expressive power: they may be used to write programs which are generic, but range over a constrained set of types (a program of type $\forall(X <: S).T$ may be instantiated only with a subtype of S). They have been used to develop formal theories of key aspects of object oriented languages such as inheritance [3, 17]. This combination is not without its challenges: for example, discovering type systems in which the fundamental problems, such as typechecking of terms, are efficiently decidable [19].

Our aim is to describe and relate simple, concrete notions of subtype and generic, parametric polymorphism and show that they can work together to give an interpretation of bounded polymorphism which is both tractable and intensional, yielding a formal semantic account of the constraints on behaviour which can be expressed in such a setting. This allows for models which combine bounded polymorphism with computational effects (in particular, state) and, potentially, for semantics-based subtyping theories which capture aspects of program behaviour.

Earlier models of subtyping, and bounded polymorphism in particular, are based on realizability-style interpretations (partial equivalence relations) [2]. These have made an important contribution to the semantic understanding and development of typing systems such as System $\mathsf{F}_{<:}$ [4, 7], but do not give a direct and effective characterization of the subtyping relation. Indeed, the subtyping relation of System $\mathsf{F}_{<:}$ (which they validate) is itself problematic from an algorithmic point of view – in particular, it is undecidable [19]. There is a body of work dedicated to giving typing systems for bounded quantification which are more tractable, but still expressive [12, 23, 10]. As advocated in [5], semantics should be a guide to this search, and this is one of our motivations. We show that our interpretation of subtyping may be used to interpret a typing system which subsumes several proposed

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 33; pp. 33:1–33:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

restrictions of System $\mathsf{F}_{<:}$ by distinguishing the introduction and elimination forms of bounded quantification: in [16] this type system, and its subtyping and type-checking algorithms is studied in more detail.

The nature of the subtype order makes it difficult to capture in a simple and finitary way using semantic structures such as domains: a subtype may be a restriction of its supertype (e.g. signed and unsigned integers) or an extension of it (e.g. record types). This dependence on interaction with consuming contexts can be captured via the Player/Opponent duality of game semantics in the *liveness ordering* [6]. We generalize this ordering to graph structures which may be derived from AJM-style games, directly from types themselves via the subtype ordering, and Hyland-Ong style arenas (which are are a widely-used basis for sequential and concurrent semantics of functional and object-oriented programming languages), and and use its lattice structure to construct an interpretation of bounded quantification.

The game semantics of parametric polymorphism underlying our interpretation is a novel presentation of the well-bracketed second order games model [13, 14], which is based on instantiating pairs of question-answer moves with copycat behaviour, structure which is made more immediate by using second-order types as moves. The glue which binds this to subtyping (and allows for a sound model of bounded quantification) is the *dinaturality* of strategy instantiation with respect to copycat strategies – canonical subtyping coercions which (as we show) are characteristic of the liveness order. Dinaturality has been proposed as a core semantic principle for modelling polymorphism [1] but as a general property is neither preserved by composition, nor possessed by all terms of System F [8]. However, copycat dinaturality is the key to soundly modelling bounded abstraction and instantiation.

1.1 Contribution of this work

This work establishes the relevance and tractability of the calculus for bounded polymorphism described in [16] by giving a concrete denotational semantics for it, in a setting (HO game semantics) which is readily extendable with relevant computational effects such as stateful objects. Giving such a semantics requires the integration of two kinds of polymorphism – subtyping and parametric polymorphism. The more general contribution is to show that this can be achieved in such a setting, using dinaturality and copycat strategies to relate rather diverse elements of game semantics – the liveness order and the well-bracketing condition. Finally, by generalising the liveness order, and showing that it may be described directly at the syntactic level, on types, it is hoped to draw it to wider attention as a way of understanding and studying subtyping.

2 Subtyping Graphs

The game semantic interpretation of subtyping as a *liveness ordering* was introduced by Chroboczek [6] for games presented positionally, as sequences of moves. Here, we generalize it to graph structures of disjoint sets of nodes, which include Hyland-Ong (HO) style arenas [11], which provide a general setting for interpreting types in sequential and concurrent game semantics. In the next section we will describe the derivation of these graphs directly from the subtyping order on second-order types.

▶ **Definition 1.** *A graph-arena is a rooted, directed graph with a partition of its non-root nodes: given as a tuple $(O, P, \triangleright)$ consisting of :*

- *Disjoint sets of nodes O and P, not containing the distinguished root node $\top$.*
- *An edge relation $\triangleright \subseteq (O \cup P \cup \{\top\}) \times (O \cup P)$*

with the following properties:

Well-Foundedness. *There is no infinite chain* $\ldots \rhd m_{i+1} \rhd m_i \rhd \ldots \rhd m_0$
Quasi-Arborescence. *If* $l \rhd m$, $m \rhd n$ *and* $m' \rhd n$ *then* $l \rhd m'$.

$$\begin{tikzcd}
& l \arrow[dl] \arrow[dr, dotted] & \\
m \arrow[dr] & & m' \arrow[dl] \\
& n &
\end{tikzcd}$$

Quasi-Partition. *If* $m \rhd n$ *and* $m' \rhd n$ *then* $m, m' \in O$ *or* $m, m' \in P$

Fixing a graph $U = (O, P, \rhd)$, for any set of nodes $A \subseteq O \cup P$, let $E(A) = \{u \in O \cup P \mid \exists u \in A \cup \{\top\}.u \rhd v\}$ be the set of vertices accessible from A. Writing $\rhd_A$ for the restriction of $\rhd$ to $A \cup \{\top\}$:

▶ **Definition 2.** *A sub-arena of* U *is a non-empty subset* $A \subseteq O \cup P$ *such that* $\top \rhd_A^* a$ *for all* $a \in A$ *(i.e. a root-connected subgraph of* U*).*

▶ **Proposition 3.** $A \subseteq O \cup P$ *is a sub-arena if and only if* $A \subseteq E(A)$.

Proof. Evidently, if A is root-connected, every node in A is connected to one in $A \cup \{\top\}$. The converse follows by Noetherian induction: if a is initial then it is evidently hereditarily enabled in A. Otherwise there exists $a' \in A$ such that $a' \rhd a$. By hypothesis, this is connected to $\top$ through A and hence so is a. ◀

Partitioning $E(A)$ into the sets $P(A) = E(A) \cap P$ and $O(A) = E(A) \cap O$, we now define the liveness order on sub-arenas.

▶ **Definition 4.** *Let* $(\mathcal{S}(U), \preccurlyeq)$ *be the partially ordered set of sub-arenas of* U*, where* $\preccurlyeq$ *is the liveness order:*

$A \preccurlyeq B$ *if and only if* $O(A) \cap B \subseteq A$ *and* $P(B) \cap A \subseteq B$.

In other words, $A \preccurlyeq B$ if all of the P-nodes in A which are accesible from B are already in B, and all O-nodes in B which are accessible from A are in A.

For example AJM-style games are given by sets of plays (alternating sequences over a set of moves partitioned between Opponent and Proponent). Their arena graphs are given by taking O and P to be the sets of plays ending in Proponent and Opponent moves, respectively, with an edge from s to t if the latter has the form sa. In this case $A \preccurlyeq B$ if for any Opponent move a, $s \in A$ and $sa \in B$ implies $sa \in A$, and for any Proponent move b, $s \in B$ and $sb \in A$ implies $sb \in B$. This is the liveness ordering defined in [6].

The following lemma is used to show that this is a well-defined partial order,

▶ **Lemma 5.** *If* $A \preccurlyeq B$ *and* $B \preccurlyeq C$ *then* $E(A) \cap E(C) \subseteq E(B)$.

Proof. By Noetherian induction. Suppose $b \in E(A) \cap E(C)$. If $\top \lhd b$ then $b \in E(B)$ as required. Otherwise, there exist $a \in A$ and $c \in C$ such that $a \rhd b$ and $c \rhd b$, and $a, c \in P$ or $a, c \in O$ by quasi-partition. Supposing the former, by connectedness of A there exists $a' \in A \cup \{\top\}$ such that $a' \rhd a$. By quasi-arboresecence, $a' \rhd c$, and so $c \in E(A) \cap E(C)$. By induction hypothesis, $c \in E(B)$ and so $c \in P(B) \cap C$. Thus $c \in B$ (since $B \preccurlyeq C$) and so $b \in E(B)$ as required. The case where $a, c \in O$ is symmetric. ◀

▶ **Proposition 6.** *$(\mathcal{S}(U), \preccurlyeq)$ is a partial order.*

Proof. Reflexivity is evident.

For transitivity, suppose $A \preccurlyeq B$ and $B \preccurlyeq C$. Then $O(A) \cap C \subseteq O(A) \cap E(C) \cap C$ by connectedness of C
$\subseteq O(A) \cap O(B). \cap C$ (by Lemma 5)
$\subseteq O(A) \cap B$ (since $B \preccurlyeq C$)
$\subseteq A$ (since $A \preccurlyeq B$). By symmetry, $P(C) \cap A \subseteq C$.

For antisymmetry, suppose $A \preccurlyeq B$ and $B \preccurlyeq A$. Then $A = A \cap E(A)$ by connectedness
$\subseteq A \cap E(B)$ by Lemma 5
$= A \cap O(B) \cup P(B) = (A \cap O(B)) \cup (A \cap P(B))$
$\subseteq B \cup B = B$ as $A \preccurlyeq B$ and $B \preccurlyeq A$.

By symmetry, $B \subseteq A$ and hence $A = B$ as required. ◀

In fact $(\mathcal{S}(U), \preccurlyeq)$ is a complete lattice.

▶ **Definition 7.** *The maximal sub-arena of a set $X \subseteq O \cup P$ is $X^\circ = \bigcup\{A \in \mathcal{S}(U) \mid A \subseteq X\}$.*

X° is root-connected – $X^\circ = \bigcup\{A \in \mathcal{S}(U) \mid A \subseteq X\} \subseteq \bigcup\{E(A) \in \mathcal{S}(U) \mid A \subseteq X\} = E(X^\circ)$ – and thus a well-defined sub-arena.

▶ **Lemma 8.** *For any $X \subseteq O \cup P$, $E(X^\circ) \cap X = X^\circ$*

Proof. $X^\circ \subseteq E(X^\circ)$ and $X^\circ \subseteq X$ by definition. Conversely, suppose $x \in X \cap E(X^\circ)$. Then $X^\circ \cup \{x\} \subseteq X$ and $X^\circ \cup \{x\} \subseteq E(X^\circ)$ and so $X^\circ \cup \{x\} \subseteq X^\circ$ – i.e. $x \in X^\circ$ and so $X \cap E(X^\circ) \subseteq X^\circ$ as required. ◀

The $\preccurlyeq$-infimum of a set of sub-arenas $\Delta \subseteq \mathcal{S}(U)$ is the maximal sub-arena of the maximal subset of $\bigcup \Delta$ in which any P-node which has an enabling node in two sub-arenas $A, B \in \Delta$ is in their intersection.

▶ **Definition 9.** *Given $\Delta \subseteq \mathcal{S}(U)$, let*

$$\underline{\bigwedge \Delta} = \{m \in \bigcup \Delta \mid m \in P(A) \cap P(B) \Longrightarrow m \in A \cap B\}$$

and define

$$\bigwedge \Delta = (\underline{\bigwedge \Delta})^\circ.$$

▶ **Proposition 10.** *$\bigwedge \Delta$ is the greatest lower bound of Δ in $\mathcal{S}(U)$.*

Proof. We suppose $A \in \Delta$ and show that $\bigwedge \Delta \preccurlyeq A$. First if $m \in O(\bigwedge \Delta) \cap A$ then $m \in \underline{\bigwedge \Delta}$, as the condition $m \notin P(B) \cap P(C)$ for all $B, C \in \Delta$. So $m \in E(\bigwedge \Delta) \cap \underline{\bigwedge \Delta}$
$= \bigwedge \Delta$ by Lemma 8.
Now suppose $m \in P(A) \cap \bigwedge \Delta$, so that $m \in B \subseteq E(B)$ for some $B \in \Delta$.
Then $m \in P(A) \cap P(B)$, implying that $m \in A \cap B \subseteq A$ as required, since $m \in \underline{\bigwedge \Delta}$.

Now we suppose $C \preccurlyeq A$ for all $A \in \Delta$ and show that $C \preccurlyeq \bigwedge \Delta$. First, if $m \in O(C) \cap \bigwedge \Delta$ then $m \in O(C) \cap A$ for some $A \in \Delta$ with $C \preccurlyeq A$ and so $m \in C$ as required. Now suppose $m \in P(\bigwedge \Delta) \cap C$. Then $m \in P(A) \cap C \subseteq A$ for some $A \in \Delta$, so $A \in \bigcup \Delta$. If $m \in P(A) \cap P(B)$ for $A, B \in \Delta$ (so $C \preccurlyeq A, B$) then $m \in P(A) \cap P(B) \cap (C) \subseteq A \cap B$ – i.e. $m \in P(A) \cap P(B))$ implies $m \in A \cap B$ and thus $m \in \underline{\bigwedge \Delta}$. So $m \in E(\bigwedge \Delta) \cap \underline{\bigwedge \Delta} = \bigwedge \Delta$ by Lemma 8. ◀

3 Arena Graphs from Types

As an example showing the relationship between the liveness ordering and subtyping, (and a step towards a semantics of bounded quantification) we derive arena-graphs directly from the subtyping relation of System $\mathrm{F}_\top$, which is System F (the second-order λ-calculus [9, 21]) extended with with products and a supertype $\top$ [4] – i.e. its set of types is given by the grammar:

$$S ::= \top \mid X \mid S \to S \mid S \times S \mid \forall X.S$$

The $\top$ type allows for a for non-trivial subtyping relation, given by the following derivation rules:

$$\frac{}{T <: \top}\ \textsf{Top} \qquad \frac{}{T <: T}\ \textsf{Refl} \qquad \frac{T <: T' \qquad T' <: T''}{T <: T''}\ \textsf{Trans}$$

$$\frac{S' <: S \qquad T <: T'}{S \to T <: S' \to T'}\ \to \qquad \frac{S <: S' \qquad T <: T'}{S \times T <: S' \times T'}\ \times \qquad \frac{T <: T'}{\forall X.T <: \forall X.T'}\ \forall$$

For example, the standard System F representation of the Booleans – the type $\forall X.X \to X \to X$ has the subtypes $\forall X.\top \to X \to X$, $\forall X.X \to \top \to X$ and $\forall X.\top \to \top \to X$.

3.1 Type Arenas for System $\mathrm{F}_\top$

We now derive an arena-graph in which to interpret System $\mathrm{F}_\top$ subtyping. The sets O and P of O-nodes and P-nodes (respectively) consist of the types given by the grammars:

$$o ::= X \mid \top \to o \mid p \to \top \mid \top \times o \mid o \times \top \mid \forall X.o$$
$$p ::= \top \to p \mid o \to \top \mid \top \times p \mid p \times \top \mid \forall X.p$$

where X ranges over an unbounded set of *type variables*. Note that each such *node-type* m contains exactly one occurrence of a type-variable X, which may be bound or free, so we may write it as $m[X]$. It is an O-node if X occurs "positively" and a P-node if it occurs "negatively".

The edge relation for our second-order type arena-graph is derived from the subtyping order. Let $<\!\cdot$ be the covering relation for its restriction to $O \cup P \cup \{\top\}$ – i.e. $m <\!\cdot\, m'$ if $m <: m'$, $m \neq m'$ and if $m <: m'' <: m'$ then $m'' = m$ or $m'' = m'$.

▶ **Definition 11.** *Let $\vartriangleright$ be the least relation on $(O \cup P \cup \{\top\}) \times (O \cup P)$ such that:*

- *If o is $<:$-minimal (i.e. $o' <: o$ implies $o = o'$) then $\top \vartriangleright o$.*
- *If $p <\!\cdot\, p'$, $p' \vartriangleright o$ and $o <: p'$ then $o \vartriangleright p$.*
- *If $o <\!\cdot\, o'$, $o \vartriangleright p$ and $o' <: p$ then $p \vartriangleright o$.*

In other words, a node m is initial (enabled by $\top$) if its type variable occurs on the right of every arrow ($\to$) in m. Otherwise it has the form $C[m' \to \top]$, for some unique initial node m', and is enabled by any node of the form $C[\top \to n]$, where n is an initial node. Quasi-arboresecence follows from this characterization: while $\vartriangleright$ is not a tree, (e.g. $\top \to (X \times \top) \vartriangleright X \to \top$ and $\top \to (\top \times X) \vartriangleright X \to \top$, both $\top \to (X \times \top)$ and $\top \to (\top \times X)$ are initial. Hence $(O, P, \vartriangleright)$ is an arena – it is a bipartite graph by construction, and it is well-founded, as any chain $\ldots o_{n+1} \vartriangleright p_n \vartriangleright o_n \vartriangleright \ldots \vartriangleright p_1 \vartriangleright o_1$ contains an infinite descending chain $\ldots <: o_{n+1} <: o_n <: \ldots <: o_1$: a straightforward induction establishes that no such chain exists.

We may now define a sub-arena (its *type-arena*) for each type.

▶ **Definition 12.** *Let $\sqsubseteq_\top$ be the least congruence on types such that $S \sqsubseteq_\top \top$ for all types S. Then for any type T, let $[\![T]\!] = \{m \in O \cup P \| \ m \sqsubseteq_\top T\}^\circ$*

In other words, $[\![T]\!]$ is the (root-connected) set of nodes which may be obtained from T by replacing subterms of T with $\top$. For example, the Boolean type $\forall X.X \to X \to X$ denotes the set containing the O-node $\forall X.\top \to X$ and the P-nodes $\forall X.\top \to X \to \top$ and $\forall X.X \to \top$. The edge-relation restricts to these nodes as follows:

- $\top \triangleright \forall X.\top \to \top \to X$, since $\forall X.\top \to \top \to X$ is a $<:$-minimal type.
- $\forall X.\top \to \top \to X \triangleright \forall X.\top \to X \to \top, \forall X.X \to \top$ since $\top \triangleright \forall X.\top \to \top \to X$, and $\forall X.\top \to \top \to X <: \forall X.\top \to X \to \top, \forall X.X \to \top$, $\forall X.\top \to X \to \top, \forall X.X \to \top <: \top$.

yielding the expected graph structure:

$$\begin{tikzcd}
 & \top \arrow[d] & \\
 & \forall X.\top \to \top \to X \arrow[dl] \arrow[dr] & \\
\forall X.X \to \top & & \forall X.\top \to X \to \top
\end{tikzcd}$$

Type-arenas may also be defined compositionally:

- $[\![\top]\!] = \varnothing$, $[\![X]\!] = \{X\}$,
- $[\![S \times T]\!] = [\![S]\!] \times [\![T]\!] \triangleq \{m \times \top \mid m \in S\} \cup \{\top \times m \mid m \in T\}$,
- $[\![S \to T]\!] = [\![S]\!] \to [\![T]\!] \triangleq (\{m \to \top \mid m \in S\} \cup \{\top \to m \mid m \in T\})^\circ$,
- $[\![\forall X.S]\!] = \forall X.M \triangleq \{\forall X.m \mid m \in S\}$.

Using this decomposition, and the soundness of each of the subtyping derivation rules: e.g. if $A' \preccurlyeq A$ and $B \preccurlyeq B'$ then $A \to B \preccurlyeq A' \to B'$, it follows that subtyping is sound with respect to the liveness order:

▶ **Proposition 13.** *If $S <: T$ then $[\![S]\!] \preccurlyeq [\![T]\!]$.*

4 Bounded Quantification

We now describe an interpretation of bounded (universal) quantification types. The standard typing system for the second-order λ-calculus with such types is System $\mathsf{F}_{<:}$ [7, 4, 18], in which terms of type $\forall(X <: S).T$ may only be instantiated with a subtype of S. (Thanks to the presence of a $<:$-greatest type $\top$, System F may be viewed as a proper subsystem of System $\mathsf{F}_{<:}$ by reading unbounded quantification $\forall X.S$ as the bounded quantification $\forall(X <: \top).S$.) However, System $\mathsf{F}_{<:}$ lacks reasonable algorithmic characteristics – in particular, its subtyping relation (and thus also typechecking) is undecidable [19]. The culprit is the rule for subtyping bounded quantification:

$$\frac{\mathcal{E} \vdash T_0 <: S_0 \qquad \mathcal{E}, X <: T_0 \vdash S_1 <: T_1}{\mathcal{E} \vdash \forall(X <: S_0).S_1 <: \forall(X <: T_0).T_1}\ \forall - \mathsf{Orig}$$

where $\mathcal{E}$ is a subtyping context – a sequence of subtyping assumptions $X_1 <: S_1, \ldots, X_n <: S_n$ such that $X_1, \ldots, X_{i-1} \vdash S_i$ for each $1 <: i <: n$.

This problem is also reflected in the difficulty of giving a direct game semantics of the System $\mathsf{F}_{<:}$ subtyping relation. The game semantics for bounded quantification given in [15] constructs a subtyping relation from the derivation system: in particular, the above rule does not respect the liveness in that model. Various modifications to $\mathsf{F}_{<:}$ have been proposed with the aim of giving a more tractable system [12, 23, 10, 5], including three restrictions of this rule.

The first restricts the subtype order to quantified types which have identical bounds:

$$\frac{\mathcal{E}, X <: S \vdash T <: T'}{\mathcal{E} \vdash \forall(X <: S).T <: \forall(X <: S).T'}\ \forall - \mathsf{Fun}$$

This yields a calculus, Kernel $\mathsf{F}_{<:}$, which is well-behaved (subtyping and type-checking are efficiently decidable) at the cost of expressiveness.

