1st International Conference on Formal Structures for Computation and Deduction

FSCD 2016, June 22–226, 2016, Porto, Portugal

Edited by

Delia Kesner
Brigitte Pientka

LIPIcs – Vol. 52 – FSCD'16 www.dagstuhl.de/lipics

Editors

Delia Kesner	Brigitte Pientka
IRIF, Université Paris-Diderot	School of Computer Science
Paris, France	Montreal, Canada
`beditor@uni-city.edu`	`bpientka@cs.mcgill.ca`

ACM Classification 1998
F. Theory of Computation, F.1 Computation by abstract devices, F.2 Analysis of algorithms and problem complexity, F.3 Logics and meanings of programs, F.4 Mathematical logic and formal languages, D.1 Programming techniques, D.2.4 Software/Program Verification, D.3 Programming languages, I.2.3 Deduction and Theorem Proving, I.2.4 Knowledge Representation Formalisms and Methods, E.1 Data Structures

ISBN 978-3-95977-010-1

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-95977-010-1.

Publication date
June, 2016

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

Digital Object Identifier: 10.4230/LIPIcs.CVIT.2016.0

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Contents

Invited Talks

Regular Research Papers

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

System Descriptions

Preface

This volume contains the proceedings of the First International Conference on Formal Structures for Computation and Deduction (FSCD'16), which was held from June 22 to June 26, in Porto, Portugal.

FSCD (http://fscdconference.org/) covers all aspects of formal structures for computation and deduction, from theoretical foundations to applications. It builds on two communities: RTA (Rewriting Techniques and Applications) which goes back to 1983 and TLCA (Typed Lambda Calculi and Applications) which was founded in 1993. Since 2003, both conferences have co-located together for most meetings under the umbrella of the Federated Conference on Rewriting, Deduction and Programming. FSCD continues this tradition and broadens their scope to closely related areas in logics, proof theory and new emerging models of computation.

FSCD'16 has received 82 submissions (77 regular research papers and 5 system descriptions) with contributing authors from 27 countries. The program committee consisted of 34 members from 16 countries. Each submitted paper was reviewed by at least 3 members of the program committee, with the help of 103 external reviewers. The reviewing process was handled by the Easychair system over a whole period of 8 weeks, including a rebuttal phase. A total of 28 regular papers and 4 system descriptions were accepted for publication and are included in this proceeding. The FSCD program also featured four invited talks given by Amal Ahmed (Northeastern University, USA), Ichiro Hasuo (University of Tokyo, Japan), Gérard Huet (INRIA, France), and Tobias Nipkow (Technical University Munich, Germany) whose extended abstracts appear in the proceeding.

In addition to the main program, 11 workshops covering a wide range of topics co-located with FSCD and took place before, in parallel, and after FSCD:

- 6th Workshop on Classical Logic and Computation (CL&C)
- 2nd Workshop on Higher-Dimensional Rewriting and Applications (HDRA)
- 8th Workshop on Higher-Order Rewriting (HOR)
- 2nd Workshop on Homotopy Type Theory/Univalent Foundations (HoTT/UF)
- IFIP Working Group 1.6: Term Rewriting
- 8th Workshop on Intersection Types and Related Systems (ITRS)
- 18th Workshop on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP)
- 11th Logical and Semantic Frameworks with Applications (LSFA)
- 4th Workshop on Linearity (Linearity)
- 30th Workshop on Unification (UNIF)
- 3rd Workshop on Rewriting Techniques for Program Transformation and Evaluation (WPTE)

Many people helped make FSCD 2016 a success. We are very grateful to all the authors of submitted papers for considering FSCD for publishing their work. We also would like to thank all the members of the program committee, as well as all external reviewers, for carefully reviewing and evaluating papers. Their hard work helped to select a balanced and attractive program. Thanks also goes to Andrei Voronkov and his team for making available the easychair tool to manage the submission and reviewing process. On behalf of the program committee, we thank all invited speakers for enriching the conference with their talks. We also acknowledge the important contributions of the workshop organizers who were enthusiastic about co-locating with FSCD. The workshops helped shine the light on many areas related to computation and deduction and made the conference a vibrant event.