The second rule (proposed by Castagna and Pierce as the basis of System $\mathsf{F}^{\top}_{<:}$ [5]) does not use assumptions about the bounds on variables when inferring the subtype relation between the bodies.

$$\frac{\mathcal{E} \vdash T_0 <: S_0 \qquad \mathcal{E}, X <: \top \vdash S_1 <: T_1}{\forall(X <: S_0).S_1 <: \forall(X <: T_0).T_1}\ \forall - \mathsf{Top}$$

It has an expressive subtyping relation with nice properties, including decidability. However, it lacks the minimal typing property, and thus an evident typechecking algorithm.

The third rule [5] uses the greater of the two bounds when inferring the subtype relation between the bodies.

$$\frac{\mathcal{E} \vdash T_0 <: S_0 \qquad \mathcal{E}, X <: S_0 \vdash S_1 <: T_1}{\mathcal{E} \vdash \forall(X <: S_0).S_1 <: \forall(X <: T_0).T_1}\ \forall - \mathsf{Loc}$$

This is expressive (subsuming both rules $\forall - \mathsf{Fun}$ and $\forall - \mathsf{Top}$) but lacks a sound and complete (let alone, decidable) subtyping algorithm.

In [16], it is shown that these three rules may coexist in a single system ($\mathsf{F}^{\mathsf{F}\top}_{<:}$) for subtyping bounded quantification by distinguishing the introduction and elimination forms of bounded quantification, and making the former a subtype of the latter. This avoids the pitfalls of $\forall - \mathsf{Top}$ and $\forall - \mathsf{Loc}$ but allows for a more expressive typing relation than Kernel $\mathsf{F}_{<:}$.

4.1 System $\mathsf{F}^{\mathsf{F}\top}_{<:}$

The type system (System $\mathsf{F}^{\mathsf{F}\top}_{<:}$) decorates bounded quantifiers with the superscripts $\{\mathsf{F}, \top\}$ (F for the introduction form, which obeys the rule of Kernel $\mathsf{F}_{<:}$ and $\top$ for the elimination form, which obeys that of System $\mathsf{F}^{\top}_{<:}$). Raw types are given by the grammar:

$$T ::= \top \mid X \mid T \rightarrow T \mid \forall^{\mathsf{F}}(X <: T).T \mid \forall^{\top}(X <: T).T$$

Table 1 gives rules defining subtyping judgments, $\mathcal{E} \vdash S <: T$, where $\mathcal{E}$ is a context of type variable bounds. These replace the single original typing rule for bounded quantification of System $\mathsf{F}_{<:}$ with the rules $\forall - \mathsf{Top}$, $\forall - \mathsf{Fun}$ and $\forall - \mathsf{Loc}$.

4.2 Semantics of Bounded Quantification

To interpret bounded quantification, we first observe that type-variable substitution can be extended to arenas. Node-types (which properly come with a context of free variables $\Theta \vdash m$) are closed under substitution.

▶ **Lemma 14.** *Given nodes $\Theta, X, \Theta'' \vdash m$ and $\Theta, \Theta' \vdash n$, the substitution of n for any free occurrence of X in m yields a node $\Theta, \Theta', \Theta'' \vdash m[n/X]$.*

Table 1 Subtyping Rules for System $\mathsf{F}^{\mathsf{F}\top}_{<:}$.

$$\frac{\Theta, X <: T, \Theta' \vdash \top}{\Theta, X <: T, \Theta' \vdash X <: T}\ \mathsf{Var} \qquad \frac{\Theta \vdash T}{\Theta \vdash T <: \top}\ \mathsf{Top} \qquad \frac{\Theta \vdash T}{\Theta \vdash T <: T}\ \mathsf{Refl}$$

$$\frac{\Theta \vdash T <: T' \qquad \Theta \vdash T' <: T''}{\Theta \vdash T <: T''}\ \mathsf{Trans}$$

$$\frac{\Theta \vdash S' <: S \quad \Theta \vdash T <: T'}{\Theta \vdash S \to T <: S' \to T'}\ \to \qquad \frac{\Theta, X <: S \vdash T <: T'}{\Theta \vdash \forall^{\mathsf{F}}(X <: S).T <: \forall^{\mathsf{F}}(X <: S).T'}\ \forall - \mathsf{Fun}$$

$$\frac{\Theta \vdash T_0 <: S_0 \qquad \Theta, X <: S_0 \vdash S_1 <: T_1}{\Theta \vdash \forall^{\mathsf{F}}(X <: S_0).S_1 <: \forall^{\top}(X <: T_0).T_1}\ \forall - \mathsf{Loc}$$

$$\frac{\Theta \vdash T_0 <: S_0 \qquad \Theta, X <: \top \vdash S_1 <: T_1}{\Theta \vdash \forall^{\top}(X <: S_0).S_1 <: \forall^{\top}(X <: T_0).T_1}\ \forall - \top$$

Noting that $m[n/X] \in O$ if $m, n \in O$ or $m, n \in P$, and $m[n/X] \in P$ otherwise, we define substitution into sub-arenas separately for negative and positive occurrences. Writing $\Theta \vdash A$ if $\Theta \vdash m$ for every $m \in A$:

▶ **Definition 15.** *The substitution of sub-arenas $\Theta, \Theta' \vdash B, C$ into $\Theta, X, \Theta'' \vdash A$ is defined:*

$$\Theta, \Theta', \Theta'' \vdash A(B, C)_X = \{m[n/X] \mid m \in A \cap P, n \in B\} \cup \{m[n/X] \mid m \in A \cap O, n \in C\}^{\circ}.$$

This substitution operation is antitone (with respect to the liveness ordering) in the first argument and monotone in the second:

▶ **Lemma 16.** *If $B' \preccurlyeq B$ and $C \preccurlyeq C'$ then $A(B, C)_X \preccurlyeq A(B', C')_X$.*

Proof. Suppose $m \in E(A(B, C)_X) \cap O(A(B', C')_X)$. Then $m = m'[n/X]$ for some $m' \in A$ and $n \in B' \cup C'$. There are three possibilities.

- $m' \in O(A)$ and $n \in O(C')$. Then $n \in E(C) \cap O(C')$ and so $n \in C$ and $m = m'[n/X] \in A(B, C)_X$ as required.
- $m' \in P(A)$ and $n \in P(B')$. Then $n \in E(B) \cap P(B')$ and so $n \in B$ and $m = m'[n/X] \in A(B, C)_X$ as required.
- X is not free in m' – then $m = m' \in A(B, C)_X$ as required.

Similarly, $E(A(B', C')_X) \cap P(E(A, B)) \subseteq A(B', C')_X$. ◀

A subtyping constraint of the form $X <: S$ corresponds to the ability to subsume any term of type X into the type S – i.e. there should be a canonical coercion of $[\![X]\!]$ into S. Thus a S-bounded type-variable may be represented as the arena $\Theta, X \vdash \{X\} \wedge [\![S]\!]$ (cf [18]) – i.e. $X <: S \vdash T$ denotes $[\![T]\!](\{X\} \wedge [\![S]\!], X \wedge [\![S]\!])$, so that $[\![X <: S \vdash X]\!] = \{X\} \wedge [\![S]\!] \preccurlyeq [\![X <: S \vdash S]\!]$.

However, these coercions are are only actually used for type-variables which occur negatively. This allows for two possible interpretations of bounded quantification – as $\forall X.[\![T]\!](X \wedge S, X \wedge S)_X$ or as $\forall X.[\![T]\!](X \wedge S, X)_X$. Noting that:

- $\forall X.[\![T]\!](X \wedge S, X \wedge S)_X \preccurlyeq \forall X.[\![T]\!](X \wedge S, X)_X$ by Lemma 16.
- Subtypings inferred using the bound hold for the first interpretation but not the second.
- The second interpretation is antitone in the variable bound but not the first.

we extend the interpretation of System $\mathrm{F}_\top$ types to System $\mathsf{F}^{\mathsf{F}\top}_{<:}$.

$$[\![\mathcal{E} \vdash \forall^{\mathsf{K}}(X <: S).T]\!] = \forall X.[\![\mathcal{E}, X <: S \vdash T]\!]$$
$$[\![\mathcal{E} \vdash \forall^{\top}(X <: S).T]\!] = \forall X.[\![\mathcal{E}, X <: \top \vdash T]\!](\{X\} \wedge [\![\mathcal{E} \vdash S]\!], \{X\})$$

Using Lemma 16, we show that this is sound with respect to the three subtyping rules for bounded quantification:

$\forall$ – Fun: If $A(\{X\} \wedge S, \{X\} \wedge S)_X \preccurlyeq B(\{X\} \wedge S, \{X\} \wedge S)_X$ then $\forall X.A(\{X\} \wedge S, \{X\} \wedge S)_X \preccurlyeq \forall X.B(\{X\} \wedge S, \{X\} \wedge S)_X$.

$\forall$ – Loc: If $D \preccurlyeq C$ and $A(\{X\} \wedge C, \{X\} \wedge C)_X \preccurlyeq B(\{X\} \wedge C, \{X\} \wedge C)_X$ then $\forall X.A(\{X\} \wedge C, \{X\} \wedge C)_X \preccurlyeq \forall X.B(\{X\} \wedge C, \{X\} \wedge C)_X \preccurlyeq \forall X.B(\{X\} \wedge C, \{X\})_X \preccurlyeq \forall X.B(\{X\} \wedge D, \{X\})_X$.

$\forall$ – Top: If $D \preccurlyeq C$ and $A(\{X\}, \{X\}) \preccurlyeq B(\{X\}, \{X\})$ then $\forall X.A(\{X\} \wedge C, \{X\}) \preccurlyeq \forall X.B(\{X\} \wedge C, \{X\})_X \preccurlyeq \forall X.B(\{X\} \wedge D, \{X\})_X$.

Thus (by induction on the length of derivation of $\Theta \vdash S <: T$):

▶ **Proposition 17** (Soundness). *If $\mathcal{E} \vdash S <: T$ then $[\![\mathcal{E} \vdash S]\!] \preccurlyeq [\![\mathcal{E} \vdash T]\!]$.*

5 Copycat Strategies

Having described an interpretation of bounded polymorphism at the level of subtyping, it is now necessary to show that this carries through to the term level. Interpreting terms as *strategies* on our type-arenas yields a semantics for subtype and parametric polymorphism which is based on *copycat strategies*. As canonical coercions, these give an alternative characterization of the liveness ordering – a "hereditarily total" copycat strategy exists between two arenas if and only if they are in the ordering. We now identify nodes explicitly with moves.

▶ **Definition 18.** *A* legal sequence *on an arena graph is a finite sequence of its moves which starts with an Opponent move, alternates between Opponent and Proponent moves and is equipped with a* justification pointer *[11] from each non-initial move to some preceding, enabling move.*

- Given arenas A, B, let $L(A, B)$ denote the set of legal sequences from A to B – that is, legal sequences on the arena $\overline{A} \oplus B$, where $\overline{(O, P, \rhd)} \triangleq (P, O, \rhd)$ swaps Proponent and Opponent moves, and $(O, P, \rhd) \oplus (O', P', \rhd') \triangleq (O \uplus O', P \uplus P', \rhd \oplus \rhd')$ is the smash sum of rooted graphs.
- Let $C(A, B)$ denote the set of *copycat sequences* from A to B – that is, legal sequences $t \in L(A, B)$ such that for every even-prefix $s \sqsubseteq^E t$, $s \upharpoonright A = s \upharpoonright B$.

It is easy to see that $C(A, B)$ is a deterministic strategy from A to B – that is, a non-empty set of even-length sequences in $L(A, B)$ which is even-prefix-closed and even-branching (if $s, t \in C(A, B)$ then $s \sqcap t$ has even length). Moreover:

▶ **Lemma 19.** *If $A \preccurlyeq B$ then $C(A, B)$ is a* hereditarily total *strategy: for any sequence $sm \in L(A, B)$ with $s \in C(A, B)$ there exists n such that $smn \in \sigma$.*

Proof. Suppose $s \in C(A, B)$ and $s(b.r) \in L(A, B)$, where b is an Opponent move in B. Either b is initial, or else it has a justifier in $s \upharpoonright B$, which therefore also occurs in $s \upharpoonright A$. So $b \in E(A) \cap O(B) \subseteq A$ (since $A \preccurlyeq B$) and $s(b.r)(b.l) \in C(A, B)$ as required. The case where $s(a.l) \in L(A, B)$ for some Proponent move in A is similar. ◀

The proof of the converse uses the following lemma.

▶ **Lemma 20.** *Suppose $C(A,B)$ is heredarily total.*
(i) *If $m \in E(A) \cap O(B)$ then there is a justified sequence of the form $s(m.r)(m.l)$ in $C(A,B)$.*
(ii) *If $m \in E(B) \cap P(A)$ then there is a justified sequence of the form $s(m.l)(m.r)$ in $C(A,B)$.*

Proof. By Noetherian induction. If m is initial then by totality, if $m \in O(B)$ then $(m.r)(m.l) \in C(A,B)$ and if $m \in P(B)$ then $(m.l)(m.r) \in C(A,B)$. For the induction case suppose, for example, that $m \in E(A) \cap O(B)$ is enabled by $a \in A$ and $b \in B$, which must both be Proponent or both Opponent moves by quasi-bipartiteness. In the former case, there exists $b' \in B \cup \{\top\}$ such that $b' \rhd B$, and by quasi-arborescence $b' \rhd a$ – i.e. $a \in E(B) \cap P(A)$, and so by inductive hypothesis there exists a justified sequence of the form $s(a.l)(a.r) \in C(A,B)$. Then $s(a.l)(a.r).(m.r)$ (with a pointer from $m.r$ to $a.r$) is a well-formed justfied sequence, and so by totality $s(a.l)(a.r)(m.r)(m.l) \in C(A,B)$ as required. The other cases are similar. ◀

Evidently, (i) and (ii) imply that $A \preccurlyeq B$ and so:

▶ **Proposition 21.** *$C(A,B)$ is hereditarily total if and only if $A \preccurlyeq B$.*

6 QA-arenas

To describe the interpretation of parametric polymorphism requires further structure on arenas – a *question-answer labelling* and *question-answer relation* supporting the interpretation of type abstraction and instantiation – introduced in [13, 14], (to which we refer for further details and proofs).

▶ **Definition 22.** *A QA-arena over a set $\mathcal{L}$ of labels (not containing "Q" and "A") is an arena $(O, P, \rhd)$ with the following additional structure:*

- *a labelling function $\lambda : O \cup P \to \mathcal{L} \cup \{Q, A\}$ partitioning moves into sets of* questions,answers *and $\mathcal{L}$-labelled* holes.
- *a* scoped question answer/relation *– a ternary relation on moves – $\lhd \subseteq (O \cup P) \times \lambda)^{-1}(Q) \times (\lambda)^{-1}(A)$. The relation $m \lhd (q,a)$ holds if a can answer q within the scope of m: Proponent questions are scoped by Proponent moves and answered by Opponent moves, and Opponent moves are scoped by Opponent moves and answered by Proponent moves.*

The type-arena defined in Section 3 is refined to a QA-arena for each type-variable context Θ by defining a QA-labelling and QA-relation over Θ for moves $\Theta \vdash m$. This is defined inductively via judgments of the form $\Theta \vdash^{QA} m : L$ (where $\Theta \vdash m$ and $L \in \{Q, A\} \cup \Theta$) and $\Theta \vdash^{QA} m \lhd (q,a)$ (where $\Theta \vdash m, q, a$), for which derivation rules are given in Table 2.

Informally, a move $m[X]$ is labelled with X if X occurs free in M. Otherwise (if X is bound):

- m is a question if X occurs in m with the same polarity as its binder.
- Its answers are moves $m'[X]$ in which X occurs with the opposite polarity to its binder.
- They are scoped by moves of the form $C[\forall X.m'']$ where m' is an initial move.

For example, in the sub-arena $\forall X.X \to X \to X$, the initial (Opponent) move $\forall X.\top \to \top \to X$ is a question, with answers $\forall X.\top \to X \to \top$ and $\forall X.X \to \top$. The scoping move for both question-answer pairs is the initial move $\forall X.\top \to \top \to X$. In other words, this corresponds to the usual QA-labelling for the arena of Booleans.

Table 2 Typing Judgments for QA-Labelling and QA-relation.

$$\frac{}{\Theta \vdash^{QA} X{:}X} \qquad \frac{\Theta \vdash^{QA} m{:}X}{\Theta \vdash^{QA} \forall X.m{:}Q} m \in= O \qquad \frac{\Theta \vdash^{QA} m{:}X}{\Theta \vdash^{QA} \forall X.M{:}A} m \in P \qquad \frac{\Theta \vdash^{QA} m{:}L}{\Theta \vdash^{QA} \forall X.m{:}L} L \neq X$$

$$\frac{\Theta \vdash^{QA} m{:}L}{\Theta \vdash^{QA} M \to \top{:}L} \qquad \frac{\Theta \vdash^{QA} m{:}L}{\Theta \vdash^{QA} \top \to m{:}L} \qquad \frac{\Theta \vdash^{QA} m{:}L}{\Theta \vdash^{QA} m \times \top{:}L} \qquad \frac{\Theta \vdash^{QA} m{:}L}{\Theta \vdash^{QA} \top \times m{:}L}$$

$$\frac{\Theta \vdash \top \rhd m \qquad \Theta \vdash^{QA} m'{:}X \qquad \Theta \vdash^{QA} m''{:}X}{\Theta \vdash^{QA} \forall X.m \lhd (\forall X.m', \forall X.m'')} (m', m'') \in (O, P)$$

$$\frac{\Theta \vdash^{QA} m \lhd (m', m'')}{\Theta \vdash^{QA} m \to \top \lhd (m' \to \top, m'' \to \top)} \qquad \frac{\Theta \vdash^{QA} m \lhd (m', m'')}{\Theta \vdash^{QA} \top \to m \lhd (\top \to m', \top \to m'')}$$

$$\frac{\Theta \vdash^{QA} m \lhd (m', m'')}{\Theta \vdash^{QA} m \times \top \lhd (m' \times \top, m'' \times \top)} \qquad \frac{\Theta \vdash^{QA} m \lhd (m', m'')}{\Theta \vdash^{QA} \top \times m \lhd (\top \times m', \top \times m'')}$$

6.1 Well-bracketed Strategies

A QA-arena over the empty set of labels is *closed*: matching questions and answers as opening and closing parentheses induces a relation between the moves of any legal sequence over a closed QA-arena : it is *well-bracketed* if this relation is contained within the QA-relation. More precisely, let the *pending question* of a sequence of moves t over such a closed arena be the prefix of t given (where defined) by:

$$\mathsf{pending}(sm) = \begin{cases} sm & \text{if } m \text{ is a question} \\ \mathsf{pending}(s') & \text{if } m \text{ is an answer and } \mathsf{pending}(s) = s'm' \end{cases}.$$

▶ **Definition 23.** *A well-bracketed sequence on a closed arena A is a legal sequence t on A which satisfies the* bracketing condition*:* Whenever $sa \sqsubseteq t$, where a is an answer, then $\mathsf{pending}(s) = s'q$, where $m \lhd (q, a)$ for some move m in s' which hereditarily justifies both q and a.

The *closure* of a QA-arena A over $\mathcal{L}$ is obtained by converting $\mathcal{L}$-labelled Opponent and Proponent moves to questions and answers, respectively, and extending the QA-relation by making initial moves into scoping moves for pairs of (Opponent,Proponent) moves with the same label.

▶ **Definition 24.** *Given $\mathcal{L} \vdash A$, the closed arena $\forall A$ has the same underlying arena. The QA-labelling is:*

$$\lambda_{\forall A}(m) = \begin{cases} Q & \textit{if } \lambda_A(m) \in \mathcal{L} \textit{ and } m \in O \\ A & \textit{if } \lambda_A(m) \in \mathcal{L} \textit{ and } m \in P \\ \lambda_A(m) & \textit{otherwise} \end{cases}$$

and the QA-relation is:

$$m \lhd_{\forall A} (q, a) \textit{ if } m \lhd_A (q, a) \textit{ or } \lambda_A(q) = \lambda_A(a) = l \in \mathcal{L} \textit{ and } \top \rhd m$$

A strategy σ from A to B is *well-bracketed* if every $s \in \sigma$ is a well-bracketed sequence on $\forall(\overline{A} \uplus B)$. It is *thread-independent* if:

For any well-bracketed r, s, t, where r is an interleaving of s and t:
$r \in \sigma$ if and only if $s, t \in \sigma$.

Let $CW(A, B)$ be the set of well-bracketed copycat sequences from A to B: this is well-bracketed by definition, and since any legal interleaving of legal sequences s and t is a copycat sequence if and only if r and s are copycats, it is a thread-independent strategy.

▶ **Definition 25.** *For a QA-arena U, the category $\mathcal{G}(U)$ consists of (objects) the sub-arenas of U with (morphisms) thread-independent strategies between them. Strategies are composed by parallel composition plus hiding*

$$\{s \in w(A, C) \mid \exists t \in (A + B + C)^*.t{\upharpoonright}A, B \in \sigma \wedge t{\upharpoonright}B, C \in \tau\}$$

The identity on A is the well-bracketed copycat $CW(A, A)$.

Well-bracketed copycats have a more general identity property.