FSCD would not have been possible without the dedicated hard work of the Organizing Committee, headed by the Conference Chair Sandra Alves. The steering committee, lead by Luke Ong, provided valuable guidance ensuring FSCD will have a bright and long future ahead. Last, but not least, we thank all participants of the conference for creating a lively and exciting event.

This volume of FSCD 2016 is being published in the LIPIcs series under a Creative Common license, with free online access to all, and with authors retaining rights over their contributions. We thank in particular Marc Herbstritt form Schloss Dagstuhl, Leibniz Center for Informatics, for his helpful support during the production of this proceedings.

FSCD received support from many organizations. On behalf of all organizers, we gratefully acknowledge the support by Center for Research in Advanced Computing Systems (CRACS), Center for Mathematics from the University of Porto (CMUP), Artificial Intelligence and Computer Science Laboratory (LIACC), Two Sigma Investments, and University of Porto.

Delia Kesner and Brigitte Pientka
Co-chairs of FSCD'16

Steering Committee

Thorsten Altenkirch	University Nottingham	United Kingdom
Gilles Dowek	INRIA	France
Santiago Escobar	University Politecnica de Valencia	Spain
Maribel Fernandez	King's College London	United Kingdom
Masahito Hasegawa	University Kyoto	Japan
Hugo Herbelin	INRIA	France
Nao Hirokawa	JAIST	Japan
Luke Ong (Chair)	University Oxford	United Kingdom
Jens Palsberg	UCLA	United States
Kristoffer Rose	Two Sigma Investments	United States
Rene Thiemann	University Innsbruck	Austria
Pawel Urzyczyn	University Warsaw	Poland
Femke van Raamsdonk	VU University Amsterdam	Netherlands

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Program Committee

Andreas Abel	Gothenburg University	Sweden
Zena Ariola	University Oregon	USA
Patrick Baillot	CNRS & ENS Lyon	France
Andrej Bauer	University Ljubljana	Slovenia
Eduardo Bonelli	University Quilmes	Argentina
Patricia Bouyer	ENS Cachan	France
Ugo Dal Lago	University Bologna	Italy
Nachum Dershowitz	University Tel Aviv	Israel
M. Dezani-Ciancaglini	University Torino	Italy
Derek Dreyer	MPI-SWS	Germany
Santiago Figueira	University Buenos Aires	Argentina
Marcelo Fiore	University Cambridge	United Kingdom
Jürgen Giesl	University Aachen	Germany
Nao Hirokawa	JAIST	Japan
Martin Hofmann	LMU München	Germany
Delia Kesner (co-chair)	University Paris-Diderot	France
Naoki Kobayashi	University Tokyo	Japan
Dan Licata	Wesleyan University	USA
Chris Lynch	Clarkson University	USA
Narciso Martí-Oliet	University Complutense	Spain
Aart Middeldorp	University Innsbruck	Austria
Dale Miller	INRIA Saclay	France
César Muñoz	NASA	USA
Vivek Nigam	University Paraiba	Brazil
Brigitte Pientka (co-chair)	McGill University	Canada
Jakob Rehof	University Dortmund	Germany
Xavier Rival	ENS Paris	France
Peter Selinger	Dalhousie University	Canada
Paula Severi	University Leicester	United Kingdom
Jakob Grue Simonsen	University Copenhagen	Denmark
Matthieu Sozeau	INRIA Rocquencourt	France
Sophie Tison	University Lille	France
Femke van Raamsdonk	VU University Amsterdam	Netherlands
Nobuko Yoshida	Imperial College	United Kingdom

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Beniamino Accattoli
Klaus Aehlig
Luis Aguirre
Laura Alonso Alemany
Thorsten Altenkirch
Takahito Aoto
Martin Avanzini
Holger Bock Axelsen
Mauricio Ayala-Rincon
Demis Ballis
Franco Barbanera
Pablo Barenbaum
Emmanuel Beffara
Chantal Berline
Alexis Bernadet
Jan Bessai
Guillaume Bonfante
Adel Bouhoula
Flavien Breuvart
Andrew Cave
Arthur Charguéraud
Kaustuv Chaudhuri
Vincent Cheval
Pierre Clairambault
Mario Coppo
Raphaelle Crubillé
Łukasz Czajka
Laure Daviaud
Fer-Jan de Vries
Stéphanie Delaune
Romain Demangeon
Maria Emilia Descotte
Andrej Dudenhefner
Boris Duedder
Naohi Eguchi
David Frutos Escrig
Bertram Felgenhauer
Thomas Ferrère
Bernd Finkbeiner
Jonas Frey
Florian Frohn
Ilias Garnier
Thomas Genet
Silvia Ghilezan
Paola Giannini
Stéphane Graham-Lengrand
Charles Grellois
Masahito Hasegawa
Simon Huber
Benedetto Intrigila
Jan Johannsen
Philip Johnson-Freyd
Neil Jones
Jean-Pierre Jouannaud
Vïctor Lòpez Juan
Klaus Keimel
Jeroen Ketema
Hélène Kirchner
Yoonseok Ko
Vasileios Koutavas
Robbert Krebbers
Neelakantan Krishnaswami
Keiichirou Kusakari
Temur Kutsia
Edmund Soon Lee Lam
Julien Lange
Huisong Li
Tomer Libal
Carlos Lombardi
Ian Mackie
Andrea Masini
Samuel Mimram
Jean-Yves Moyen
Julian Nagele
Koji Nakazawa
Lasse Nielsen
Ulf Norell
Dominic Orchard
Yolanda Ortega-Mallén
Miguel Palomino
Luca Paolini
Pierre-Marie Pédrot
Elaine Pimentel
Detlef Plump
Carlos Gustavo Lopez Pombo
Christopher M. Poskitt
Matija Pretnar
Mike Rainey
Giselle Reis
Colin Riba
Camilo Rocha
Simona Ronchi Della Rocca
Vlad Rusu
Sam Staton
Gabriela Steren
Christian Sternagel
Thomas Streicher
Christine Tasson
René Thiemann
Laura Titolo
Bernardo Toninho
Vincent Van Oostrom
Daniele Varacca
Andrea Vezzosi
Mateu Vilaret
Andrés Ezequiel Viso
Johannes Waldmann
Sarah Winkler
Hans Zantema
Beta Ziliani

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Organising Committee

Sandra Alves (Conference Chair)	University of Porto	Portugal
Sabine Broda (Workshop Chair)	University Porto	Portugal
Jose Espìrito-Santo	University do Minho	Portugal
Mário Florido	University Porto	Portugal
Nelma Moreira	University Porto	Portugal
Luis Pinto	University do Minho	Portugal
Rogério Reis	University Porto	Portugal
Ana Paula Tomas	University Porto	Portugal
Pedro Vasconcelos	University Porto	Portugal

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Compositional Compiler Verification for a Multi-Language World*

Amal Ahmed

Northeastern University, Boston, USA
amal@ccs.neu.edu

Abstract

Verified compilers are typically proved correct under severe restrictions on what the compiler's output may be linked with, from no linking at all to linking only with code compiled from the same source language. Such assumptions contradict the reality of how we use these compilers since most software systems today are comprised of components written in different languages compiled by different compilers to a common target, as well as low-level libraries that may be handwritten in the target language.

The key challenge in verifying compilers for today's world of multi-language software is how to formally state a compiler correctness theorem that is compositional along two dimensions. First, the theorem must guarantee correct compilation of components while allowing compiled code to be composed (linked) with target-language components of arbitrary provenance, including those compiled from other languages. Second, the theorem must support verification of multi-pass compilers by composing correctness proofs for individual passes. In this talk, I will describe a methodology for verifying compositional compiler correctness for a higher-order typed language and discuss the challenges that lie ahead [1, 2]. I will argue that compositional compiler correctness is, in essence, a language interoperability problem: for viable solutions in the long term, high-level languages must be equipped with *principled foreign-function interfaces* that specify safe interoperability between high-level and low-level components, and between more precisely and less precisely typed code.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages, D.3.1 Formal Definitions and Theory, D.3.4 Processors

Keywords and phrases verified compilation, compositional compiler correctness, multi-language semantics, typed low-level languages

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.1

Category Invited Talk

References

1 Amal Ahmed. Verified compilers for a multi-language world. In *1st Summit on Advances in Programming Languages (SNAPL 2015)*, volume 32 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15–31, 2015.