▶ **Proposition 26.** *If $A' \preccurlyeq A$ and $B \preccurlyeq B'$ then for any $\sigma : A \to B$, $CW(A, A'); \sigma; CW(B, B') = \sigma \cap L(A', B')$.*

In particular, if $A \preccurlyeq B$ and $B \preccurlyeq C$ then $CW(A, B); CW(B, C) = CW(A, C)$, justifying:

▶ **Definition 27.** *Let $CW : \mathcal{S}(U) \to \mathcal{G}(U)$ be the identity-on-objects functor from the lattice $\mathcal{S}(U)$ (considered as a thin category) into $\mathcal{G}(U)$, sending $A \preccurlyeq B$ to the copycat $CW(A, B)$.*

7 Copycat Dinaturality

Returning to our family of second-order type arenas, the instantiation of a sub-arena into a strategy is given (as in [14]) by playing copycat between the arenas plugged into the holes. To define this on type arenas we make use of a restriction operation – a partial inverse to subsitution on moves:

▶ **Definition 28.** *Given moves $\Theta, \Theta', \Theta'' \vdash m$ and $\Theta, X, \Theta'' \vdash n[X]$ the restriction of m to $n[X]$ is defined:*

$$m{\upharpoonright}n[X] = \begin{cases} l & \textit{if } m = n[l/X] \\ \textit{undefined} & \textit{otherwise} \end{cases}.$$

By definition, $m[n/X]{\upharpoonright}m = n$ and if $m{\upharpoonright}n[X]$ is defined, then $m[(m{\upharpoonright}n[X])/X] = m$.

This operation lifts to justified sequences by applying it pointwise to the moves on which it is defined, and omitting moves on which it is not defined.

▶ **Definition 29.** *Given a justified sequence t, and move n, let $t{\upharpoonright}n$ be the justified sequence such that $\varepsilon{\upharpoonright}n = \varepsilon$ and*

$$(tm){\upharpoonright}n = \begin{cases} (t{\upharpoonright}n)(m{\upharpoonright}n) & \textit{if } m{\upharpoonright}n\downarrow \\ t{\upharpoonright}n & \textit{otherwise} \end{cases}$$

where $m{\upharpoonright}n$ points to $m'{\upharpoonright}n'$ in $t{\upharpoonright}n$, if both are defined and m points to m' in t. (Otherwise $m{\upharpoonright}n$ is initial and requires no pointer.)

We may now describe the instantiation of an arena into a strategy on type-arenas. Suppose t is a well-bracketed sequence from $A(C, C)_X$ to $B(C, C)_X$, such that $t \upharpoonright \forall(\overline{A} \uplus B)$ is a well-bracketed sequence. For any even prefix $sp[X] \sqsubseteq t{\upharpoonright}\forall\overline{A} \uplus B$, $p[X]$ is a Proponent answer in $\forall\overline{A} \uplus B$, so by well-bracketing $\mathsf{pending}(s) = s'o[X]$, where $o[X]$ is an Opponent question in $\forall\overline{A} \uplus B$. Say that t is a copycat-instantiated sequence for X if it is a copycat between every such pair of moves – i.e. for any even-length sequence t' with $s \sqsubseteq t' \sqsubseteq t$, $t{\upharpoonright}p[X] = t{\upharpoonright}o[X]$.

▶ **Definition 30.** *Given a strategy $\Theta, X, \Theta' \vdash \sigma : A \to B$,let $\Theta, \Theta', \Theta'' \vdash \sigma[C]_X : A(C,C)_X \to B(C,C)_X$ be the set of sequences t on $\forall A(C,C)_X \to B(C,C)_X$ which are copycat-instantiated for X, such that there exists $s \in \sigma$ with $s = t{\upharpoonright}\forall(\overline{A} \uplus B)$.*

This yields a family of strategies $\sigma[\_]_X =$

$$\{\Theta, \Theta', \Theta'' \vdash \sigma[C]_X : A(C,C)_X \to B(C,C)_X \mid \Theta, \Theta' \vdash C\}.$$

By Proposition 16, $A(\_,\_)_X$ and $B(\_,\_)_X$ act as mixed variance functors from $\mathcal{S}(\Theta, \Theta')$ to $\mathcal{S}(\Theta, \Theta', \Theta'')$, which may be composed with the copycat functor $CW : \mathcal{S}(\Theta, \Theta', \Theta'') \to \mathcal{G}(\Theta, \Theta', \Theta'')$.

▶ **Proposition 31.** *$\sigma[\_]_X$ is a* dinatural transformation *from $CW \cdot A(\_,\_)_X$ to $CW \cdot B(\_,\_)_X$.*

Proof. In other words, for any arenas $\Theta, \Theta' \vdash C \preccurlyeq D$, the dinaturality hexagon:

$$\begin{tikzcd}
 & A(C,C)_X \arrow[r, "\sigma[C]_X"] & B(C,C)_X \arrow[dr, "{CW(B(C,C),B(C,D))}"] & \\
A(D,C)_X \arrow[ur, "{CW(A(D,C),A(C,C))}"] \arrow[dr, "{CW(A(D,C),A(D,D))}"'] & & & B(C,D)_X \\
 & A(D,D)_X \arrow[r, "\sigma[D]_X"] & B(D,D)_X \arrow[ur, "{CW(B(D,D),B(C,D)}"'] &
\end{tikzcd}$$

commutes. By Proposition 26, this is equivalent to requiring that $\sigma[C]_X \cap W(C,D) = \sigma[D]_X \cap W(C,D)$, which follows from the definition of copycat instantiation. ◀

Note the importance of the restriction to copycat (or, at least, strict) strategies – the dinaturality hexagon above need not commute for all morphisms from C to D (even if σ is the denotation of a term of System F : see [8] for examples).

Instantiation extends to tuples of arenas – $B(A_1, A_1')_{X_1} \ldots (A_n, A_n')_{X_n}$ substitutes the moves of A_1 for negative occurrences of X_1 in B, the moves of A_1' for positive occurrences of X_1 and so on, giving a family of n-ary mixed variance functors on $\mathcal{S}(\Theta)$ to $\mathcal{S}(\Theta)$ which is closed under composition. Using these we may define a hyperdoctrine model of System F [20].

- Let $\mathcal{I}$ be the category in which objects are type-variable contexts and morphisms from Θ to $\Theta' = X_1, \ldots, X_n$ are substitutions – n-tuples of arenas $\langle \Theta \vdash A_1, \ldots, \Theta \vdash A_n \rangle$ composed by instantiation: $\langle B_1, \ldots, B_m \rangle \cdot \langle A_1, \ldots, A_n \rangle =$ $\langle B_1(A_1, A_1)_{X_1} \ldots (A_n, A_n)_{X_n}, \ldots, B_m(A_1, A_1)_{X_1} \ldots (A_n, A_n)_{X_n} \rangle$.
- For $\langle A_1, \ldots, A_n \rangle : \Theta \to \Theta'$, the *instantiation functor* $\mathcal{G}\langle A_1, \ldots, A_n \rangle : \mathcal{G}(\Theta') \to \mathcal{G}(\Theta)$ sends $\Theta' \vdash B$ to $\Theta \vdash B(A_1, A_1)_{X_1} \ldots (A_n, A_n)_{X_n}$, and $\Theta' \vdash \sigma : B \to C$ to $\Theta' \vdash \sigma[A_1]_{X_1} \ldots [A_n]_{X_n}$.
- Each category $\mathcal{G}(\Theta)$ is cartesian closed ($A \times B$ is the cartesian product of A and B, and $A \to B$ is their internal hom) and instantiation preserves this structure. Thus we have a functor $\mathcal{G}$ from $\mathcal{I}^{op}$ to the category of cartesian closed categories sending Θ to $\mathcal{G}(\Theta)$ and $\langle A_1, \ldots, A_n \rangle$ to $\mathcal{G}\langle A_1, \ldots, A_n \rangle$.
- For context Θ, Θ, X is the product of Θ with X in $\mathcal{I}$, and the image of the corresponding projections, $\mathcal{G}(\pi) : \mathcal{G}(\Theta) \to \mathcal{G}(\Theta, X)$ has an indexed left adjoint $\forall X$. Since instantiation commutes with type-variable abstraction, this satisfies the Beck-Chevalley condition.

Table 3 Typing Judgments for System $\mathsf{F}^{\mathsf{F}\top}_{<:}$.

$$\frac{\mathcal{E} \vdash \Gamma \qquad \mathcal{E} \vdash \top}{\Theta \vdash \mathsf{top} : \top}\ \top \qquad \frac{\mathcal{E} \vdash \Gamma, x : T, \Gamma'}{\mathcal{E}; \Gamma, x : T, \Gamma' \vdash x : T}\ \mathsf{var}$$

$$\frac{\mathcal{E}; \Gamma \vdash t : T \qquad \mathcal{E} \vdash T <: T'}{\mathcal{E}; \Gamma \vdash t : T'}\ \mathsf{sub}$$

$$\frac{\mathcal{E}; \Gamma, x : S \vdash t : T}{\mathcal{E}; \Gamma \vdash \lambda x : S.t : S \to T} \to -\mathsf{i} \qquad \frac{\mathcal{E}; \Gamma \vdash t : S \to T \qquad \mathcal{E}; \Gamma \vdash s : S}{\mathcal{E}; \Gamma \vdash t\, s : T} \to -\mathsf{e}$$

$$\frac{\mathcal{E}, X <: S; \Gamma \vdash t : T \qquad \mathcal{E} \vdash \Gamma}{\mathcal{E}; \Gamma \vdash \Lambda(X <: S).t : \forall^{\mathsf{F}}(X <: S).T}\ \forall - \mathsf{i}$$

$$\frac{\mathcal{E}; \Gamma \vdash t : \forall^{\top}(X <: S).T \qquad \mathcal{E} \vdash S' <: S}{\mathcal{E}; \Gamma \vdash t\{S'\} : T[S'/X]}\ \forall - \mathsf{e}$$

8 System $\mathsf{F}^{\mathsf{F}\top}_{<:}$ and its Semantics

The raw terms of System $\mathsf{F}^{\mathsf{F}\top}_{<:}$ are those of System $\mathsf{F}_{<:}$ – given by the grammar:

$$t ::= \mathsf{top} \mid x \mid \lambda(x : T).t \mid \Lambda(X <: T).t \mid t\, t \mid t\{T\}$$

Typing judgments (derived according to the rules in Table 3) take the form $\mathcal{E}; \Gamma \vdash t : T$, where Γ is a context of term-variables $x_1 : T_1, \ldots x_n : T_n$ such that $\mathcal{E} \vdash \Gamma$ – i.e. $\mathcal{E} \vdash T_1, \ldots, \mathcal{E} \vdash T_n$. Type-variable abstraction ($\forall$-introduction) is typed using $\forall^{\mathsf{F}}$ and instantiation ($\forall$-elimination) is typed using $\forall^{\top}$ – subsumption allows conversion from the former to the latter.

Each term-in-context $\mathcal{E}; x_1 : S_1, \ldots, x_n : S_n \vdash t : T$ denotes a morphism from $\llbracket \mathcal{E} \vdash S_1 \rrbracket \times \ldots \times \llbracket \mathcal{E} \vdash S_n \rrbracket$ to $\llbracket \mathcal{E} \vdash T \rrbracket$ in the category $\mathcal{G}(\llbracket |\mathcal{E}| \rrbracket)$, where $|X_1 <: R_1, \ldots, X_m <: R_m| = X_1, \ldots, X_m$. To interpret instantiation for bounded variables we require an operation taking a strategy $\sigma : A \to \forall X.B(\{X\} \wedge C, \{X\})_X$ and a bounded argument $D \preccurlyeq C$ to a strategy on $A \to B(D, D)_X$. The obvious way to do this is by instantiation of D for X in σ, followed by subsumption. However, the substitution operation on arenas does not respect the lattice structure of the liveness ordering.

▶ **Lemma 32.** *$(C \wedge A)(B, B)_X \neq C(B, B)_X \wedge C(B, B)_X$ in general.*

Proof. The meet $\{X\} \wedge A$ is the union $\{X\} \cup A$ (since X contains only a single initial move): the instantiation $(\{X\} \wedge A)(B, B)_X$ is thus equal to the union $A \cup B$, which is not the same as $A \wedge B$ in general. ◀

However, when substituting in bounded types we can use the following lemma

▶ **Lemma 33.** *If $C \preccurlyeq B$ then $C \preccurlyeq (B \cup C)$.*

Proof. $O(C) \cap (C \cup B) = (O(C) \cap C) \cup O(C) \cap B \subseteq C \cup C = C$. $P(B \cup C) \cap C \subseteq C \subseteq B \cup C$. ◀

So if $C \preccurlyeq D$, then $A(X \wedge B, X)_X(C, C)_X = A(C \cup B, C)_X \preccurlyeq A(C, C)_X$, and we may interpret bounded instantiation by substitution followed by subsumption – if $\mathcal{E} \vdash S <: S'$ then $\llbracket \mathcal{E}; \Gamma \vdash t\{S'\} : T[S'/X] \rrbracket =$

$$\llbracket \mathcal{E}; \Gamma \vdash t : \forall^{\top}(X <: S) \rrbracket; CW(\llbracket E, X <: S \vdash T \rrbracket(\llbracket E \vdash S' \rrbracket, \llbracket E \vdash S' \rrbracket)_X, \llbracket E \vdash T[S'/X] \rrbracket).$$

Since use of the subsumption rule may yield multiple derivations of the same typing judgment it is not immediately obvious that this defines a unique denotation for each term-in-context – we need to show that any derivation for a given term in context yields the same denotation (coherence).

▶ **Proposition 34.** *Every derivable typing judgment* $\Theta;\Gamma \vdash t : T$ *denotes a unique morphism* $\llbracket\Theta;\Gamma \vdash M : T\rrbracket$.

Proof. In [16] we define a derivation system for *minimal types*, such that if t is typable in context $\mathcal{E};\Gamma$ there is a unique derivation of a minimal type $\mathcal{E};\Gamma \vdash t : T$ such that if $\mathcal{E};\Gamma \vdash t : T'$ then $\mathcal{E} \vdash T <: T'$. To establish coherence, we show that any denotation for $\mathcal{E};\Gamma \vdash t : T'$ satisfies $\llbracket\mathcal{E};\Gamma \vdash t : T'\rrbracket = \llbracket\mathcal{E};\Gamma \vdash t : T\rrbracket; cw(\llbracket\mathcal{E} \vdash T\rrbracket, \llbracket\mathcal{E} \vdash T\rrbracket)$. ◀

Copycat dinaturality is the key to showing that the semantics is sound with respect to second-order β and η-equivalence:

▶ **Lemma 35.** *The model soundly interprets the rules:*

$$\frac{\mathcal{E}, X <: S;\Gamma \vdash t : T \qquad \mathcal{E};\Gamma \vdash R <: S}{\mathcal{E};\Gamma \vdash (\Lambda(X <: S).t)\{R\} = t[R/X] : T[R/X]}\ \beta_2$$

$$Y \notin \mathsf{dom}(\mathcal{E})\ \frac{\mathcal{E};\Gamma \vdash t : \forall^\top X <: S.T}{\mathcal{E};\Gamma \vdash \Lambda(Y <: S).(t\{Y\}) = t : \forall^\top(X <: S).T}\ \eta_2$$

Proof. We give the case of second-order η-equivalence as an example. Suppose σ is the uncurrying of $\llbracket\mathcal{E};\Gamma \vdash t : \forall^\top(X <: S).T\rrbracket$. The diagram

$$\begin{tikzcd}
& \llbracket E, X <: S \vdash T\rrbracket(\{X\} \wedge \llbracket\mathcal{E} \vdash S\rrbracket, \{X\} \wedge \llbracket E \vdash S\rrbracket) \arrow[dr, "\preccurlyeq" description] & \\
\llbracket\mathcal{E} \vdash \Gamma\rrbracket \arrow[ur, "\sigma[\{X\}\wedge\llbracket\mathcal{E}\vdash S\rrbracket_X]"] \arrow[dr, "\sigma[\{X\}]_X"'] & & \llbracket E, X <: \top \vdash T\rrbracket(\{X\} \wedge S, X)_X \\
& \llbracket E, X <: S \vdash T\rrbracket(\{X\}, \{X\})_X \arrow[ur, "\preccurlyeq" description] &
\end{tikzcd}$$

commutes by copycat dinaturality, and so $\llbracket\mathcal{E};\Gamma \vdash t : \forall^\top X <: S.T\rrbracket = \llbracket\mathcal{E};\Gamma \vdash \Lambda X.t\{X\} : \forall^\top(X <: S).T\rrbracket$ as required. ◀

▶ **Proposition 36.** *If* $\mathcal{E};\Gamma \vdash t =_{\beta\eta} t' : T$ *then* $\llbracket\mathcal{E};\Gamma \vdash t\rrbracket = \llbracket\mathcal{E};\Gamma \vdash t' : T\rrbracket$.

9 Conclusions and Further Directions

Our interpretation of the subtyping relation on Hyland-Ong arenas may be applied to a wide range of games models which employ this basic structure. We have also shown that it can be integrated, via dinaturality, with an interpretation of generic polymorphism based on bracketing structure, providing a way to use this principle which might be further explored in reasoning about program equivalence. This gives us the ingredients to develop existing games semantics for stateful objects with more expressive type theories such as Dependent Object Types [17, 22] (with an appropriate treatment for subtyping bounded quantification – see also [10]) as well as bounded abstract data types.

References

1 E. S. Bainbridge, P. J. Freyd, A. Scedrov, and P.J. Scott. Functorial polymorphism. *Theoretical Computer Science*, 70(1):35–64, 1990. `doi:10.1016/0304-3975(90)90055-m`.

2 K. Bruce and G. Longo. A modest model of records, inheritance and bounded quantification. *Information and Computation*, 87(1/2):196–240, 1990. `doi:10.1109/lics.1988.5099`.

3 L. Cardelli and P. Wegner. On understanding types, data abstraction and polymorphism. *Computing Surveys*, 17(4):471–522, 1985.

4 Luca Cardelli, John C. Mitchell, Simone Martini, and Andre Scedrov. An extension of System F with subtyping. *Information and Computation*, 109(1–2):4–56, 1994.

5 G. Castagna and B. C. Pierce. Decidable bounded quantification. In *Proceedings of POPL '94*, pages 1–29, 1994. `doi:10.1145/174675.177844`.

6 J. Chroboczek. Game semantics and subtyping. In *Proceedings of the fifteenth annual symposium on Logic in Computer Science*, pages 192–203. IEEE press, 2000. `doi:10.1109/lics.2000.855769`.

7 P.-L. Curien and G. Ghelli. Coherence of subsumption, minimum typing and type-checking in $\mathsf{F}_{<:}$. *Mathematical Structures in Computer Science*, 2(1):55–91, 1992. `doi:10.1007/3-540-52590-4_45`.

8 J. de Lataillade. Dinatural terms in System F. In *Proceedings of the 24th annual symposium on Logic in Computer Science, LICS '09*. IEEE Press, 2009. `doi:10.1109/lics.2009.30`.

9 J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50, 1987.

10 J. Z. S. Hu and O. Lhoták. Undecidability of $\mathsf{D}_{<:}$ and its decidable fragments. *Proceedings of the ACM on Programming Languages (POPL)*, 4(9):1–30, 2020. `doi:10.1145/3371077`.

11 J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. *Information and Computation*, 163:285–408, 2000.

12 D. Katiyar and S. Sankar. Completely bounded quantification is decidable. In *Proceedings of the ACM SIGPLAN Workshop on ML and its Applications*, pages 68–77, 1992.

13 J. Laird. Game semantics for a polymorphic programming language. In *Proceedings of LICS '10*. IEEE Press, 2010.

14 J. Laird. Game semantics for a polymorphic programming language. *Journal of the ACM*, 60(4), 2013.

15 J. Laird. Game semantics for bounded polymorphism. In *Proceedings of FoSSaCS '16*, number 9634 in LNCS. Springer, 2016. `doi:10.1007/978-3-662-49630-5_4`.

16 J. Laird. Revisiting decidable bounded quantification, via dinaturality. In *Proceedings of Mathematical Foundations of Program Semantics '22*, Electronic Notes in Theoretical Computer Science, 2022. To appear.

17 N. Amin, S. Grütter, M. Odersky, T. Rompf, and S. Stucki. The essence of dependent object types. In *A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, number 9600 in LNCS, pages 249–272. Springer, 2016. `doi:10.1007/978-3-319-30936-1_14`.

18 B. C. Pierce. *Programming with Intersection Types and Bounded Polymorphism.* PhD thesis, Carnegie Mellon University, 1991. URL: `https://dl.acm.org/doi/10.5555/145640`.

19 Benjamin C. Pierce. Bounded quantification is undecidable. In *POPL*, pages 305–315, 1992. `doi:10.1006/inco.1994.1055`.

20 A. M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.

21 J. C. Reynolds. Syntactic Control of Interference. In *Conf. Record 5^{th} ACM Symposium on Principles of Programming Languages*, pages 39–46, 1978.

22 Tiark Rompf and Nada Amin. From F to DOT: type soundness proofs with definitional interpreters. *CoRR*, abs/1510.05216, 2015. `arXiv:1510.05216`.

23 Sergei Vorobyov. Structural decidable extensions of bounded quantification. In *Proceedings of POPL '95*, pages 164–175, 1995. `doi:10.1145/199448.199479`.