2 James T. Perconti and Amal Ahmed. Verifying an open compiler using multi-language semantics. In *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014*, pages 128–148, April 2014.

* This talk describes work supported in part by the National Science Foundation (grants CCF-1453796, CCF-1422133, and CCF-1203008) and a Google Faculty Research Award.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 1; pp. 1:1–1:1
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Coalgebras and Higher-Order Computation: a GoI Approach*

Ichiro Hasuo

Department of Computer Science, University of Tokyo, Japan
ichiro@is.s.u-tokyo.ac.jp

Abstract

Girard's *geometry of interaction (GoI)* [3] can be seen – in one practical aspect of it – as a compositional compilation method from functional programs to sequential machines (see e.g. [8, 2]). There tokens move around and express interactions between (parts of) programs. Intrigued by the combination of abstract structures and concrete dynamics in GoI, our line of work [4, 5, 10, 11, 6, 9] has aimed at exploiting, in GoI, results from the theory of *coalgebra* – a categorical abstraction of state-based transition systems that has found its use principally in concurrency theory. Such reinforced connection between higher-order computation and state-based dynamics is made possible thanks to an elegant categorical axiomatization of GoI by Abramsky, Haghverdi and Scott [1], where *traced monoidal categories* [7] are identified to be the essential structure behind. In the talk I shall lay out these basic ideas, together with some of our results on: GoI semantics for a *quantum programming language* [4, 5]; and our "memoryful" extension of GoI [10, 11, 6, 9] with *algebraic effects* [12].

The talk is based on my joint work with my colleague Naohiko Hoshino (RIMS, Kyoto University) and my (former) students Koko Muroya (University of Birmingham) and Toshiki Kataoka (University of Tokyo), to whom I owe special thanks.

1998 ACM Subject Classification D.3.1 [Formal Definitions and Theory] Semantics, F.3.2 [Semantics of Programming Languages] Algebraic approaches to semantics

Keywords and phrases functional programming, geometry of interaction, categorical semantics, coalgebra

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.2

Category Invited Talk

References

1 S. Abramsky, E. Haghverdi and P. Scott. Geometry of interaction and linear combinatory algebras. *Math. Struct. in Comp. Sci.*, 12(5):625–665, 2002.

2 D.R. Ghica, A.I. Smith and S. Singh. Geometry of synthesis IV: compiling affine recursion into static hardware. In M.M.T. Chakravarty, Z. Hu and O. Danvy, editors, *ICFP*, pp. 221–233. ACM, 2011.

3 J.Y. Girard. Geometry of interaction I: Interpretation of system F. In R. Ferro *et al.*, editors, *Logic Colloquium 1988*, pp. 221–260. North-Holland, 1989.

4 I. Hasuo and N. Hoshino. Semantics of higher-order quantum computation via geometry of interaction. In *LICS*, pp. 237–246. IEEE Computer Society, 2011.

* Supported by Grants-in-Aid No. 24680001, 15K11984 & 15KT0012, JSPS; by the JSPS-INRIA Bilateral Joint Research Project "CRECOGI"; and by Aihara Innovative Mathematical Modelling Project, FIRST Program, JSPS/CSTP.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 2; pp. 2:1–2:2
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

5 I. Hasuo and N. Hoshino. Semantics of higher-order quantum computation via geometry of interaction, 2016. Extended version of [Hasuo & Hoshino, LICS 2011], to appear in *Ann. Pure & Appl. Logic*. Preprint available at arXiv.

6 N. Hoshino, K. Muroya and I. Hasuo. Memoryful geometry of interaction: from coalgebraic components to algebraic effects. In T.A. Henzinger and D. Miller, editors, *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*, page 52. ACM, 2014.

7 A. Joyal, R. Street and D. Verity. Traced monoidal categories. *Math. Proc. Cambridge Phil. Soc.*, 119(3):425–446, 1996.

8 I. Mackie. The geometry of interaction machine. In *POPL'95: Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 198–208. ACM, New York, NY, USA, 1995.

9 K. Muroya, N. Hoshino and I. Hasuo. Memoryful GoI with recursion. In *International Workshop on Syntax and Semantics of Low-Level Language (LOLA 2015)*. 2015.

10 K. Muroya, N. Hoshino and I. Hasuo. Memoryful geometry of interaction II: recursion and adequacy. In R. Bodik and R. 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*, pp. 748–760. ACM, 2016.

11 K. Muroya, T. Kataoka, I. Hasuo and N. Hoshino. Compiling effectful terms to transducers: Prototype implementation of memoryful geometry of interaction. In *International Workshop on Syntax and Semantics of Low-Level Language (LOLA 2014)*. 2014.

12 G.D. Plotkin and J. Power. Adequacy for algebraic effects. In F. Honsell and M. Miculan, editors, *FoSSaCS*, vol. 2030 of *Lecture Notes in Computer Science*, pp. 1–24. Springer, 2001.

Teaching Foundations of Computation and Deduction Through Literate Functional Programming and Type Theory Formalization

Gérard Huet

Inria Paris Laboratory, France

Abstract

We describe experiments in teaching fundamental informatics notions around mathematical structures for formal concepts, and effective algorithms to manipulate them. The major themes of lambda-calculus and type theory served as guides for the effective implementation of functional programming languages and higher-order proof assistants, appropriate for reflecting the theoretical material into effective tools to represent constructively the concepts and formally certify the proofs of their properties. Progressively, a literate programming and proving style replaced informal mathematics in the presentation of the material as executable course notes. The talk will evoke the various stages of (in)completion of the corresponding set of notes along the years, and tell how their elaboration proved to be essential to the discovery of fundamental results.

1998 ACM Subject Classification F.3.1 Specifying and Verifying Programs, F.4.1 Mathematical Logic and G.4 Mathematical Software

Keywords and phrases Foundations, Computation, Deduction, Programming, Proofs

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.3

Category Invited Talk

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 3; pp. 3:1–3:1
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Verified Analysis of Functional Data Structures[*]

Tobias Nipkow

Technische Universität München, Munich, Germany

Abstract

In recent work the author has analyzed a number of classical functional search tree and priority queue implementations with the help of the theorem prover Isabelle/HOL. The functional correctness proofs of AVL trees, red-black trees, 2-3 trees, 2-3-4 trees, 1-2 brother trees, AA trees and splay trees could be automated. The amortized logarithmic complexity of skew heaps, splay trees, splay heaps and pairing heaps had to be proved manually.

1998 ACM Subject Classification F.3.1 Specifying and Verifying Programs and F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Program Verification, Algorithm Analysis, Functional Programming

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.4

Category Invited Talk

1 Summary

Recent work on the analysis of functional data structures [6, 7] considers two questions: functional correctness and amortized complexity. In the theorem proving community, functional correctness of programs is the primary issue and their complexity is analyzed much less frequently. In the algorithms community it is the other way around: functional correctness is often viewed as obvious and the main issue is the complexity. We confirm the latter point of view in two case studies involving a number of functional search tree and priority queue implementations. The proofs were all conducted with the help of the theorem prover Isabelle/HOL [8, 9].

In [7] it is shown how to automate the functional correctness proofs of insertion and deletion in search trees: by means of an inorder traversal function that projects trees to lists, the proofs are reduced from trees to lists. With the help of a small lemma library, functional correctness and preservation of the search tree property are proved automatically for a range of data structures: unbalanced binary trees, AVL trees, red-black trees, 2-3 trees, 2-3-4 trees, 1-2 brother trees, AA trees and splay trees.