Representing Guardedness in Call-By-Value

Sergey Goncharov
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Abstract

Like the notion of computation via (strong) monads serves to classify various flavours of impurity, including exceptions, non-determinism, probability, local and global store, the notion of guardedness classifies well-behavedness of cycles in various settings. In its most general form, the guardedness discipline applies to general symmetric monoidal categories and further specializes to Cartesian and co-Cartesian categories, where it governs guarded recursion and guarded iteration respectively. Here, even more specifically, we deal with the semantics of call-by-value guarded iteration. It was shown by Levy, Power and Thielecke that call-by-value languages can be generally interpreted in Freyd categories, but in order to represent effectful function spaces, such a category must canonically arise from a strong monad. We generalize this fact by showing that representing *guarded* effectful function spaces calls for certain parametrized monads (in the sense of Uustalu). This provides a description of guardedness as an intrinsic categorical property of programs, complementing the existing description of guardedness as a predicate on a category.

2012 ACM Subject Classification Theory of computation → Categorical semantics; Theory of computation → Axiomatic semantics

Keywords and phrases Fine-grain call-by-value, Abstract guardedness, Freyd category, Kleisli category, Elgot iteration

Digital Object Identifier 10.4230/LIPIcs.FSCD.2023.34

Acknowledgements The author would like to thank anonymous reviewers of the present and previous editions of the paper for their diligence in their effort to improve it.

1 Introduction

A traditional way to model call-by-value languages is based on a clear cut separation between computations and values. A computation can be *suspended* and thus turned into a value, and a value can be *executed*, and thus again be turned into a computation. The paradigmatic example of these conversions are the application and abstraction mechanisms of λ-calculus. From the categorical modelling perspective, this view naturally requires two categories, suitably connected with each other. As essentially suggested by Moggi [28], a minimal modelling framework requires a Cartesian category (i.e. a category with finite products) as a category of values and a Kleisli category of a strong monad over it, as a category of (side-effecting) computations (also called *producers* [23]). A generic *computational metalanguage* thus arises as an internal language of strong monads. Levy, Power and Thielecke [25], designed a refinement of Moggi's computational metalanguage, called *fine-grain call-by-value (FGCBV)*, whose models are not necessarily strong monads, but are more general *Freyd categories*. They have shown that a strong monad in fact always emerges from a Freyd category if certain function spaces (needed to interpret higher-order functions), are *representable* as objects of the value category – thus strong monads arise from first principles.

Here we analyse an extension of the FGCBV paradigm with a notion of guardedness, which is a certain predicate on computations, certifying their well-behavedness, in particular that they can be iterated [19, 24]. A typical example is guardedness in process algebra, where guardedness is often used to ensure that recursive systems of process definitions have unique solutions [27]. FGCBV does not directly deal with fixpoints, since these are usually

8th International Conference on Formal Structures for Computation and Deduction (FSCD 2023).
Editors: Marco Gaboardi and Femke van Raamsdonk; Article No. 34; pp. 34:1–34:21
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

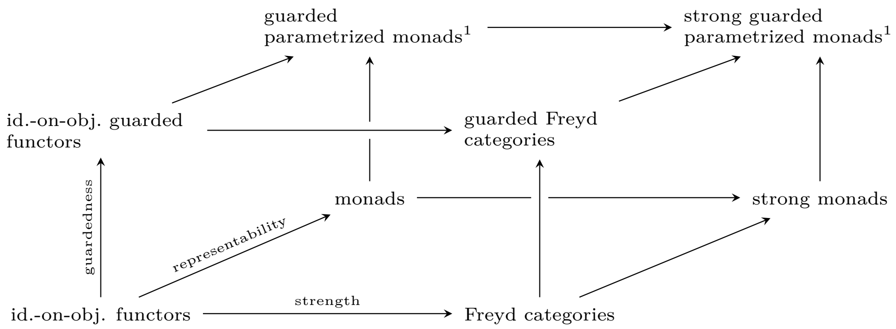


Figure 1 Three dimensions within call-by-value.

considered to be features orthogonal to computational effects and evaluation strategies. Analogously, even though the notion of guardedness is motivated by fixpoints, here we do not consider (guarded) fixpoints as a core language feature. In fact, in practically relevant cases guardedness is meaningful on its own as a suitable notion of productivity of computation, and need not be justified via fixpoints, which may or may not exist. In FGCBV one typically regards general recursion to be supported by the category of values, and once the latter indeed supports it, it is obvious to add a corresponding fixpoint construct to the language. Nevertheless, general recursion entails partiality for programs, which means that even if we abstract away from recursion, the corresponding effect of *partiality* must be part of the computational effect abstraction (see e.g. [12]). Recursion and computational effects are thus intimately connected. This connection persists under the restriction from general recursion to iteration, which is subject to a much broader range of models, and triggers the partiality effect just as well. Arguably, the largest class of monads, supporting iteration are *Elgot monads* [3, 16]. These are monads T, equipped with *Elgot iteration*:

$$\frac{f\colon X \to T(Y+X)}{f^\dagger\colon X \to TY} \tag{1}$$

and subject to established equational laws of iteration [7, 35]. Intuitively, $f^\dagger$ is obtained from f by iterating away the right summand in the output type $Y + X$. For example, the maybe-monad $(\text{–}) + 1$ is an Elgot monad over the category of *classical* sets, which yields a model for a while-language with non-termination as the only computational effect. Now, *guarded Elgot monads* [24] refine Elgot monads in that, the operator (1) needs only be defined w.r.t. a custom class of *guarded morphisms*, governed by simple laws. Proper partiality of the guardedness predicate is relevant for various reasons, including the following:

- Guarded fixpoints often uniquely satisfy the corresponding fixpoint equation, which greatly facilitates reasoning, which is extensively used in process algebra.
- In type-theoretic and constructive setting guarded iteration can often be defined natively and more generally, e.g. the "simplest" guarded Elgot monad is Capretta's *delay monad* (initially called "partiality monad") [8], rendered by final coalgebras $D = \nu\gamma.\,(\text{–}+\gamma)$, which yield an intensional counterpart of the maybe-monad; guardedness then means

[1] More precisely, representability yields parametrized guarded monads, subject to an additional monicity condition. This is treated in detail in Section 7.

productivity, i.e. that the computation signals as it evolves. Contrastingly, the "simplest" Elgot monad is much harder to construct and arguably requires additional principles to be available in the underlying metatheory [9, 4, 11, 14].

- Guardedness is a compositional type discipline, and hence it potentially helps to encapsulate additional information about productivity of programs in types, like monads encapsulate the information about potential side-effects.

As indicated above, strong monads can be regarded as structures, in a canonical way arising from FGCBV by adding the requirement of representability of certain function spaces in the category of values. This is behind the mechanism of representing computational effects via monads in type systems (e.g. in Fω, by quantification over higher kinds) and hence in programming languages (e.g. in Haskell). Our goal is to provide an analogous mechanism for guardedness and for its combinations with computational effects and strength. That is, (strong) monads are an answer to the question: *what is the categorical/type-theoretic structure that faithfully represents computational effects within a higher-order universe?* Here, we answer the question: *what is the categorical/type-theoretic structure that faithfully represents guarded computational effects within a higher-order universe?* In other words, we seek a formulation of guardedness as an intrinsic structural property of morphisms, instead of additional data that (anonymously) identifies guarded morphisms among others. In doing so, we take inspiration from the view of monads as structures for representing effects, as summarized above. In fact, we show that strength, representability and guardedness can be naturally arranged within FGCBV as three orthogonal dimensions, as shown in Figure 1 (the arrows point from more general concepts to more specific ones). The bottom face of the cube features the above mentioned connection between Freyd categories and strong monads, and a corresponding connection between identity-on-object functors and (not necessarily strong) monads. We contribute with the top face, which combines guardedness with the other dimensions. The pivotal point is the combination of guardedness with representability, which produces a certain class of *parametrized monads* [37], which we dub *guarded parametrized monads.*

Related work. We benefit from the analysis of Power and Robinson [32] who introduced *premonoidal categories* as an abstraction of Kleisli categories. Freyd categories were subsequently defined by Power and Thielecke [33] as premonoidal categories with additional structure and also connected to strong monads. Levy [23] came up with an equivalent definition, which we use throughout. In the previous characterization [33, 25], strong monads were shown to arise jointly with Kleisli exponentials from *closed Freyd categories.* We refine this characterization (Corollary 9) by showing that strong monads in fact arise independently of exponentials (Proposition 8). *Distributive Freyd categories* were defined by Staton [36] – here we use them to extend the FGCBV language by coproducts and subsequently with guardedness predicates. Previous approaches to identifying structures for ensuring guardedness on monads involved *monad modules* [30, 2] – we make do with guarded parametrized monads instead, which combine monads with modules over them and arise universally.

Plan of the paper. After short technical preliminaries, we start off by introducing a restricted version of FGCBV in Section 3 and extensively discuss motivating examples, which (with a little effort) can already be encoded despite restrictions. We establish a very simple form of the representability scenario, producing monads, and meant to serve as a model for subsequent sections. In Section 4 we deal with full FGCBV, Freyd categories, modelling them and strong monads, representing Freyd categories. The guardedness dimension is added in Section 5 where we introduce *guarded Freyd categories* and in Section 6 we analyse the representability issue for them. Finally, in Section 7 we introduce an equational axiomatization of a categorical structure for representing guardedness, called *guarded parametrized monads*. As a crucial technical step, we establish a coherence property in the style of Mac Lane's coherence theorem for monoidal categories [26]. Conclusions are drawn in Section 8.

2 Preliminaries

We assume familiarity with the basics of category theory [26, 5]. For a category $\mathbf{V}$, $|\mathbf{V}|$ will denote the class of objects and $\mathbf{V}(X, Y)$ will denote morphisms from X to Y. We tend to omit indexes at natural transformations for readability. A category with finite (co-)products is called (co-)Cartesian. In a co-Cartesian category with selected coproducts, we write $!\colon 0 \to A$ for the initial morphism, and $\mathsf{inl}\colon A \to A + B$ and $\mathsf{inr}\colon B \to A + B$ for the left and right coproduct injections correspondingly. A *distributive category* [10] is a Cartesian and co-Cartesian category, in which the natural transformation

$$X \times Y + X \times Z \xrightarrow{[\mathsf{id}\times\mathsf{inl},\ \mathsf{id}\times\mathsf{inr}]} X \times (Y + Z)$$

is an isomorphism, whose inverse we denote $\mathsf{dist}_{X,Y,Z}$ (a co-Cartesian and Cartesian closed category is always distributive). Let $\Delta = \langle \mathsf{id}, \mathsf{id}\rangle\colon X \to X \times X$ and $\nabla = [\mathsf{id}, \mathsf{id}]\colon X + X \to X$.

A monad $\mathbb{T}$ on $\mathbf{V}$ is determined by a *Kleisli triple* $(T, \eta, (-)^\star)$, consisting of a map $T\colon |\mathbf{V}| \to |\mathbf{V}|$, a family of morphisms $(\eta_X\colon X \to TX)_{X\in|\mathbf{V}|}$ and *Kleisli lifting* sending each $f\colon X \to TY$ to $f^\star\colon TX \to TY$ and obeying *monad laws*:

$$\eta^\star = \mathsf{id}, \qquad f^\star \circ \eta = f, \qquad (f^\star \circ g)^\star = f^\star \circ g^\star.$$

It follows that T extends to a functor, η extends to a natural transformation – *unit*, $\mu = \mathsf{id}^\star\colon TTX \to TX$ extends to a natural transformation – *multiplication*, and that (T, η, μ) is a monad in the standard sense [26]. We will generally use blackboard capitals (such as $\mathbb{T}$) to refer to monads and the corresponding Roman letters (such as T) to refer to their functor parts. Morphisms of the form $f\colon X \to TY$ are called *Kleisli morphisms* and form the *Kleisli category* $\mathbf{V}_{\mathbb{T}}$ of $\mathbb{T}$ under *Kleisli composition* $f, g \mapsto f^\star \circ g$ with identity η.

A functor F is *strong* if it is equipped with a natural transformation *strength* $\tau\colon X \times FY \to F(X \times Y)$, such that the diagrams

$$\begin{tikzcd}
1 \times FX \arrow[r, "\mathsf{snd}"] \arrow[d, "\tau"'] & FX \\
F(1 \times X) \arrow[ru, "F\,\mathsf{snd}"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
(X \times Y) \times FZ \arrow[rr, "\tau"] \arrow[d, phantom, "\cong" rotate=-90] & & F((X \times Y) \times Z) \arrow[d, phantom, "\cong" rotate=-90] \\
X \times (Y \times FY) \arrow[r, "X \times \tau"] & X \times F(Y \times Z) \arrow[r, "\tau"] & F(X \times (Y \times Z))
\end{tikzcd}$$

commute. A natural transformation, between two strong functors is strong if it preserves strength in the obvious sense, and a monad $\mathbb{T}$ is strong if T is strong with some strength $\tau\colon X \times TY \to T(X \times Y)$ and η and μ are strong with id being the strength of Id and $T\tau \circ \tau\colon X \times TTY \to TT(X \times Y)$ being the strength of TT.

3 Simple FGCBV with Coproducts

We start off with a restricted – single-variable – fragment of FGCBV, but extended with coproduct types. Since we will not deal with operational semantics, we simplify the language slightly (e.g. we do not include let-expressions for values). We also stick to a Haskell-style

$$\frac{}{x\colon A \vdash_{\mathsf{v}} x\colon A} \qquad \frac{f\colon A \to B \in \Sigma_v \quad \Gamma \vdash_{\mathsf{v}} v\colon A}{\Gamma \vdash_{\mathsf{v}} f(v)\colon B} \qquad \frac{f\colon A \to B \in \Sigma_c \quad \Gamma \vdash_{\mathsf{v}} v\colon A}{\Gamma \vdash_{\mathsf{c}} f(v)\colon B}$$

$$\frac{\Gamma \vdash_{\mathsf{v}} v\colon A}{\Gamma \vdash_{\mathsf{c}} \mathsf{return}\, v\colon A} \qquad \frac{\Gamma \vdash_{\mathsf{c}} p\colon A \quad x\colon A \vdash_{\mathsf{c}} q\colon B}{\Gamma \vdash_{\mathsf{c}} \mathsf{do}\, x \leftarrow p;\, q\colon B} \qquad \frac{\Gamma \vdash_{\mathsf{v}} v\colon 0}{\Gamma \vdash_{\mathsf{v}} \mathsf{init}\, v\colon A}$$

$$\frac{\Gamma \vdash_{\mathsf{v}} v\colon A}{\Gamma \vdash_{\mathsf{v}} \mathsf{inl}\, v\colon A + B} \qquad \frac{\Gamma \vdash_{\mathsf{v}} v\colon B}{\Gamma \vdash_{\mathsf{v}} \mathsf{inr}\, v\colon A + B} \qquad \frac{\Gamma \vdash_{\mathsf{v}} v\colon A + B \quad x\colon A \vdash_{\mathsf{c}} p\colon C \quad y\colon B \vdash_{\mathsf{c}} q\colon C}{\Gamma \vdash_{\mathsf{c}} \mathsf{case}\, v\, \mathsf{of}\, \mathsf{inl}\, x \mapsto p;\, \mathsf{inr}\, y \mapsto q\colon C}$$

Figure 2 Simple FGCBV with coproducts.

syntax with do-notation and case-expressions. We fix a collection of sorts $S_1, S_2, \ldots$, a signature Σ_v of pure programs $f\colon A \to B$, and a signature Σ_c of effectful programs $f\colon A \to B$ (also called *generic effects* [31]) where A and B are types, generated with the grammar

$$A, B ::= S_1, S_2, \ldots \mid 0 \mid A + B. \tag{2}$$

We then define terms in context of the form $x\colon A \vdash_{\mathsf{v}} v\colon B$ and $x\colon A \vdash_{\mathsf{c}} p\colon B$ for value terms and computation terms inductively by the rules given in Figure 2. (where we chose to stick to the syntax of the familiar Haskell's do-notation): This language is essentially a refinement of Moggi's *simple (!) computational metalanguage*, which only has one-variable contexts, instead of the fully fledged multi-variable contexts. In terms of monads, the present language corresponds to not necessarily strong ones. Such monads are not very useful in traditional programming languages semantics, however we dwell on this case for several reasons. We aim to explore the interaction between guardedness and monads from a foundational perspective and stay as general as possible to cover the cases where strength does not exist or is not relevant. We also would like to identify the basic representation scenario, to be extended later in more sophisticated cases.

An obvious extension of the presented language would be the iteration operator:

$$\frac{\Gamma \vdash_{\mathsf{c}} p\colon A \qquad x\colon A \vdash_{\mathsf{c}} q\colon B + A}{\Gamma \vdash_{\mathsf{c}} \mathsf{iter}\, x \leftarrow p;\, q\colon B} \tag{3}$$

meant to satisfy the fixpoint equality $\mathsf{iter}\, x \leftarrow p;\, q = \mathsf{iter}\, x \leftarrow (\mathsf{do}\, x \leftarrow p;\, q);\, q$. This syntax serves its technical purpose of adding expressivity to the language, but can be criticized from a pragmatic perspective – an arguably more convenient, equivalent syntax of *"labelled iteration"* can be used instead [13], and carried over to guarded setting [17]. Presently, we focus on representing guardedness as such and do not include iteration in the language.

We present three examples, which can be interpreted w.r.t. the single-variable case, to demonstrate the unifying power of FGCBV and to illustrate various flavours of guardedness.

▶ **Example 1** (Basic Process Algebra [6]). *Basic process algebra (BPA)* over a set of actions $\mathcal{A}$ is defined by the grammar: $P, Q ::= (a \in \mathcal{A}) \mid P + Q \mid P \cdot Q$. One typically considers BPA-terms over free variables (seen as process names) to solve systems of recursive process equations w.r.t. these variables. E.g. we can specify a 2-bit FIFO buffer as a solution to

$$B_0 = \mathsf{in}_0 \cdot B_1^0 + \mathsf{in}_1 \cdot B_1^1, \qquad B_1^i = \mathsf{in}_0 \cdot B_2^{0,i} + \mathsf{in}_1 \cdot B_2^{1,i} + \mathsf{out}_i \cdot B_0, \qquad B_2^{i,j} = \mathsf{out}_j \cdot B_1^i,$$

where $i, j \in \{0, 1\}$. We view B_0 as an empty FIFO, B_1^i as a FIFO carrying only i and $B_2^{i,j}$ as a FIFO, carrying i and j. For example, the trace $B_0 \xrightarrow{\mathsf{in}_0} B_1^0 \xrightarrow{\mathsf{in}_1} B_2^{1,0} \xrightarrow{\mathsf{out}_0} B_1^1 \xrightarrow{\mathsf{out}_1} B_0$ is valid and represents pushing 0 and 1 to an empty FIFO and then popping them out in the same order. We can model such systems of equations in FGCBV as follows. Let us fix a single sort 1 and identify an n-fold sum $(\ldots(1+\ldots)\ldots)+1$ with the natural number n. The injections $\mathsf{inj}_i\colon 1 \to n$ are defined inductively in the obvious way. Let $\Sigma_v = \emptyset$ and $\Sigma_c = \{a\colon 1 \to 1 \mid a \in \mathcal{A}\} \cup \{\mathsf{toss}\colon 1 \to 2\}$. A BPA-term over process names $\{N_1, \ldots, N_n\}$ can be translated to FGCBV recursively, with the following rules where $\rightsquigarrow$ reads as "translates":

$$\frac{}{N_i \rightsquigarrow x\colon 1 \vdash_{\mathsf{c}} \mathsf{return}(\mathsf{inl}(\mathsf{inj}_i\, x))\colon n+1} \qquad \frac{}{a \rightsquigarrow x\colon 1 \vdash_{\mathsf{c}} \mathsf{do}\, x \leftarrow a(x);\, \mathsf{return}(\mathsf{inr}\, x)\colon n+1}$$

$$\frac{P \rightsquigarrow x\colon 1 \vdash_{\mathsf{c}} p\colon n+1 \qquad Q \rightsquigarrow x\colon 1 \vdash_{\mathsf{c}} q\colon n+1}{P+Q \rightsquigarrow x\colon 1 \vdash_{\mathsf{c}} \mathsf{do}\, x \leftarrow \mathsf{toss}(x);\, \mathsf{case}\, x\, \mathsf{of}\, \mathsf{inl}\, x \mapsto p;\, \mathsf{inr}\, x \mapsto q\colon n+1}$$

$$\frac{P \rightsquigarrow x\colon 1 \vdash_{\mathsf{c}} p\colon n+1 \qquad Q \rightsquigarrow x\colon 1 \vdash_{\mathsf{c}} q\colon n+1}{P \cdot Q \rightsquigarrow x\colon 1 \vdash_{\mathsf{c}} \mathsf{do}\, x \leftarrow p;\, \mathsf{case}\, x\, \mathsf{of}\, \mathsf{inl}\, x \mapsto \mathsf{return}(\mathsf{inl}\, x);\, \mathsf{inr}\, x \mapsto q(x)\colon n+1}$$

Intuitively, the terms $x\colon 1 \vdash_{\mathsf{c}} p\colon n+1$ represent processes with $n+1$ exit points: every process name N_i identifies an exit i, in addition to the global anonymous exit, as e.g. in an action, regarded as a process. The generic effect toss induces binary nondeterminism as a coin-tossing act. Every tuple $(x\colon 1 \vdash_{\mathsf{c}} p_i\colon m)_{i<n}$ can be represented by a single term $x\colon n \vdash_{\mathsf{c}} \hat{p}_n\colon m$, inductively defined as follows:

$$\hat{p}_0 = \mathsf{return}(\mathsf{init}\, x), \qquad \hat{p}_{n+1} = \mathsf{case}\, x\, \mathsf{of}\, \mathsf{inl}\, x \mapsto \hat{p}_n;\, \mathsf{inr}\, x \mapsto p_{n+1}.$$

Every system of n equations with $m+n$ variables is thus represented by a term $x\colon n \vdash_{\mathsf{c}} p\colon m+n$. The iteration $x\colon n \vdash_{\mathsf{c}} \mathsf{iter}\, x \leftarrow \mathsf{return}\, x;\, p\colon m$ computes a solution of this system, sending every i-th variable to a term over the remaining m free variables. Guarded systems are those, where recursive calls are preceded by actions. Such systems have a unique solution (under bisimilarity). The simplest unguarded example $P = P$ has arbitrary solutions, and translates to $x\colon 1 \vdash_{\mathsf{c}} \mathsf{iter}\, x \leftarrow \mathsf{return}\, x;\, \mathsf{return}(\mathsf{inr}\, x)\colon 0$.