In [6] a framework for the analysis of the amortized complexity of (functional) data structures is formalized and applied to a number of standard examples and to three famous non-trivial ones: skew heaps, splay trees and splay heaps. More recently, pairing heaps were added in collaboration with Hauke Brinkop [2, 5]. In all cases we proved logarithmic amortized complexity and the proofs were largely manual, following the existing algorithms literature.

[*] Supported by DFG grant NI 491/16-1.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 4; pp. 4:1–4:2
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2 Related Work

Very close to the above work is Charguéraud' and Pottier's verification of the almost-linear amortized complexity of an OCaml implementation of Union-Find in Coq [3]. Using different methods but also aiming for performance analysis is work on automatic analysis of worst case execution time [11], analysis of complexity of term rewriting systems (e.g. [1, 10]), and automatic complexity analysis of functional programs (e.g. [4]).

References

1 Martin Avanzini, Georg Moser, and Michael Schaper. TcT: Tyrolean Complexity Tool. In M. Chechik and J.-F. Raskin, editors, *TACAS*, volume 9636 of *LNCS*, pages 407–423. Springer, 2016.

2 Hauke Brinkop. Verifikation der amortisierten Laufzeit von Pairing Heaps in Isabelle. Bachelor's thesis, Fakultät für Informatik, Technische Universität München, 2015.

3 Arthur Charguéraud and François Pottier. Machine-checked verification of the correctness and amortized complexity of an efficient union-find implementation. In C. Urban and X. Zhang, editors, *ITP 2015*, volume 9236 of *LNCS*, pages 137–153. Springer, 2015.

4 Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. Multivariate amortized resource analysis. *ACM Trans. Program. Lang. Syst.*, 34(3):14, 2012.

5 Tobias Nipkow. Amortized complexity verified. *Archive of Formal Proofs*, 2014. `http://isa-afp.org/entries/Amortized_Complexity.shtml`, Formal proof development.

6 Tobias Nipkow. Amortized complexity verified. In C. Urban and X. Zhang, editors, *ITP 2015*, volume 9236 of *LNCS*, pages 310–324. Springer, 2015.

7 Tobias Nipkow. Automatic functional correctness proofs for functional search trees. In J. Blanchette and S. Merz, editors, *ITP 2016*, LNCS. Springer, 2015. To appear.

8 Tobias Nipkow and Gerwin Klein. *Concrete Semantics with Isabelle/HOL*. Springer, 2014. URL: `http://concrete-semantics.org`.

9 Tobias Nipkow, Lawrence Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

10 Lars Noschinski, Fabian Emmes, and Jürgen Giesl. Analyzing innermost runtime complexity of term rewriting by dependency pairs. *J. Autom. Reasoning*, 51(1):27–56, 2013.

11 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David B. Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter P. Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem – overview of methods and survey of tools. *ACM Trans. Embedded Comput. Syst.*, 7(3), 2008.

Strong Normalization for the Parameter-Free Polymorphic Lambda Calculus Based on the Ω-Rule

Ryota Akiyoshi[1] and Kazushige Terui[2]

1 WIAS, Waseda University, Tokyo, Japan
georg.logic@gmail.com

2 RIMS, Kyoto University, Kyoto, Japan
terui@kurims.kyoto-u.ac.jp

Abstract

Following Aehlig [3], we consider a hierarchy $\mathbf{F}^p = \{\mathbf{F}^p_n\}_{n\in\mathbb{N}}$ of parameter-free subsystems of System $\mathbf{F}$, where each $\mathbf{F}^p_n$ corresponds to $\mathbf{ID}_n$, the theory of n-times iterated inductive definitions (thus our $\mathbf{F}^p_n$ corresponds to the $n + 1$th system of [3]). We here present two proofs of strong normalization for $\mathbf{F}^p_n$, which are directly formalizable with inductive definitions. The first one, based on the Joachimski-Matthes method, can be fully formalized in $\mathbf{ID}_{n+1}$. This provides a tight upper bound on the complexity of the normalization theorem for System $\mathbf{F}^p_n$. The second one, based on the Gödel-Tait method, can be locally formalized in $\mathbf{ID}_n$. This provides a direct proof to the known result that the representable functions in $\mathbf{F}^p_n$ are provably total in $\mathbf{ID}_n$. In both cases, Buchholz' Ω-rule plays a central role.