▶ **Example 2** (Imperative Traces). We adapt the semantic framework of Nakata and Uustalu [29] for imperative coinductive traces to our setting. Let us fix a set P of *predicates*, a set T of *state transformers*, and let the corresponding pure and effectful signatures be $\Sigma_v = \{p\colon S \to S+S \mid p \in P\} \cup \{t\colon S \to S \mid t \in T\}$ and $\Sigma_c = \{\mathsf{put}\colon S \to 1, \mathsf{get}\colon 1 \to S\}$ over the set of sorts $\{S, 1\}$. The intended interpretation of this data is as follows:

- S is a set of memory states, e.g. the set of finitely supported partial functions $\mathbb{N} \hookrightarrow 2$;
- T are state transformers, e.g. functions, updating precisely one specified memory bit;
- $p \in P$ encode predicates: $p(s) = \mathsf{inl}\, s$ if the predicate is satisfied and $p(s) = \mathsf{inr}\, s$ otherwise, e.g. p can capture functions that give a Boolean answer to the questions "is the specified bit 0?" and "is the specified bit 1?".

For example, the following program negates the i-th memory bit (if it is present)

$$x\colon 1 \vdash_{\mathsf{c}} \mathsf{do}\, s \leftarrow \mathsf{get}(x);\, \mathsf{case}\, (s[i] = 0)\, \mathsf{of}\, \mathsf{inl}\, s \mapsto \mathsf{put}(s[i := 1]);\, \mathsf{inr}\, s \mapsto \mathsf{put}(s[i := 0])\colon 1$$

where $(–[i] = 0)$, $(–[i := 0])$ and $(–[i := 1])$ are the obvious predicate and state transformers. Nakata and Uustalu [29] argued in favour of (infinite) traces as a particularly suitable semantics for reasoning about imperative programs. This means that store updates must contribute to the semantics, which can be ensured by a judicious choice of syntax, e.g. by using $\mathsf{skip} = \mathsf{do}\, s \leftarrow \mathsf{get}(x);\, \mathsf{put}(s)$, but not return. In FGCBV, however, iterating $x\colon 1 \vdash_{\mathsf{c}} \mathsf{return}(\mathsf{inr}\, x)\colon 1$ would not yield any trace. By restricting to guarded iteration with guardedness meaning writing to the store, we can indeed prevent iterating such programs, by defining guardedness in such a way that recursive calls are preceded by put.

▶ **Example 3** (Hybrid Programs)**.** Hybrid programs combine discrete and continuous capabilities and thus can be used to describe behaviours of cyber-physical systems. For simplicity we consider time delays as the only hybrid facility – more sophisticated scenarios are treated elsewhere [15] (more sophisticates scenarios can be modelled in a similar way [15]). Let $\mathbb{R}_{\geqslant 0}$ be the sort of non-negative real numbers and let Σ_v contain all unary operations on non-negative reals and additionally $\mathsf{is}_0\colon \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0} + \mathbb{R}_{\geqslant 0}$, which sends $n = 0$ to $\mathsf{inl}\, n$ and $n > 0$ to $\mathsf{inr}\, n$. Let $\Sigma_c = \{\mathsf{wait}\colon \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}\}$. With $\mathsf{wait}(r)$ we can introduce a time delay of length r and return r. With iteration we can write programs like

$$\begin{aligned} x\colon \mathbb{R}_{\geqslant 0} \vdash_{\mathsf{c}} \mathsf{iter}\, x \leftarrow \mathsf{return}\, x;\ \mathsf{case}\ \mathsf{is}_0(x)\ \mathsf{of}\ & \\ & \mathsf{inl}\, x \mapsto \mathsf{return}(\mathsf{inl}\, x); \\ & \mathsf{inr}\, x \mapsto (\mathsf{do}\, x \leftarrow \mathsf{wait}(x);\ \mathsf{return}(\mathsf{inr}\, f(x)))\colon \mathbb{R}_{\geqslant 0}, \end{aligned}$$

which terminate successfully in finite time ($f(x) = x \dot{-} 1^2$), run infinitely ($f(x) = 1$), or exhibit *Zeno behaviour* ($f(x) = x/2$), i.e. consume finite time, but never terminate. In all these examples, every iteration consumes non-zero time. This is also often considered to be a well-behavedness condition, which can be interpreted as guardedness.

In order to interpret the language from Figure 2, let us fix two co-Cartesian categories $\mathbf{V}$ and $\mathbf{C}$, and an identity-on-objects functor $J\colon \mathbf{V} \to \mathbf{C}$ (hence $|\mathbf{V}| = |\mathbf{C}|$), strictly preserving coproducts. A semantics of (Σ_v, Σ_c) over J assigns

- an object $[\![A]\!] \in |\mathbf{V}|$ to each sort A;
- a morphism $[\![f]\!] \in \mathbf{V}([\![A]\!], [\![B]\!])$ to each $f\colon A \to B \in \Sigma_v$;
- a morphism $[\![f]\!] \in \mathbf{C}([\![A]\!], [\![B]\!])$ to each $f\colon A \to B \in \Sigma_c$,

which extends to types and terms as follows: $[\![0]\!] = 0$, $[\![A + B]\!] = [\![A]\!] + [\![B]\!]$,

- $[\![x\colon A \vdash_{\mathsf{v}} x\colon A]\!] = \mathsf{id}$;
- $[\![\Gamma \vdash_{\mathsf{v}} f(v)\colon B]\!] = [\![f]\!] \circ [\![\Gamma \vdash_{\mathsf{v}} v\colon A]\!]$;
- $[\![\Gamma \vdash_{\mathsf{c}} f(v)\colon B]\!] = [\![f]\!] \circ J[\![\Gamma \vdash_{\mathsf{v}} v\colon A]\!]$;
- $[\![\Gamma \vdash_{\mathsf{c}} \mathsf{return}\, v\colon A]\!] = J[\![\Gamma \vdash_{\mathsf{v}} v\colon A]\!]$;
- $[\![\Gamma \vdash_{\mathsf{c}} \mathsf{do}\, x \leftarrow p;\, q\colon B]\!] = [\![x\colon A \vdash_{\mathsf{c}} q\colon B]\!] \circ [\![\Gamma \vdash_{\mathsf{c}} p\colon A]\!]$;
- $[\![\Gamma \vdash_{\mathsf{v}} \mathsf{init}\, v\colon A]\!] = \,!$;
- $[\![\Gamma \vdash_{\mathsf{v}} \mathsf{inl}\, v\colon A + B]\!] = \mathsf{inl} \circ [\![\Gamma \vdash_{\mathsf{v}} v\colon A]\!]$;
- $[\![\Gamma \vdash_{\mathsf{v}} \mathsf{inr}\, v\colon A + B]\!] = \mathsf{inr} \circ [\![\Gamma \vdash_{\mathsf{v}} v\colon B]\!]$;
- $[\![\Gamma \vdash_{\mathsf{c}} \mathsf{case}\, v\, \mathsf{of}\, \mathsf{inl}\, x \mapsto p;\, \mathsf{inr}\, y \mapsto q\colon C]\!]$
  $= [[\![x\colon A \vdash_{\mathsf{c}} p\colon C]\!], [\![y\colon B \vdash_{\mathsf{c}} q\colon C]\!]] \circ J[\![\Gamma \vdash_{\mathsf{v}} v\colon A + B]\!]$.

As observed by Power and Robinson [32] (cf. [34, 0.1]), monads arise from the requirement that J is a left adjoint, thus simple FGCBV can be interpreted w.r.t. a monad on $\mathbf{V}$:

▶ **Proposition 4.** *Let $J\colon \mathbf{V} \to \mathbf{C}$ be an identity-on-objects functor. Then J is a left adjoint iff $\mathbf{C}$ is isomorphic to a Kleisli category of some monad $\mathbf{T}$ on $\mathbf{V}$ and $Jf = H(\eta \circ f)$ for all $f \in \mathbf{V}(X, Y)$ where $H\colon \mathbf{V}_{\mathbf{T}} \cong \mathbf{C}$ is the relevant isomorphism.*

Moreover, in this situation, finite coproducts in $\mathbf{C}$ are inherited from those in $\mathbf{V}$, i.e. $J!$ is the initial morphism in $\mathbf{C}$ and $(A + B, J\,\mathsf{inl}, J\,\mathsf{inr})$ is a binary coproduct in $\mathbf{C}$.

[2] $\dot{-}$ refers to truncated subtraction: $x \dot{-} y = x - y$ if $x \geqslant y$, and $x \dot{-} y = 0$ otherwise.

$$\frac{x\colon A \;\text{ in }\; \Gamma}{\Gamma \vdash_{\mathsf{v}} x\colon A} \qquad \frac{f\colon A \to B \in \Sigma_v \quad \Gamma \vdash_{\mathsf{v}} v\colon A}{\Gamma \vdash_{\mathsf{v}} f(v)\colon B} \qquad \frac{f\colon A \to B \in \Sigma_c \quad \Gamma \vdash_{\mathsf{v}} v\colon A}{\Gamma \vdash_{\mathsf{c}} f(v)\colon B}$$

$$\frac{\Gamma \vdash_{\mathsf{v}} v\colon A}{\Gamma \vdash_{\mathsf{c}} \mathsf{return}\, v\colon A} \qquad \frac{\Gamma \vdash_{\mathsf{c}} p\colon A \quad \Gamma, x\colon A \vdash_{\mathsf{c}} q\colon B}{\Gamma \vdash_{\mathsf{c}} \mathsf{do}\, x \leftarrow p;\, q\colon B} \qquad \frac{\Gamma \vdash_{\mathsf{v}} v\colon 0}{\Gamma \vdash_{\mathsf{v}} \mathsf{init}\, v\colon A}$$

$$\frac{\Gamma \vdash_{\mathsf{v}} v\colon A}{\Gamma \vdash_{\mathsf{v}} \mathsf{inl}\, v\colon A + B} \qquad \frac{\Gamma \vdash_{\mathsf{v}} v\colon B}{\Gamma \vdash_{\mathsf{v}} \mathsf{inr}\, v\colon A + B} \qquad \frac{\Gamma \vdash_{\mathsf{v}} v\colon A + B \quad x\colon A \vdash_{\mathsf{c}} p\colon C \quad y\colon B \vdash_{\mathsf{c}} q\colon C}{\Gamma \vdash_{\mathsf{c}} \mathsf{case}\, v \,\mathsf{of}\, \mathsf{inl}\, x \mapsto p;\, \mathsf{inr}\, y \mapsto q\colon C}$$

$$\frac{\Gamma \vdash_{\mathsf{v}} v\colon A \qquad \Gamma \vdash_{\mathsf{v}} w\colon B}{\Gamma \vdash_{\mathsf{v}} \langle v, w\rangle\colon A \times B} \qquad \frac{\Gamma \vdash_{\mathsf{v}} p\colon A \times B \qquad \Gamma, x\colon A, y\colon B \vdash_{\mathsf{c}} q\colon C}{\Gamma \vdash_{\mathsf{c}} \mathsf{case}\; p \;\mathsf{of}\; \langle x, y\rangle \mapsto q\colon C}$$

Figure 3 FGCBV with coproducts.

▶ **Example 5** (Monads). Let us recall relevant monads on $\mathbf{V} = \mathbf{Set}$ for further reference.

1. $TX = \nu\gamma.\,\mathcal{P}_\omega((X+1)+A\times\gamma)$ where $\mathcal{P}_\omega$ is the finite powerset functor and $\nu\gamma.\,F\gamma$ denotes a final F-coalgebra. This monad provides a standard strong bisimulation semantics for BPA (Example 1). The denotations in TX are finitely branching trees with edges labelled by actions and with terminal nodes labelled in X (free variables) or in 1 (successful termination). This monad is an instance of the *coinductive resumption monad* [30].
2. $TX = \mathcal{P}(A^\star \times (X+1) + A^\star)$ is the monad of finite traces (terminating successfully $A^\star \times (X+1)$ and divergent $A^\star$), which can again be used as a semantics of Example 1.
3. $TX = \mathcal{P}(A^\star \times (X+1) + (A^\star + A^\omega))$ is a refinement of **2.** collecting not only finite, but also infinite traces. If we extend BPA with countable non-determinism, we obtain a semantics properly between strong bisimilarity finite trace equivalence. For example, the equation $P = a \cdot P$ produces the infinite trace a^ω and $P' = \sum_{i\in\mathbb{N}} P_i$ with $P_0 = a$ and $P_{i+1} = a \cdot P_i$ does not, and P is finite trace equivalent to $P + P'$, but not infinite trace equivalent.
4. $TX = (\nu\gamma.\,X \times S + \gamma \times S)^S$ can be for Example 2. In **Set**, $TX \cong (X \times S^+ + S^\omega)^S$, i.e. an element TX is isomorphic to a function that takes an initial state in S and returns either a finite trace in $X \times S^+$ or an infinite trace in S^ω. We can use Proposition 4 to argue that T indeed extends to a monad. Let $\mathbf{C}$ be the category with $\mathbf{C}(X,Y) = \mathbf{Set}(X \times S, \nu\gamma.\,Y \times S + \gamma \times S)$, which is a full subcategory of the Kleisli category of the coinductive resumption monad $\nu\gamma.\,(\text{--} + \gamma \times S)$. Now, the obvious identity-on-objects functor $J\colon \mathbf{Set} \to \mathbf{C}$ is a left adjoint, which yields the original T.
5. $TX = \mathbb{R}_{\geqslant 0} \times X + \bar{\mathbb{R}}_{\geqslant 0}$ is a monad, which can be used for Example 3. Here $\mathbb{R}_{\geqslant 0} \times X$ refers to terminating behaviours and $\bar{\mathbb{R}}_{\geqslant 0} = \mathbb{R}_{\geqslant 0} \cup \{\infty\}$ to Zeno and infinite behaviours.

4 Freyd Categories and Strong Monads

The full FGCBV (with coproducts) is obtained by extending the type syntax (2) with products $A \times B$, and by replacing the rules in Figure 2 with the rules in Figure 3. We now assume that variable contexts Γ are (possibly empty) lists $(x_1\colon A_1, \ldots, x_n\colon A_n)$ with non-repetitive $x_1, \ldots, x_n$. To interpret the resulting language, again, we need an identity-on-objects functor $J\colon \mathbf{V} \to \mathbf{C}$, an action of $\mathbf{V}$ on $\mathbf{C}$, and J to preserve this action.

▶ **Definition 6** (Actegory [20]). *Let $(\mathbf{V}, \otimes, I)$ be a monoidal category. Then an* action *of $\mathbf{V}$ on a category $\mathbf{C}$ is a functor $\oslash\colon \mathbf{V} \times \mathbf{C} \to \mathbf{C}$ together with the* unitor *and the* actor *natural isomorphisms $\upsilon\colon 1 \oslash X \cong X$, $\alpha\colon X \oslash (Y \oslash Z) \cong (X \otimes Y) \oslash Z$, satisfying expected coherence conditions. Then $\mathbf{C}$ is called an* ($\mathbf{V}$-)actegory*.*

Note that every monoidal category trivially acts on itself via $\oslash = \otimes$. In the sequel, we will only consider *Cartesian* categories, i.e. actegories w.r.t. $(\mathbf{V}, \times, 1)$.

▶ **Definition 7** ((Distributive) Freyd Category [23, 36])**.** *A* Freyd category $(\mathbf{V}, \mathbf{C}, J(-), \oslash)$ *consists of the following data:*

- *a Cartesian category* $\mathbf{V}$*;*
- *a category* $\mathbf{C}$ *with* $|\mathbf{V}| = |\mathbf{C}|$*;*
- *an identity-on-objects functor* $J\colon \mathbf{V} \to \mathbf{C}$*;*
- *a monoidal action of* $\mathbf{V}$ *on* $\mathbf{C}$*, such that* J *preserves the* $\mathbf{V}$*-action, i.e.* $J(f \times g) = f \oslash Jg$ *for all* $f \in \mathbf{V}(X, X')$*,* $g \in \mathbf{V}(Y, Y')$*,* $\upsilon = J\,\mathsf{snd}$ *and* $\alpha = J\langle \mathsf{id} \times \mathsf{fst}, \mathsf{snd} \circ \mathsf{snd}\rangle$*.*

A Freyd category $(\mathbf{V}, \mathbf{C}, J(-), \oslash)$ *is* distributive *if* $\mathbf{V}$ *is distributive,* $\mathbf{C}$ *is co-Cartesian and* J *strictly preserves coproducts (this is equivalent to the requirement that the action preserves binary coproducts in the second argument coherently with* dist*).*

Given a distributive Freyd category $(\mathbf{V}, \mathbf{C}, J(-), \oslash)$, we update the semantics from Section 3 as follows, where $[\![A \times B]\!] = [\![A]\!] \times [\![B]\!]$, $[\![x_1\colon A_1, \ldots, x_n\colon A_n]\!] = [\![A_1]\!] \times \ldots \times [\![A_n]\!]$:

- $[\![x_1\colon A_1, \ldots, x_n\colon A_n \vdash_{\mathsf{v}} x_i\colon A_i]\!] = \mathsf{proj}_i$;
- $[\![\Gamma \vdash_{\mathsf{c}} \mathsf{do}\, x \leftarrow p;\, q\colon B]\!] = [\![\Gamma, x\colon A \vdash_{\mathsf{c}} q\colon B]\!] \circ (\mathsf{id} \oslash [\![\Gamma \vdash_{\mathsf{c}} p\colon A]\!]) \circ \Delta$;
- $[\![\Gamma \vdash_{\mathsf{c}} \mathsf{case}\, v\, \mathsf{of}\, \mathsf{inl}\, x \mapsto p;\, \mathsf{inr}\, y \mapsto q\colon C]\!]$
  $= \big[[\![\Gamma, x\colon A \vdash_{\mathsf{c}} p\colon C]\!], [\![\Gamma, y\colon B \vdash_{\mathsf{c}} q\colon C]\!]\big] \circ J\,\mathsf{dist} \circ (\mathsf{id} \oslash J[\![\Gamma \vdash_{\mathsf{v}} v\colon A + B]\!]) \circ J\Delta$;
- $[\![\Gamma \vdash_{\mathsf{v}} \langle v, w\rangle\colon A \times B]\!] = \langle [\![\Gamma \vdash_{\mathsf{v}} v\colon A]\!], [\![\Gamma \vdash_{\mathsf{v}} w\colon B]\!]\rangle$.

Freyd categories are to strong monads as identity-on-objects functors to monads.

▶ **Proposition 8.** *Let* $(\mathbf{V}, \mathbf{C}, J(-), \oslash)$ *be a Freyd category. Then* J *is a left adjoint iff* $\mathbf{C}$ *is isomorphic to a Kleisli category of some strong monad* $\mathbf{T}$ *on* $\mathbf{V}$ *and* $Jf = H(\eta \circ f)$ *for all* $f \in \mathbf{V}(X, Y)$ *where* $H\colon \mathbf{V}_{\mathbf{T}} \cong \mathbf{C}$ *is the relevant isomorphism.*

Proposition 8 allows us to refactor the existing characterization of *closed Freyd categories* [25, Theorem 7.3] along the following lines. In order to include higher-order types to the language, we would need to add $A \to B$ as a new type former and the following term formation rules:

$$\frac{\Gamma, x\colon A \vdash_{\mathsf{c}} p\colon B}{\Gamma \vdash_{\mathsf{v}} \lambda x.\, p\colon A \to B} \qquad\qquad \frac{\Gamma \vdash_{\mathsf{v}} w\colon A \qquad \Gamma \vdash_{\mathsf{v}} v\colon A \to B}{\Gamma \vdash_{\mathsf{c}} vw\colon B}$$

We then need to provide the following additional semantic clauses:

- $[\![\Gamma \vdash_{\mathsf{v}} \lambda x.\, p\colon A \to B]\!] = \mathsf{curry}[\![\Gamma, x\colon A \vdash_{\mathsf{c}} p\colon B]\!]$;
- $[\![\Gamma \vdash_{\mathsf{c}} vw\colon B]\!] = \mathsf{curry}^{-1}[\![\Gamma \vdash_{\mathsf{v}} v\colon A \to B]\!] \circ (\mathsf{id} \oslash J[\![\Gamma \vdash_{\mathsf{v}} w\colon A]\!]) \circ J\Delta$,

where $[\![A \to B]\!] = [\![A]\!] \multimap [\![B]\!]$, $\multimap\colon |\mathbf{V}| \times |\mathbf{C}| \to |\mathbf{C}|$, and curry is an isomorphism

$$\mathsf{curry}\colon \mathbf{C}(J(X \times A), B) \cong \mathbf{V}(X, A \multimap B) \tag{4}$$

natural in X. In particular, this says that J is left adjoint to $1 \multimap (-)$, which, as we have seen in Proposition 4, means that $\mathbf{C}$ is isomorphic to the Kleisli category of a strong monad $\mathbf{T}$, and hence (4) amounts to $\mathbf{V}(X \times A, TB) \cong \mathbf{V}(X, A \multimap B)$, i.e. to the existence of *Kleisli exponentials*, which are exponentials of the form $(TB)^A$. We thus obtain the following

▶ **Corollary 9.** *Let* $(\mathbf{V}, \mathbf{C}, J(-), \oslash)$ *be a Freyd category. The following are equivalent:*

- *an isomorphism* (4) *natural in* X *exists;*
- *for all* $A \in |\mathbf{V}|$*,* $J(- \times A)\colon \mathbf{V} \to \mathbf{C}$ *is a left adjoint;*
- $\mathbf{C}$ *is isomorphic to a Kleisli category of a strong monad, and Kleisli exponentials exist.*

A yet another way to express (4) is to state that the presheaves $\mathbf{C}(J(- \times A), B)\colon \mathbf{V}^{\mathsf{op}} \to \mathbf{Set}$ are representable. We will use this formulation in our subsequent analysis of guardedness.

5 Guarded Freyd Categories

We proceed to recall the formal notion of guardedness [19, 24].

▶ **Definition 10** (Guardedness). *A* guardedness predicate *on a co-Cartesian category* $\mathbf{C}$ *provides for all* X, Y, $Z \in |\mathbf{C}|$ *a subset* $\mathbf{C}_\bullet(X,Y,Z) \subseteq \mathbf{C}(X, Y+Z)$*, whose elements we write as* $f\colon X \to Y \rangle Z$ *and call guarded (in* Z*), such that*

$$(\mathbf{trv}_\bullet)\ \dfrac{f\colon X \to Y}{\mathsf{inl}\circ f\colon X \to Y \rangle Z} \qquad (\mathbf{par}_\bullet)\ \dfrac{f\colon X \to V \rangle W \qquad g\colon Y \to V \rangle W}{[f,g]\colon X + Y \to V \rangle W}$$

$$(\mathbf{cmp}_\bullet)\ \dfrac{f\colon X \to Y \rangle Z \qquad g\colon Y \to V \rangle W \qquad h\colon Z \to V + W}{[g,h]\circ f\colon X \to V \rangle W}$$

A guarded category *is a category, equipped with a guardedness predicate. A* guarded functor *between two guarded categories is a functor* $F\colon \mathbf{C} \to \mathbf{D}$ *that strictly preserves coproducts, and preserves guardedness in the following sense:* $f \in \mathbf{C}_\bullet(X,Y,Z)$ *entails* $f \in \mathbf{D}_\bullet(FX,FY,FZ)$*.*

Intuitively, $\mathbf{C}_\bullet(X,Y,Z)$ axiomatically and compositionally distinguishes those morphisms $X \to Y + Z$ for which the program flow from X to Z is guarded, in particular, if $X = Z$ then the corresponding guarded loop can be safely closed. Note that the standard (total) iteration is an instance with $\mathbf{C}_\bullet(X,Y,Z) = \mathbf{C}(X, Y+Z)$. Consider other instances.

▶ **Example 11** (Vacuous Guardedness [18]). The least guardedness predicate is as follows: $\mathbf{C}_\bullet(X,Y,Z) = \{\mathsf{inl}\circ f\colon X \to Y + Z \mid f \in \mathbf{C}(X,Y)\}$. Such $\mathbf{C}$ is called *vacuously guarded.*

▶ **Example 12** (Coalgebraic Resumptions). Let $\mathbf{T}$ be a monad on a co-Cartesian category $\mathbf{V}$ and let $H\colon \mathbf{V} \to \mathbf{V}$ be an endofunctor such that all fixpoints $T_H X = \nu\gamma.\, T(X + H\gamma)$ exist. These extend to a monad $\mathbf{T}_H$, called the *(generalized) coalgebraic resumption monad (transform)* of $\mathbf{T}$ [30, 19]. The Kleisli category of $\mathbf{T}_H$ is guarded with $f\colon X \to Y \rangle Z$ if

$$\begin{tikzcd}
X \arrow[r, "g"] \arrow[d, "f"'] & T(Y + HT_H(Y+Z)) \arrow[d, "T(\mathsf{inl}+\mathsf{id})"] \\
T_H(Y+Z) \arrow[r, "\mathsf{out}"] & T((Y+Z) + HT_H(Y+Z))
\end{tikzcd}$$

for some $g\colon X \to T(Y + HT_H(Y+Z))$. Guarded iteration operators canonically extend from $\mathbf{T}$ to $\mathbf{T}_H$ [24].

▶ **Example 13** (Algebraic Resumptions). A simple variation of the previous example involves least fixpoints $T^H X = \mu\gamma.\, T(X + H\gamma)$ instead of the greatest ones and in^{-1} instead of out where $\mathsf{in}\colon T(X + HT^H X) \to T^H X$ is the initial algebra structure of $T^H X$, which is an isomorphism by Lambek's lemma. However, we can no longer generally induce non-trivial (guarded) iteration operators for $\mathbf{T}^H$.

▶ **Example 14.** Let us describe natural guardedness predicates on the Kleisli categories of monads from Example 5.

1. $TX = \nu\gamma.\, \mathcal{P}_\omega((X+1) + A \times \gamma)$ is a special case of Example 12.
2. For $TX = \mathcal{P}(A^\star \times (X+1) + A^\star)$, let $f\colon X \to Y \rangle Z$ if for every $x \in X$, $\mathsf{inl}(w, \mathsf{inl}\,\mathsf{inr}\, y) \in f(x)$ entails $w \neq \epsilon$.
3. For $TX = \mathcal{P}(A^\star \times (X+1) + (A^\star + A^\omega))$ guardedness is defined as in clause **2.**
4. For $TX = (\nu\gamma.\, X \times S + \gamma \times S)^S$, recall that $\mathbf{Set}_{\mathbf{T}}$ is isomorphic to a full subcategory of the Kleisli category of $\nu\gamma.\,(\text{–} + \gamma \times S)$, which is again an instance of Example 12 with $TX = X$ and $HX = X \times S$. The guardedness predicate for $\mathbf{T}$ thus restricts accordingly.
5. For $TX = (\mathbb{R}_{\geqslant 0} \times X) + \bar{\mathbb{R}}_{\geqslant 0}$ let $f\colon X \to Y \rangle Z$ if $f(x) = \mathsf{inl}\,(r, \mathsf{inr}\, z)$ implies $r > 0$.

$$\frac{x\colon A\ \text{ in }\ \Gamma}{\Gamma \vdash_{\mathsf{v}} x\colon A} \qquad \frac{f\colon A \to B \in \Sigma_v \quad \Gamma \vdash_{\mathsf{v}} v\colon A}{\Gamma \vdash_{\mathsf{v}} f(v)\colon B} \qquad \frac{f\colon A \to B \rangle C \in \Sigma_c \qquad \Gamma \vdash_{\mathsf{v}} v\colon A}{\Gamma \vdash_{\mathsf{c}} f(v)\colon B \rangle C}$$

$$\frac{\Gamma \vdash_{\mathsf{v}} v\colon A}{\Gamma \vdash_{\mathsf{c}} \mathsf{return}\, v\colon A \rangle B} \qquad \frac{\Gamma \vdash_{\mathsf{c}} p\colon A \rangle B \quad \Gamma, x\colon A \vdash_{\mathsf{c}} q\colon C \rangle D \quad \Gamma, y\colon B \vdash_{\mathsf{c}} r\colon C + D \rangle 0}{\Gamma \vdash_{\mathsf{c}} \mathsf{docase}\, p\, \mathsf{of}\ \mathsf{inl}\, x \mapsto q;\ \mathsf{inr}\, y \mapsto r\colon C \rangle D}$$

$$\frac{\Gamma \vdash_{\mathsf{v}} v\colon 0}{\Gamma \vdash_{\mathsf{v}} \mathsf{init}\, v\colon A} \qquad \frac{\Gamma \vdash_{\mathsf{v}} v\colon A}{\Gamma \vdash_{\mathsf{v}} \mathsf{inl}\, v\colon A + B} \qquad \frac{\Gamma \vdash_{\mathsf{v}} v\colon B}{\Gamma \vdash_{\mathsf{v}} \mathsf{inr}\, v\colon A + B}$$

$$\frac{\Gamma \vdash_{\mathsf{v}} v\colon A + B \qquad \Gamma, x\colon A \vdash_{\mathsf{c}} p\colon C \rangle D \qquad \Gamma, y\colon B \vdash_{\mathsf{c}} q\colon C \rangle D}{\Gamma \vdash_{\mathsf{c}} \mathsf{case}\, v\, \mathsf{of}\ \mathsf{inl}\, x \mapsto p;\ \mathsf{inr}\, y \mapsto q\colon C \rangle D}$$

$$\frac{\Gamma \vdash_{\mathsf{v}} v\colon A \qquad \Gamma \vdash_{\mathsf{v}} w\colon B}{\Gamma \vdash_{\mathsf{v}} \langle v, w\rangle\colon A \times B} \qquad \frac{\Gamma \vdash_{\mathsf{v}} p\colon A \times B \qquad \Gamma, x\colon A, y\colon B \vdash_{\mathsf{c}} q\colon C \rangle D}{\Gamma \vdash_{\mathsf{c}} \mathsf{case}\ p\ \mathsf{of}\ \langle x, y\rangle \mapsto q\colon C \rangle D}$$

Figure 4 Term formation rules of guarded FGCBV.

We proceed to extend the language of Figure 3 with guardedness data. As before, Σ_v consists of constructs of the form $f\colon A \to B$, while Σ_c consists of constructs of the form $f\colon A \to B \rangle C$. In Figure 4 we display the new formation rules that replace their counterparts from Figure 3. The rule for return corresponds to the $(\mathbf{trv}_\bullet)$ rule. The rule for do now handles the guarded and unguarded branches, as prescribed by the $(\mathbf{cmp}_\bullet)$ rule, that is,

$$\mathsf{docase}\, p\, \mathsf{of}\ \mathsf{inl}\, x \mapsto q;\ \mathsf{inr}\, y \mapsto r$$

is meant to have the same semantics as $\mathsf{do}\, z \leftarrow p;\ \mathsf{case}\, z\, \mathsf{of}\ \mathsf{inl}\, x \mapsto q;\ \mathsf{inr}\, y \mapsto r$, modulo guardedness information. The rule $(\mathbf{par}_\bullet)$ corresponds to the rule for case, which is essentially unchanged w.r.t. Figure 3. Note that a guarded iteration operator could be added with the following rule:

$$\frac{\Gamma \vdash_{\mathsf{c}} p\colon A \rangle 0 \qquad \Gamma, x\colon A \vdash_{\mathsf{c}} q\colon B \rangle C + A}{\Gamma \vdash_{\mathsf{c}} \mathsf{iter}\, x \leftarrow p;\ q\colon B \rangle C}$$

The rule $(\mathbf{par}_\bullet)$ corresponds to the rule for case. For every $\Gamma \vdash_{\mathsf{c}} p\colon A \rangle B + C$ we can construct

$$\begin{aligned}\Gamma \vdash_{\mathsf{c}} \mathsf{docase}\, p\, \mathsf{of}\ \mathsf{inl}\, x \mapsto{}& \mathsf{return}(\mathsf{inl}\, x);\\ \mathsf{inr}\, z \mapsto{}& \mathsf{case}\, z\, \mathsf{of}\ \mathsf{inl}\, x \mapsto \mathsf{return}(\mathsf{inl}\, \mathsf{inr}\, x);\\ &\qquad\qquad \mathsf{inr}\, y \mapsto \mathsf{return}(\mathsf{inr}\, y)\colon A + B \rangle C,\end{aligned}$$

which means that weakening the guardedness guarantee is expressible.

▶ **Example 15.** The updated effectful signature of Example 1 now involves $a\colon 1 \to 0 \rangle 1$ and $\mathsf{toss}\colon 1 \to 2 \rangle 0$, indicating that actions guard everything, while nondeterminism guards nothing. The signature Σ_c from Example 2 can be refined to $\{\mathsf{put}\colon S \to 0 \rangle 1, \mathsf{get}\colon 1 \to S \rangle 0\}$, meaning again that put guards everything and get guards nothing. Example 3 is more subtle since $\mathsf{wait}\colon \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ is meant to be guarded only for non-zero inputs. We thus can embed the involved case distinction into wait by redefining it as $\mathsf{wait}\colon \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0} \rangle \mathbb{R}_{\geqslant 0}$.

▶ **Definition 16** (Guarded Freyd Category). *A distributive Freyd category* $(\mathbf{V}, \mathbf{C}, J(-), \oslash)$ *is guarded if* $\mathbf{C}$ *is guarded and the action of* $\mathbf{V}$ *on* $\mathbf{C}$ *preserves guardedness in the following sense: Given* $f \in \mathbf{V}(A, B)$, $g \in \mathbf{C}_\bullet(X, Y, Z)$, $J\,\mathsf{dist} \circ (f \oslash g) \in \mathbf{C}_\bullet(A \times X, B \times Y, B \times Z)$.

The semantics of (Σ_v, Σ_c) over a guarded Freyd category $(\mathbf{V}, \mathbf{C}, J(\text{-}), \oslash)$ interprets types and operations from Σ_v as before and sends each $f\colon A \to B \rangle C \in \Sigma_c$ to $[\![f]\!] \in \mathbf{C}_\bullet([\![A]\!], [\![B]\!], [\![C]\!])$. Terms in context are interpreted as $[\![\Gamma \vdash_{\mathsf{v}} v\colon B]\!] \in \mathbf{V}([\![A]\!], [\![B]\!])$ and $[\![\Gamma \vdash_{\mathsf{c}} p\colon B \rangle C]\!] \in \mathbf{C}([\![A]\!], [\![B]\!] + [\![C]\!])$ as follows:

- $[\![x_1\colon A_1, \ldots, x_n\colon A_n \vdash_{\mathsf{v}} x_i\colon A_i]\!] = \mathsf{inj}_i$;
- $[\![\Gamma \vdash_{\mathsf{v}} f(v)\colon B]\!] = [\![f]\!] \circ [\![\Gamma \vdash_{\mathsf{v}} v\colon A]\!]$;
- $[\![\Gamma \vdash_{\mathsf{c}} f(v)\colon B]\!] = [\![f]\!] \circ J[\![\Gamma \vdash_{\mathsf{v}} v\colon A]\!]$;
- $[\![\Gamma \vdash_{\mathsf{c}} \mathsf{return}\, v\colon A \rangle B]\!] = J\,\mathsf{inl} \circ J[\![\Gamma \vdash_{\mathsf{v}} v\colon A]\!]$;
- $[\![\Gamma \vdash_{\mathsf{c}} \mathsf{docase}\, p\, \mathsf{of}\, \mathsf{inl}\, x \mapsto q;\ \mathsf{inr}\, y \mapsto r\colon C \rangle D]\!]$
  $= \big[[\![\Gamma, x\colon A \vdash_{\mathsf{c}} q\colon C \rangle D]\!], J[\mathsf{id}, !] \circ [\![\Gamma, y\colon B \vdash_{\mathsf{c}} r\colon C + D \rangle 0]\!]\big]$
  $\circ\, J\,\mathsf{dist} \circ (\mathsf{id} \oslash [\![\Gamma \vdash_{\mathsf{c}} p\colon A \rangle B]\!]) \circ \Delta$;
- $[\![\Gamma \vdash_{\mathsf{v}} \mathsf{init}\, v\colon A]\!] = !$;
- $[\![\Gamma \vdash_{\mathsf{v}} \mathsf{inl}\, v\colon A + B]\!] = \mathsf{inl} \circ [\![\Gamma \vdash_{\mathsf{v}} v\colon A]\!]$;
- $[\![\Gamma \vdash_{\mathsf{v}} \mathsf{inr}\, v\colon A + B]\!] = \mathsf{inr} \circ [\![\Gamma \vdash_{\mathsf{v}} v\colon B]\!]$;
- $[\![\Gamma \vdash_{\mathsf{c}} \mathsf{case}\, v\, \mathsf{of}\, \mathsf{inl}\, x \mapsto p;\ \mathsf{inr}\, y \mapsto q\colon C \rangle D]\!]$
  $= \big[[\![\Gamma, x\colon A \vdash_{\mathsf{c}} p\colon C \rangle D]\!], [\![\Gamma, y\colon B \vdash_{\mathsf{c}} q\colon C \rangle D]\!]\big]$
  $\circ\, J\,\mathsf{dist} \circ (\mathsf{id} \oslash J[\![\Gamma \vdash_{\mathsf{v}} v\colon A + B]\!]) \circ J\Delta$.
- $[\![\Gamma \vdash_{\mathsf{v}} \langle v, w\rangle\colon A \times B]\!] = \langle [\![\Gamma \vdash_{\mathsf{v}} v\colon A]\!], [\![\Gamma \vdash_{\mathsf{v}} w\colon B]\!]\rangle$.

This is well-defined, which can be easily shown by structural induction:

▶ **Proposition 17.** *For any derivable $\Gamma \vdash_{\mathsf{c}} p\colon A \rangle B$, $[\![\Gamma \vdash_{\mathsf{c}} p\colon A \rangle B]\!] \in \mathbf{C}_\bullet([\![A]\!], [\![B]\!], [\![C]\!])$.*

6 Representing Guardedness

In Section 4 we explored the combination of strength (i.e. multivariable contexts) and representability of presheaves $\mathbf{C}(J(\text{-}), X)\colon \mathbf{V}^{\mathsf{op}} \to \mathbf{Set}$, sticking to the bottom face of the cube in Figure 1. Our plan is to obtain further concepts via representability of $\mathbf{C}_\bullet(J(\text{-}), X, Y)\colon \mathbf{V}^{\mathsf{op}} \to \mathbf{Set}$. Note that representability of guardedness jointly with function spaces amounts to representability of $\mathbf{C}_\bullet(J(\text{-} \times X), Y, Z)\colon \mathbf{V}^{\mathsf{op}} \to \mathbf{Set}$, i.e. existence of an endofunctor $\multimap\colon \mathbf{V}^{\mathsf{op}} \times \mathbf{C} \times \mathbf{C} \to \mathbf{C}$, such that $\mathbf{C}_\bullet(J(\text{-} \times X), Y, Z) \cong \mathbf{V}(\text{-}, X \multimap_Z Y)$. This is exactly the structure, one would need to extend Figure 4 with function spaces as follows:

$$\frac{\Gamma, x\colon A \vdash_{\mathsf{c}} p\colon B \rangle C}{\Gamma \vdash_{\mathsf{v}} \lambda x.\, p\colon A \to_C B} \qquad\qquad \frac{\Gamma \vdash_{\mathsf{v}} w\colon A \qquad \Gamma \vdash_{\mathsf{v}} v\colon A \to_C B}{\Gamma \vdash_{\mathsf{c}} vw\colon B \rangle C}$$

The decorated function spaces $A \to_C B$ then can be interpreted as $[\![A]\!] \multimap_{[\![C]\!]} [\![B]\!]$, which is a subobject of the Kleisli exponential $[\![A]\!] \to T([\![B]\!] + [\![C]\!])$, consisting of guarded morphisms.

▶ **Definition 18.** *Given $J\colon \mathbf{V} \to \mathbf{C}$, where $\mathbf{C}$ is guarded co-Cartesian, we call the guardedness predicate $\mathbf{C}_\bullet$ J-representable if for all $X, Y \in |\mathbf{C}|$ the presheaf $\mathbf{C}_\bullet(J(\text{-}), X, Y)\colon \mathbf{V}^{\mathsf{op}} \to \mathbf{Set}$ is representable; $\mathbf{C}_\bullet$ J-guarded if it is equipped with a J-representable guardedness predicate.*

J-representability means that for all $X, Y \in |\mathbf{C}|$ there is $U(X, Y) \in |\mathbf{V}|$ such that $\mathbf{C}_\bullet(JZ, X, Y) \cong \mathbf{V}(Z, U(X, Y))$ naturally in Z. Hence, J-representability of guardedness entails that J is a left adjoint. This can be formulated in terms of *free objects* [1].

▶ **Lemma 19.** *Given an identity-on-objects guarded functor $J\colon \mathbf{V} \to \mathbf{C}$ (with $\mathbf{V}$ regarded as vacuously guarded), $\mathbf{C}_\bullet$ is J-representable iff*

- *there is a family of objects $(U(X, Y) \in |\mathbf{V}|)_{X, Y \in |\mathbf{C}|}$;*
- *there is a family of guarded morphisms $(\epsilon_{X,Y}\colon U(X, Y) \to X \rangle Y)_{X, Y \in |\mathbf{C}|}$;*

- *there is an operator* $(\text{–})^{\natural}\colon \mathbf{C}_{\bullet}(Z,X,Y) \to \mathbf{V}(Z,U(X,Y))$ *sending each* $f\colon Z \to X \rangle Y$ *to the unique morphism* $f^{\natural}$ *for which the diagram*

$$\begin{tikzcd} & U(X,Y) \arrow[d, "\epsilon_{X,Y}"] \\ X \arrow[ru, "Jf^{\natural}"] \arrow[r, "f"'] & X+Y \end{tikzcd}$$

  commutes.

These conditions entail that U *is a bifunctor and that* $\epsilon_{X,Y}$ *is natural in* X *and* Y*.*

▶ **Lemma 20.** *If* $\mathbf{C}$ *is* J*-guarded co-Cartesian then* $J \dashv U(\text{–},0)$ *with* U *as in Lemma 19.*

By Lemma 20, representability fails already if J fails to be adjoint. Instructive examples of non-representability are thus only those, where J does have a right adjoint.

▶ **Proposition 21.** *Let* $\mathbf{T}$ *be a monad over the category of sets* **Set** *with the axiom of choice. If* $\mathbf{Set}_{\mathbf{T}}$ *is guarded, the guardedness predicate is representable iff every* $f\colon X \to T(Y+Z)$ *is guarded whenever all the compositions* $1 \hookrightarrow X \xrightarrow{f} T(Y+Z)$ *are guarded.*

▶ **Example 22** (Failure of Representability)**.** In **Set**, let $f\colon X \to Y+Z$ be guarded in Z if $\{z \in Z \mid f^{-1}(\mathsf{inr}\, z) \neq \emptyset\}$ is finite. The axioms of guardedness are easy to verify. By Proposition 21, this predicate is not Id-representable, as any $1 \hookrightarrow X \xrightarrow{\mathsf{inr}} 0+X$ is guarded, but inr is not if X is infinite.

In what follows, we will be using $\#$ as a binary operation that binds stronger than monoidal products $(\otimes, +, \ldots)$, so e.g. $X \otimes Y \# Z$ will read as $X \otimes (Y \# Z)$.

▶ **Theorem 23.** *Given an identity-on-objects guarded* $J\colon \mathbf{V} \to \mathbf{C}$*,* $\mathbf{C}_{\bullet}$ *is* J*-representable iff*

- *there is a bifunctor* $\#\colon \mathbf{V} \times \mathbf{V} \to \mathbf{V}$*, such that* $\text{–}\#0$ *is a monad and* $\mathbf{C} \cong \mathbf{V}_{\text{–}\#0}$*;*
- *there is a family of guarded morphisms (w.r.t. the guardedness predicate, induced by* $\mathbf{C} \cong \mathbf{V}_{\text{–}\#0}$*)* $(\epsilon_{X,Y}\colon X \# Y \to X \rangle Y)_{X,Y \in |\mathbf{V}|}$*, natural in* X *and* Y*;*
- *for every guarded* $f\colon X \to Y \rangle Z$ *there is unique* $f^{\sharp}\colon X \to Y \# Z$*, such that the diagram*

$$\begin{tikzcd} & Y \# Z \arrow[d, "\epsilon_{Y,Z}"] \\ X \arrow[ru, "f^{\sharp}"] \arrow[r, "f"'] & (Y+Z)\#0 \end{tikzcd}$$

  commutes.

Theorem 23 provides a bijective correspondence between morphisms $f\colon X \to Y \rangle Z$ in $\mathbf{C}$ and the morphisms $f^{\sharp}\colon X \to Y \# Z$ in $\mathbf{V}$, representing them. Uniqueness of the $f^{\sharp}$ is easily seen to be equivalent to the monicity of the $\epsilon_{X,Z}$.

7 Guarded Parametrized Monads

Theorem 23 describes guardedness as a certain bifunctor $\#\colon \mathbf{V} \times \mathbf{V} \to \mathbf{V}$ and a family of morphisms $(\epsilon_{X,Y})_{X,Y \in |\mathbf{V}|}$, so that the guardedness predicate is derivable. However, the guardedness laws are still formulated in terms of this predicate, and not in terms of $\#$ and ϵ. To resolve this issue, we must identify a collection of canonical morphisms, and a complete set of equations relating them, in the sense that the guardedness laws for all derived guarded morphisms follow from them. For example, by applying $(\text{–})^{\sharp}$ to the composition

$$A \# (B+C) \xrightarrow{\epsilon_{A,B+C}} (A+(B+C))\#0 \cong ((A+B)+C))\#0$$

we obtain a morphism $\upsilon_{A,B,C}\colon A\#(B+C)\to(A+B)\#C$, which represents weakening of the guardedness guarantee: in $A\#(B+C)$ the guarded part is $B+C$, while in $(A+B)\#C$ the guarded part is only C. It should not make a difference though if starting from $A\#(B+(C+D))$ we apply υ twice or rearrange $B+(C+D)$ by associativity and subsequently apply υ only once – the results must be canonically isomorphic, which is indeed provable. Similarly, we will introduce further morphisms like υ and derive laws, relating them. We then prove that the resulting axiomatization enjoys a coherence property (Theorem 25) in the style of Mac Lane's coherence theorem for (symmetric) monoidal categories [26],

▶ **Definition 24** (Guarded Parametrized Monad). *A* guarded parametrized monad *on a symmetric monoidal category* $(\mathbf{V},\otimes,I)$ *consists of a bifunctor* $\#\colon\mathbf{V}\times\mathbf{V}\to\mathbf{V}$ *and natural transformations*

$$\begin{aligned}
&\eta\colon\; A\to A\#I, \\
&\upsilon\colon\; A\#(B\otimes C)\to(A\otimes B)\#C, && \xi\colon\; (A\#B)\#C\to A\#(B\otimes C),\\
&\chi\colon\; A\#B\otimes C\#D\to(A\otimes C)\#(B\otimes D), && \zeta\colon\; A\#(B\#C)\to A\#(B\otimes C).
\end{aligned}$$

such that the following diagrams commute, where $\cong$ *refers to obvious canonical isomorphisms*

$$\begin{tikzcd}
A\#(I\otimes C) \arrow[rr, "\upsilon"] \arrow[dr, phantom, "\cong"] & & (A\otimes I)\#C \arrow[dl, phantom, "\cong"] \\
& A\#C &
\end{tikzcd}
\qquad
\begin{tikzcd}
A\#(B\otimes(C\otimes D)) \arrow[r, phantom, "\cong"] \arrow[d, "\upsilon"'] & A\#((B\otimes C)\otimes D) \arrow[dd, "\upsilon"] \\
(A\otimes B)\#(C\otimes D) \arrow[d, "\upsilon"'] & \\
((A\otimes B)\otimes C)\#D \arrow[r, phantom, "\cong"] & (A\otimes(B\otimes C))\#D
\end{tikzcd}$$

$$\begin{tikzcd}
A\otimes B \arrow[r, "\eta\otimes\eta"] \arrow[d, "\eta"'] & A\#I\otimes B\#I \arrow[d, "\chi"] \\
(A\otimes B)\#I \arrow[r, phantom, "\cong"] & (A\otimes B)\#(I\otimes I)
\end{tikzcd}
\qquad
\begin{tikzcd}
A\#B\otimes C\#D \arrow[r, phantom, "\cong"] \arrow[d, "\chi"'] & C\#D\otimes A\#B \arrow[d, "\chi"] \\
(A\otimes C)\#(B\otimes D) \arrow[r, phantom, "\cong"] & (C\otimes A)\#(D\otimes B)
\end{tikzcd}$$

$$\begin{tikzcd}
A\#B\otimes(C\#D\otimes E\#F) \arrow[r, phantom, "\cong"] \arrow[d, "\mathsf{id}\#\chi"'] & (A\#B\otimes C\#D)\otimes E\#F \arrow[d, "\chi\#\mathsf{id}"] \\
A\#B\otimes(C\otimes E)\#(D\otimes F) \arrow[d, "\chi"'] & (A\otimes C)\#(B\otimes D)\otimes E\#F \arrow[d, "\chi"] \\
(A\otimes(C\otimes E))\#(B\otimes(D\otimes F)) \arrow[r, phantom, "\cong"] & ((A\otimes C)\otimes E)\#((B\otimes D)\otimes F)
\end{tikzcd}$$

$$\begin{tikzcd}
(A\#I)\#B \arrow[rr, "\xi"] & & A\#(I\otimes B) \arrow[dl, phantom, "\cong"] \\
& A\#B \arrow[ul, "\eta\#\mathsf{id}"] &
\end{tikzcd}
\qquad
\begin{tikzcd}
A\#(B\#I) \arrow[rr, "\zeta"] & & A\#(B\otimes I) \arrow[dl, phantom, "\cong"] \\
& A\#B \arrow[ul, "\mathsf{id}\#\eta"] &
\end{tikzcd}$$

$$\begin{tikzcd}
& A\#((B\#C)\#(D\#E)) \arrow[dl, "\zeta"'] \arrow[dr, "\mathsf{id}\#\zeta"] & \\
A\#((B\#C)\otimes(D\#E)) \arrow[d, "\mathsf{id}\#\chi"'] & & A\#((B\#C)\#(D\otimes E)) \arrow[d, "\mathsf{id}\#\xi"] \\
A\#((B\otimes D)\#(C\otimes E)) \arrow[d, "\zeta"'] & & A\#(B\#(C\otimes(D\otimes E))) \arrow[d, "\zeta"] \\
A\#((B\otimes D)\otimes(C\otimes E)) \arrow[rr, phantom, "\cong"] & & A\#(B\otimes(C\otimes(D\otimes E)))
\end{tikzcd}$$

$$\begin{tikzcd}
 & (A\sharp(B\sharp C))\sharp(D\sharp E) \arrow[dl, "\xi"'] \arrow[dr, "\zeta\sharp\mathsf{id}"] & \\
A\sharp((B\sharp C)\otimes(D\sharp E)) \arrow[d, "\mathsf{id}\sharp\chi"'] & & (A\sharp(B\otimes C))\sharp(D\sharp E) \arrow[d, "\mathsf{id}\sharp\xi"] \\
A\sharp((B\sharp C)\otimes(D\sharp E)) \arrow[d, "\zeta"'] & & (A\sharp(B\otimes C))\sharp(D\otimes E) \arrow[d, "\xi"] \\
A\sharp((B\otimes D)\otimes(C\otimes E)) & \cong & A\sharp((B\otimes C)\otimes(D\otimes E))
\end{tikzcd}$$

$$\begin{tikzcd}
((A\sharp B)\sharp C)\otimes((D\sharp E)\sharp F) \arrow[r, "\chi"] \arrow[d, "\xi\sharp\xi"'] & (A\sharp B\otimes D\sharp E)\sharp(C\otimes F) \arrow[d, "\chi\sharp\mathsf{id}"] \\
A\sharp(B\otimes C)\otimes D\sharp(E\otimes F) \arrow[d, "\chi"'] & ((A\otimes D)\sharp(B\otimes E))\sharp(C\otimes F) \arrow[d, "\xi"] \\
(A\otimes D)\sharp((B\otimes C)\otimes(E\otimes F)) \arrow[r, phantom, "\cong"] & (A\otimes D)\sharp((B\otimes E)\otimes(C\otimes F))
\end{tikzcd}$$

$$\begin{tikzcd}
(A\sharp(B\sharp C))\otimes(D\sharp(E\sharp F)) \arrow[r, "\chi"] \arrow[d, "\zeta\sharp\zeta"'] & (A\otimes D)\sharp(B\sharp C\otimes E\otimes F) \arrow[d, "\mathsf{id}\sharp\chi"] \\
A\sharp(B\otimes C)\otimes D\sharp(E\otimes F) \arrow[d, "\chi"'] & (A\otimes D)\sharp((B\otimes E)\sharp(C\otimes F)) \arrow[d, "\zeta"] \\
(A\otimes D)\sharp((B\otimes C)\otimes(E\otimes F)) \arrow[r, phantom, "\cong"] & (A\otimes D)\sharp((B\otimes E)\otimes(C\otimes F))
\end{tikzcd}$$

$$\begin{tikzcd}
(A\sharp B)\sharp(C\sharp D\otimes E\sharp F) \arrow[r, "\upsilon"] \arrow[d, "\mathsf{id}\sharp\chi"'] & (A\sharp B\otimes C\sharp D)\sharp(E\sharp F) \arrow[d, "\chi\sharp\mathsf{id}"] \\
(A\sharp B)\sharp((C\otimes E)\sharp(D\otimes F)) \arrow[d, "\zeta"'] & ((A\otimes C)\sharp(B\otimes D))\sharp(E\sharp F) \arrow[d, "\zeta"] \\
(A\sharp B)\sharp((C\otimes E)\otimes(D\otimes F)) \arrow[d, "\xi"'] & ((A\otimes C)\sharp(B\otimes D))\otimes(E\sharp F) \arrow[d, "\xi"] \\
A\sharp(B\otimes((C\otimes D)\otimes(E\otimes F))) \arrow[d, phantom, sloped, "\cong"] & (A\otimes C)\sharp((B\otimes D)\otimes(E\otimes F)) \arrow[d, equal] \\
A\sharp(C\otimes((B\otimes D)\otimes(E\otimes F))) \arrow[r, "\upsilon"] & (A\otimes C)\sharp((B\otimes D)\otimes(E\otimes F))
\end{tikzcd}$$

$$\begin{tikzcd}
A\sharp B\otimes C\sharp(D\otimes E) \arrow[r, "\mathsf{id}\otimes\upsilon"] \arrow[d, "\chi"'] & A\sharp B\otimes(C\otimes D)\sharp E \arrow[d, "\chi"] \\
(A\otimes C)\sharp(B\otimes(D\otimes E)) \arrow[d, phantom, sloped, "\cong"] & (A\otimes(C\otimes D))\sharp(B\otimes E) \arrow[d, phantom, sloped, "\cong"] \\
(A\otimes C)\sharp(D\otimes(B\otimes E)) \arrow[r, "\upsilon"] & ((A\otimes C)\otimes D))\sharp(B\otimes E)
\end{tikzcd}$$

$$\begin{tikzcd}
A\sharp(B\sharp(C\otimes D)) \arrow[r, "\mathsf{id}\sharp\upsilon"] \arrow[d, "\zeta"'] & A\sharp((B\otimes C)\sharp D) \arrow[d, "\zeta"] \\
A\sharp(B\otimes(C\otimes D)) \arrow[r, phantom, "\cong"] & A\sharp((B\otimes C)\otimes D)
\end{tikzcd}$$

$$\begin{tikzcd}
(A\sharp(B\otimes C))\sharp D \arrow[r, "\upsilon\sharp\mathsf{id}"] \arrow[d, "\xi"'] & ((A\otimes B)\sharp C)\sharp D \arrow[d, "\xi"] \\
A\sharp((B\otimes C)\otimes D) \arrow[d, phantom, sloped, "\cong"] & (A\otimes B)\sharp(C\otimes D) \arrow[d, equal] \\
A\sharp(B\otimes(C\otimes D)) \arrow[r, "\upsilon"] & (A\otimes B)\sharp(C\otimes D)
\end{tikzcd}$$

The value of the presented axiomatization is attested by the following

▶ **Theorem 25** (Coherence). *Let $\mathcal{E}_1$, $\mathcal{E}_2$ and $\mathcal{E}_2$ be expressions, built from $\otimes$, $\#$ and I over a set of letters, in such a way that $\mathcal{E}_1$ and $\mathcal{E}_2 \# \mathcal{E}_2'$ contain each letter at most once and neither $\mathcal{E}_2$ nor $\mathcal{E}_2'$ contain $\#$. Given two expressions f and g built from $\otimes$ and $\#$ over identities, associators, unitors, braidings, η, υ, ξ, ζ, χ, in such a way that formally $f, g\colon \mathcal{E}_1 \to \mathcal{E}_2 \# \mathcal{E}_2'$, then $f = g$ follows from the axioms of guarded parametrized monads.*

Proof Sketch. Let us refer to the described expressions $\mathcal{E}_1$ as *object expressions* and to f formed as described as *morphism expressions*. For two morphism expressions $f, g\colon E \to E'$, let $f \equiv g$ denote '*$f = g$ follows from the axioms of guarded parametrized monads*'. An object expression is *normal* if it is of the form $\mathcal{E} \# \mathcal{E}'$ and $\mathcal{E}$ and $\mathcal{E}'$ do not contain $\#$. For an object expression $\mathcal{E}$, we define object expressions $\mathsf{nf}_1(\mathcal{E})$ and $\mathsf{nf}_2(\mathcal{E})$ recursively with the clauses:

- $\mathsf{nf}_1(\mathcal{E}) = \mathcal{E}$, $\mathsf{nf}_2(\mathcal{E}) = I$ if $\mathcal{E} = I$ or $\mathcal{E}$ is a letter;
- $\mathsf{nf}_1(\mathcal{E} \otimes \mathcal{E}') = \mathsf{nf}_1(\mathcal{E}) \otimes \mathsf{nf}_1(\mathcal{E}')$, $\mathsf{nf}_2(\mathcal{E} \otimes \mathcal{E}') = \mathsf{nf}_2(\mathcal{E}) \otimes \mathsf{nf}_2(\mathcal{E}')$;
- $\mathsf{nf}_2(\mathcal{E} \# \mathcal{E}') = \mathsf{nf}_1(\mathcal{E})$, $\mathsf{nf}_2(\mathcal{E} \otimes \mathcal{E}') = \mathsf{nf}_1(\mathcal{E}') \otimes (\mathsf{nf}_2(\mathcal{E}) \otimes \mathsf{nf}_2(\mathcal{E}'))$.

Let $\mathsf{nf}(\mathcal{E}) = \mathsf{nf}_1(\mathcal{E}) \# \mathsf{nf}_2(\mathcal{E})$, so $\mathsf{nf}(\mathcal{E})$ is normal. For any object expression $\mathcal{E}$ we also define a *normalization morphism expression* $\mathsf{nm}(\mathcal{E})\colon \mathcal{E} \to \mathsf{nf}(\mathcal{E})$, by induction as follows:

- $\mathsf{nm}(\mathcal{E}) = \eta$ if $\mathcal{E} = I$ or $\mathcal{E}$ is a letter;
- $\mathsf{nm}(\mathcal{E} \otimes \mathcal{E}') = \chi \circ (\mathsf{nm}(\mathcal{E}) \otimes \mathsf{nm}(\mathcal{E}'))$;
- $\mathsf{nm}(\mathcal{E} \# \mathcal{E}') = \xi \circ \zeta \circ (\mathsf{nm}(\mathcal{E}) \# \mathsf{nm}(\mathcal{E}'))$.

The goal will follow from the following subgoals.

1. If a morphism expression $f\colon \mathcal{E} \to \mathcal{E}'$ does not contain υ then $\mathsf{nm}(\mathcal{E}') \circ f \equiv \mathsf{nm}(\mathcal{E}) \circ g$ for a suitable isomorphism g, constructed from $\otimes$, $\#$ and the coherent isomorphisms of the monoidal structure.
2. If a morphism expression $f\colon \mathcal{E} \to \mathcal{E}'$ does not contain η, ξ, ζ, χ then there exists $g\colon \mathsf{nf}(\mathcal{E}) \to \mathsf{nf}(\mathcal{E}')$ that also does not contain η, ξ, ζ, χ and such that $\mathsf{nm}(\mathcal{E}') \circ f \equiv g \circ \mathsf{nm}(\mathcal{E})$.
3. If $f, g\colon \mathcal{E} \to \mathcal{E}'$, $\mathcal{E}$ is a normal object expression and g and f do not contains η, ξ, ζ and χ then $f \equiv g$.

Indeed, given $f, g\colon \mathcal{E} \to \mathcal{E}'$ with normal $\mathcal{E}'$, to prove $f \equiv g$, it suffices to prove that f is equal to $\mathcal{E} \xrightarrow{\mathsf{nm}(\mathcal{E})} \mathsf{nf}(\mathcal{E}) \xrightarrow{f'} \mathcal{E}'$ for a suitable f' – the analogous statement would be true for g, and we would be done by **3**. Let us represent f as a composition $f_n \circ \ldots \circ f_1$ where every f_i with even i contains precisely one occurrence of υ and every f_i with odd i contains no occurrences of υ. We obtain

$$\begin{tikzcd}
\mathcal{E} \arrow[r, "f_1"] \arrow[d, "\mathsf{nm}(\mathcal{E})"'] & \mathcal{E}_1 \arrow[r, "f_2"] \arrow[d, "\mathsf{nm}(\mathcal{E}_1)"'] & \mathcal{E}_2 \arrow[d, "\mathsf{nm}(\mathcal{E}_2)"'] & \cdots & \mathcal{E}_n \arrow[d, "\mathsf{nf}(\mathcal{E}_n)"] \\
\mathsf{nf}(\mathcal{E}) \arrow[r, phantom, "\cong"] & \mathsf{nf}(\mathcal{E}_1) \arrow[r] & \mathsf{nf}(\mathcal{E}_2) & \cdots & \mathsf{nf}(\mathcal{E}_n)
\end{tikzcd}$$

where every odd diagram commutes by **1** and every even diagram commutes by **2**. Note that $\mathsf{nf}(\mathcal{E}_n)$ is an isomorphism, since $\mathcal{E}_n = \mathcal{E}'$ is normal, and therefore we obtain the desired presentation for f. Let us show the subgoals.

1. W.l.o.g. assume that f contains precisely one letter from the list η, ξ, ζ, χ, ρ, λ, α, γ where ρ, λ are the right and left unitors of $\otimes$, α is the associator and γ is braiding. The general case will follow by induction. Furthermore, by structural induction over $\mathcal{E}$, we restrict to the situation that $f \in \{\eta, \xi, \zeta, \chi, \rho, \lambda, \alpha, \gamma\}$. The rest follows by case distinction.
2. Again, w.l.o.g. f contains precisely one occurrence of υ. The reduction to $f = \upsilon$, runs by structural induction over $\mathcal{E}$ and in contrast to the previous clause relies on the diagrams, combining υ with ξ, ζ and χ correspondingly.

3. Observe that f is a composition of morphisms of the form $\beta \circ h \circ \alpha$ where α and β are coherent isomorphisms and h contains one occurrence of υ. By induction, using the properties of υ we can reduce to the case $f = \alpha \circ h \circ \beta$ and analogously, we can reduce to $g = \alpha' \circ h' \circ \beta'$. It is then easy to see that h and h' are equal up to a coherent isomorphism, and the desired equality $f \equiv g$ follows by coherence for symmetric monoidal categories. ◀

The present version is sufficient for our purposes. It is an open question if a stronger version with general $f, g\colon \mathcal{E}_1 \to \mathcal{E}_2$ can be proven. In the sequel, we will only deal with guarded parametrized monads over $(\mathbf{V}, +, 0)$. Recall that a parametrized monad [37] is a bifunctor $T\colon \mathbf{V} \times \mathbf{V} \to \mathbf{V}$, such that each $T(\text{–}, X)$ is a monad and each $T(\text{–}, f)$ is a monad morphism.

▶ **Proposition 26.** *Every guarded parametrized monad is a parametrized monad with $A \xrightarrow{\eta} A \# 0 \xrightarrow{\mathsf{id}\#!} A \# B$ as unit and $(A \# B) \# B \xrightarrow{\xi} A \# (B + B) \xrightarrow{\mathsf{id}\#\nabla} A \# B$ as multiplication.*

Proof Sketch. The proof essentially follows from coherence. For example, consider the associativity monad law $\mu_{A,B} \circ \mu_{A\#B,B} = \mu_{A,B} \circ (\mu_{A,B} \# B)$. On the one hand (by naturality),

$$\begin{aligned}
\mu_{A,B} \circ \mu_{A\#B,B} &= (A \# \nabla) \circ \xi_{A,B,B} \circ ((A \# B) \# \nabla) \circ \xi_{A\#B,B,B} \\
&= (A \# \nabla) \circ (A \# (B + \nabla)) \circ \xi_{A,B,B+B} \circ \xi_{A\#B,B,B} \\
&= (A \# (\nabla \circ (B + \nabla))) \circ \xi_{A,B,B+B} \circ \xi_{A\#B,B,B}.
\end{aligned}$$

On the other hand,

$$\begin{aligned}
\mu_{A,B} \circ (\mu_{A,B} \# B) &= (A \# \nabla) \circ \xi_{A,B,B} \circ ((A \# \nabla) \circ \xi_{A,B,B} \# B) \\
&= (A \# \nabla) \circ (A \# (\nabla + B)) \circ \xi_{A,B+B,B} \circ (\xi_{A,B,B} \# B) \\
&= (A \# (\nabla \circ (\nabla + B))) \circ \xi_{A,B+B,B} \circ (\xi_{A,B,B} \# B).
\end{aligned}$$

By coherence, $\xi_{A,B,B+B} \circ \xi_{A\#B,B,B}$ and $\xi_{A,B+B,B} \circ (\xi_{A,B,B} \# B)$ are equal up to the canonical isomorphism $(B + B) + B \cong B + (B + B)$ and hence the expressions we computed above are equal as well. ◀

▶ **Theorem 27.** *Given co-Cartesian $\mathbf{V}$ and an identity-on-object functor $J\colon \mathbf{V} \to \mathbf{C}$ strictly preserving coproducts, $\mathbf{C}$ is guarded and $\mathbf{C}_\bullet$ is representable iff $\mathbf{C} \cong \mathbf{V}_{-\#0}$ for a guarded parametrized monad $(\#, \eta, \upsilon, \chi, \xi, \zeta)$, the compositions $\upsilon_{X,Y,0} \circ (\mathsf{id} \# \mathsf{inl})$ are all monic and $f\colon X \to Y \rangle Z$ iff f factors through $Y \# (Z + 0) \xrightarrow{\upsilon} (Y + Z) \# 0$.*

Vacuous guardedness is clearly representable and by Theorem 27 corresponds to those guarded parametrized monads $\#$, which do not depend on the parameter, i.e. to monads.

▶ **Example 28.** Let us revisit Example 12. Let $X \# Y = T(X + HT_H(X + Y))$, and note that $\text{–} \# 0$ is isomorphic to T_H. Assuming existence of some morphism $p\colon 1 \to H1$, for every X, we obtain the final map $\hat{p}\colon 1 \to T_H X$, induced by the coalgebra map $1 \xrightarrow{\eta \circ \mathsf{inr} \circ p} T(X + H1)$. Now, $T(\mathsf{inl} + \mathsf{id})$ is a section, since $T[\mathsf{id} + H\hat{p} \circ p \circ !, \mathsf{inr}] \circ T(\mathsf{inl} + \mathsf{id})$ is the identity. By Theorem 23, $\#$ is a guarded parametrized monad.

▶ **Example 29.** Let us revisit Example 14. Let $X \# Y = \mathbb{R}_{\geqslant 0} \times X + \mathbb{R}_{>0} \times Y + \bar{\mathbb{R}}_{\geqslant 0}$. Then $X \# 0 \cong \mathbb{R}_{\geqslant 0} \times X + \bar{\mathbb{R}}_{\geqslant 0}$ and there is an obvious injection $\epsilon_{X,Y}$ from $X \# Y$ to $(X + Y) \# 0$. By definition, every guarded $f\colon X \to Y \# Z$ uniquely factors through $\epsilon_{Y,Z}$, and hence $\#$ is a guarded parametrized monad.

▶ **Definition 30** (Strong Guarded Parametrized Monad). *A guarded parametrized monad* $(\#, \eta, \upsilon, \chi, \xi, \zeta)$ *is strong, if* $\#$ *is strong as a monad in the first argument and as a functor in the second argument, and the diagram*

$$\begin{tikzcd}
X \times (Y \# Z) \arrow[r, "\mathsf{id} \times \epsilon"] \arrow[d, "\tau"'] & X \times (Y + Z) \# 0 \arrow[r, "\tau"] & (X \times (Y + Z)) \# 0 \arrow[r, "\mathsf{dist} \# 0"] & (X \times Y + X \times Z) \# 0 \arrow[d, "(\mathsf{id} + \mathsf{snd}) \# 0"] \\
(X \times Y) \# Z \arrow[rrr, "\epsilon"] & & & (X \times Y + Z) \# 0
\end{tikzcd}$$

commutes, where $\epsilon_{X,Y} = \upsilon_{X,Y,0} \circ (\mathsf{id} \# \mathsf{inl})$ *and* τ *is the monadic strength of* $\#$.

▶ Remark 31. Strength is commonly referred to as a "technical condition". This is justified by the fact that in self-enriched categories strength is equivalent to enrichment of the corresponding functor or a monad [21], and in foundational categories, like **Set**, every functor and every natural transformation are canonically enriched w.r.t. Cartesian closeness as the self-enrichment structure. Then canonical strength $\rho_{X,Y} \colon X \times FY \to F(X \times Y)$ for a functor F is defined by the expression $\rho_{X,Y} = \lambda(x, z).\, F(\lambda y.\, (x, y))(z)$. We conjecture that strengths involved in Definition 30 are technical in the same sense, in particular the requested commutative diagram is entailed by enrichment of ϵ.

Finally, let us establish the analogue of Theorem 27 for Freyd categories.

▶ **Theorem 32.** *A Freyd category* $(\mathbf{V}, \mathbf{C}, J(\text{–}), \oslash)$ *is guarded and* $\mathbf{C}_\bullet$ *is representable iff* $\mathbf{C} \cong \mathbf{V}_{-\#0}$ *for a strong guarded parametrized monad* $(\#, \eta, \upsilon, \chi, \xi, \zeta)$*, the compositions* $\upsilon_{X,Y,0} \circ (\mathsf{id} \# \mathsf{inl})$ *are all monic and* $f \colon X \to Y \rangle Z$ *iff* f *factors through* $Y \# (Z + 0) \xrightarrow{\upsilon} (Y + Z) \# 0$.

For a strong guarded parametrized monad $\#$, let $\tilde{\tau}$ be the composition

$$\begin{aligned}
X \times (Y \# Z) &\xrightarrow{\Delta \times \mathsf{id}} (X \times X) \times (Y \# Z) \cong X \times (X \times (Y \# Z)) \\
&\xrightarrow{\mathsf{id} \times \rho} X \times (Y \# (X \times Z)) \xrightarrow{\tau} (X \times Y) \# (X \times Z)
\end{aligned}$$

where τ is the monadic strength of $\#$ and ρ is the functorial strength of $\#$. It is easy to check that τ and ρ are derivable from $\tilde{\tau}$, and in the sequel, we will include it as the last element in a tuple $(\#, \eta, \upsilon, \chi, \xi, \zeta, \tilde{\tau})$, defining a strong guarded parametrized monad.

Finally, we can interpret the guarded version of FGCBV over a strong guarded parametrized monad $(\#, \eta, \upsilon, \chi, \xi, \zeta, \tilde{\tau})$ on $\mathbf{V}$. Let sorts and function symbols from Σ_v be interpreted as usual and let $[\![f]\!] \in \mathbf{V}([\![A]\!], [\![B]\!] \# [\![C]\!])$ for $f \colon A \to B \rangle C \in \Sigma_c$. Then $[\![\Gamma \vdash_{\mathsf{v}} v \colon B]\!] \in \mathbf{V}([\![A]\!], [\![B]\!])$ and $[\![\Gamma \vdash_{\mathsf{c}} p \colon B \rangle C]\!] \in \mathbf{V}([\![A]\!], [\![B]\!] \# [\![C]\!])$ are defined with the following clauses:

- $[\![x_1 \colon A_1, \dots, x_n \colon A_n \vdash_{\mathsf{v}} x_i \colon A_i]\!] = \mathsf{inj}_i$;
- $[\![\Gamma \vdash_{\mathsf{v}} f(v) \colon B]\!] = [\![f]\!] \circ [\![\Gamma \vdash_{\mathsf{v}} v \colon A]\!]$;
- $[\![\Gamma \vdash_{\mathsf{c}} f(v) \colon B]\!] = [\![f]\!] \circ [\![\Gamma \vdash_{\mathsf{v}} v \colon A]\!]$;
- $[\![\Gamma \vdash_{\mathsf{c}} \mathsf{return}\, v \colon A \rangle B]\!] = \eta_{[\![A]\!],[\![B]\!]} \circ [\![\Gamma \vdash_{\mathsf{v}} v \colon A]\!]$;
- $[\![\Gamma \vdash_{\mathsf{c}} \mathsf{docase}\, p\, \mathsf{of}\ \mathsf{inl}\, x \mapsto q;\ \mathsf{inr}\, y \mapsto r \colon C \rangle D]\!] = (\nabla \# \nabla) \circ \xi_{[\![C]\!],[\![D]\!],[\![C]\!]+[\![D]\!]}$
  $\circ\, \zeta_{[\![C]\!] \# [\![D]\!],[\![C]\!],[\![D]\!]} \circ ([\![\Gamma, x \colon A \vdash_{\mathsf{c}} q \colon C \rangle D]\!] \# [\![\Gamma, y \colon B \vdash_{\mathsf{c}} r \colon C \rangle D]\!])$
  $\circ\, \tilde{\tau}_{[\![\Gamma]\!],[\![A]\!],[\![B]\!]} \circ \langle \mathsf{id}, [\![\Gamma \vdash_{\mathsf{c}} p \colon A \rangle B]\!] \rangle$;
- $[\![\Gamma \vdash_{\mathsf{v}} \mathsf{init}\, v \colon A]\!] = \,!$;
- $[\![\Gamma \vdash_{\mathsf{v}} \mathsf{inl}\, v \colon A + B]\!] = \mathsf{inl} \circ [\![\Gamma \vdash_{\mathsf{v}} v \colon A]\!]$;
- $[\![\Gamma \vdash_{\mathsf{v}} \mathsf{inr}\, v \colon A + B]\!] = \mathsf{inr} \circ [\![\Gamma \vdash_{\mathsf{v}} v \colon B]\!]$;
- $[\![\Gamma \vdash_{\mathsf{c}} \mathsf{case}\, v\, \mathsf{of}\ \mathsf{inl}\, x \mapsto p;\ \mathsf{inr}\, y \mapsto q \colon C \rangle D]\!] = (\nabla \# \nabla) \circ \chi_{[\![C]\!],[\![D]\!],[\![C]\!],[\![D]\!]}$
  $\circ\, ([\![\Gamma, x \colon A \vdash_{\mathsf{c}} p \colon C \rangle D]\!] + [\![\Gamma, y \colon B \vdash_{\mathsf{c}} q \colon C \rangle D]\!]) \circ \mathsf{dist} \circ \langle \mathsf{id}, [\![\Gamma \vdash_{\mathsf{v}} v \colon A + B]\!] \rangle$.
- $[\![\Gamma \vdash_{\mathsf{v}} \langle v, w \rangle \colon A \times B]\!] = \langle [\![\Gamma \vdash_{\mathsf{v}} v \colon A]\!], [\![\Gamma \vdash_{\mathsf{v}} w \colon B]\!] \rangle$.

Note that what allows us to sidestep the monicity condition of the representability criterion (Theorem 27) is that we gave up on the assumption that the space of guarded morphisms $X \to Y \# Z$ injectively embeds to the space of all morphisms $X \to (Y + Z) \# 0$, in particular, the entire notion of guardedness predicate is eliminated.

8 Conclusions and Further Work

We investigated a combination of FGCBV and guardedness by taking inspiration from the previous work on relating Freyd categories with strong monads via a natural requirement of representability of certain presheaves. An abstract notion of guardedness naturally fits the FGCBV paradigm and gives rise to more general formats of presheaves, which must be representable e.g. in order to be able to interpret higher-order (guarded) functions. In our case, the representability requirement gave rise to a novel categorical structure, we dub (strong) guarded parametrized monad, which encapsulate computational effects under consideration, simultaneously with guardedness guarantees.

We regard our present results as a prerequisite step for implementing guarded programs in existing higher-order languages, such as Haskell, and in proof assistants with strict support of the propositions-as-types discipline, such as Coq and Agda, where unproductive recursive definitions cannot be implemented directly, and thus the importance of guarded iteration is particularly high. It would be interesting to further refine guarded parametrized monads so as to include further quantitative information on how productive a computation is, or how unproductive it is, so that this relative unproductivity could possibly be cancelled out by composition with something very productive. Another strand for future work comes from the observation that guarded iteration is a formal dual of guarded recursion [18]. A good deal of the present theory can be easily dualized, which will presumably lead to guarded parametrized comonads and comonadic recursion – we are planning to investigate these structures from the perspective of comonadic notion of computation [38]. In terms of syntax, a natural extension of fine-gain call-by-value is call-by-push-value [22]. We expect it to be a natural environment for analysing the above mentioned aspects in the style of the presented approach.

References

1 Jiří Adámek, Horst Herrlich, and George Strecker. *Abstract and concrete categories*. John Wiley & Sons Inc., New York, 1990.

2 Jiří Adámek, Stefan Milius, and Jiří Velebil. On rational monads and free iterative theories. In *Proc. Category Theory and Computer Science (CTCS'02)*, volume 69 of *Electron. Notes Theor. Comput. Sci.*, pages 23–46, 2002.

3 Jiří Adámek, Stefan Milius, and Jiří Velebil. Equational properties of iterative monads. *Inf. Comput.*, 208(12):1306–1348, 2010.

4 Thorsten Altenkirch, Nils Danielsson, and Nicolai Kraus. Partiality, revisited - the partiality monad as a quotient inductive-inductive type. In Javier Esparza and Andrzej Murawski, editors, *Foundations of Software Science and Computation Structures, FOSSACS 2017*, volume 10203 of *LNCS*, pages 534–549, 2017.

5 Steve Awodey. *Category Theory*. Oxford University Press, Inc., New York, NY, USA, 2nd edition, 2010.

6 J. Bergstra, A. Ponse, and Scott Smolka, editors. *Handbook of Process Algebra*. Elsevier, 2001.

7 Stephen Bloom and Zoltán Ésik. *Iteration theories: the equational logic of iterative processes*. Springer, 1993.

8 Venanzio Capretta. General recursion via coinductive types. *Log. Meth. Comput. Sci.*, 1(2), 2005.

9 James Chapman, Tarmo Uustalu, and Niccolò Veltri. Quotienting the delay monad by weak bisimilarity. In *Theoretical Aspects of Computing, ICTAC 2015*, volume 9399 of *LNCS*, pages 110–125. Springer, 2015.

10 J. Robin B. Cockett. Introduction to distributive categories. *Mathematical Structures in Computer Science*, 3(3):277–307, 1993.

11 Martín H. Escardó and Cory M. Knapp. Partial Elements and Recursion via Dominances in Univalent Type Theory. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, volume 82 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

12 M.P. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps.* Distinguished Dissertations in Computer Science. Cambridge University Press, 2004.

13 Bram Geron and Paul Blain Levy. Iteration and labelled iteration. In *Proc. 32nd Conference on the Mathematical Foundations of Programming Semantics (MFPS 2016)*, volume 325 of *ENTCS*, pages 127–146. Elsevier, 2016.

14 Sergey Goncharov. Uniform Elgot Iteration in Foundations. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *LIPIcs*, pages 131:1–131:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

15 Sergey Goncharov, Renato Neves, and José Proença. Implementing hybrid semantics: From functional to imperative. In Volker Stolz Violet Ka I Pun, Adenilso da Silva Simão, editor, *17th International Colloquium on Theoretical Aspects of Computing (ICTAC 2020)*, 2020.

16 Sergey Goncharov, Christoph Rauch, and Lutz Schröder. Unguarded recursion on coinductive resumptions. In *Mathematical Foundations of Programming Semantics, MFPS 2015*, volume 319 of *ENTCS*, pages 183–198. Elsevier, 2015.

17 Sergey Goncharov, Christoph Rauch, and Lutz Schröder. A metalanguage for guarded iteration. *Theoretical Computer Science*, 880:111–137, 2021.

18 Sergey Goncharov and Lutz Schröder. Guarded traced categories. In Christel Baier and Ugo Dal Lago, editors, *Proc. 21th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2018)*, LNCS. Springer, 2018.

19 Sergey Goncharov, Lutz Schröder, Christoph Rauch, and Maciej Piróg. Unifying guarded and unguarded iteration. In Javier Esparza and Andrzej Murawski, editors, *Foundations of Software Science and Computation Structures, FoSSaCS 2017*, volume 10203 of *LNCS*, pages 517–533. Springer, 2017.

20 George Janelidze and Gregory M Kelly. A note on actions of a monoidal category. *Theory Appl. Categ*, 9(61-91):02, 2001.

21 Anders Kock. Strong functors and monoidal monads. *Archiv der Mathematik*, 23(1):113–120, 1972.

22 Paul Blain Levy. Call-by-push-value: A subsuming paradigm. In Jean-Yves Girard, editor, *TLCA*, volume 1581 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 1999.

23 Paul Blain Levy. *Call-By-Push-Value: A Functional/Imperative Synthesis (Semantics Structures in Computation, V. 2).* Kluwer Academic Publishers, USA, 2004.

24 Paul Blain Levy and Sergey Goncharov. Coinductive resumption monads: Guarded iterative and guarded elgot. In *Proc. 8rd international conference on Algebra and coalgebra in computer science (CALCO 2019)*, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

25 Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *Inf. & Comp*, 185:2003, 2002.

26 Saunders Mac Lane. *Categories for the Working Mathematician.* Springer, 1971.

27 R. Milner. *Communication and concurrency.* Prentice-Hall, 1989.

28 Eugenio Moggi. A modular approach to denotational semantics. In *Category Theory and Computer Science, CTCS 1991*, volume 530 of *LNCS*, pages 138–139. Springer, 1991.

29 Keiko Nakata and Tarmo Uustalu. A Hoare logic for the coinductive trace-based big-step semantics of While. *Log. Meth. Comput. Sci.*, 11(1), 2015.

30 Maciej Piróg and Jeremy Gibbons. The coinductive resumption monad. In *Mathematical Foundations of Programming Semantics, MFPS 2014*, volume 308 of *ENTCS*, pages 273–288, 2014.

31 Gordon Plotkin and John Power. Adequacy for algebraic effects. In *FoSSaCS'01*, volume 2030 of *LNCS*, pages 1–24, 2001.

32 A. J. Power and E. P. Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5):453–468, October 1997.

33 A. John Power and Hayo Thielecke. Closed Freyd- and kappa-categories. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming*, ICAL '99, pages 625–634, Berlin, Heidelberg, 1999. Springer-Verlag.

34 Dietmar Schumacher. Minimale und maximale tripelerzeugende und eine bemerkung zur tripelbarkeit. *Archiv der Mathematik*, 20(4):356–364, September 1969.

35 Alex Simpson and Gordon Plotkin. Complete axioms for categorical fixed-point operators. In *Logic in Computer Science, LICS 2000*, pages 30–41, 2000.

36 Sam Staton. Freyd categories are enriched Lawvere theories. *Electron. Notes Theor. Comput. Sci.*, 303:197–206, March 2014.

37 Tarmo Uustalu. Generalizing substitution. *ITA*, 37(4):315–336, 2003.

38 Tarmo Uustalu and Varmo Vene. Comonadic notions of computation. *Electron. Notes Theor. Comput. Sci.*, 203(5):263–284, 2008.