1998 ACM Subject Classification F4.1 Mathematical Logic

Keywords and phrases Polymorphic Lambda Calculus, Strong Normalization, Computability Predicate, Infinitary Proof Theory

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.5

1 Introduction

It is well known that the second-order predicate calculus admits cut-elimination as shown by Tait using a model theoretic method, which implies the consistency of the second-order Peano arithmetic **PA2** with full comprehension. Neither his proof nor its variants, however, are considered an ultimate solution to *Takeuti's conjecture* (cut-elimination for higher order logics)[1] from the viewpoint of traditional proof theory, since they do not fully elucidate the nature of impredicativity involved in second-order arithmetic. As it is quite hard to give a proof-theoretic analysis of **PA2** directly, people in proof theory have been working on its subsystems, such as Π^1_1-$\mathbf{CA}_0$, the second-order arithmetic with Π^1_1-comprehension, and $\mathbf{ID}_n$, the theory of n-times iterated inductive definitions[2]. An early important achievement

[1] Precisely speaking, Takeuti's conjecture asks for a "finitistic" proof of cut-elimination for higher order logics, where his finitistic stand point is indeed a considerable extension of Hilbert's original one. As to Takeuti's philosophical position, we refer to [16].

[2] After Gentzen's monumental cut-elimination theorem for **PA** in 1930's, Takeuti proved cut-elimination for Π^1_1-$\mathbf{CA}_0 + \mathbf{BI}$ (bar induction) in 1967 [15], and Feferman, Buchholz, Pohlers, and Sieg subsequently investigated theories of inductive definitions and Π^1_1-$\mathbf{CA}_0$ in 1970's [8]. For these developments of proof theory, we refer to Feferman's [10].

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 5; pp. 5:1–5:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is Π^1_1-$\mathbf{CA}_0 = \mathbf{ID}_{<\omega} = \bigcup_{n\in\mathbb{N}} \mathbf{ID}_n$, which provides a *reduction* of an impredicative theory Π^1_1-$\mathbf{CA}_0$ to a "predicative" one $\mathbf{ID}_{<\omega}$[3].

Translated into lambda calculus, **PA2** corresponds to System **F** and cut-elimination corresponds to normalization. As is well known, strong normalization for System **F** was proved by an extremely powerful and elegant technique known as the *Tait-Girard method* or the *reducibility candidates* argument [11]. However, the same question persists: does it really elucidate impredicativity? In fact, the proof by reducibility candidates does not give any reduction, but just defers the foundational issue to third-order arithmetic. Since the whole System **F** is too powerful, a realistic approach is to begin with its subsystems for which one can prove normalization in a "predicative" way.

Following Altenkirch, Coquand [5] and Aehlig [3], we consider the *parameter-free* fragment $\mathbf{F}^p$ of System **F** as well as its subsystems $\mathbf{F}^p_n$ so that $\mathbf{F}^p = \bigcup_{n\in\mathbb{N}} \mathbf{F}^p_n$ holds[4] (Section 2). Given a system L of lambda calculus and a system A of arithmetic, let us write $L \propto A$ if the representable functions in L coincide with the provably total functions in A. Then their main results can be stated as follows:

$$\mathbf{F}^p_0 \propto \mathbf{PA}\ [5], \qquad \mathbf{F}^p_n \propto \mathbf{ID}_n \text{ and } \mathbf{F}^p \propto \Pi^1_1\text{-}\mathbf{CA}_0\ [3].$$

As usual, one direction of the correspondence is established by a forgetful translation of arithmetical derivations into lambda terms. A little bit delicate is the other direction, which is shown by *locally* formalizing a normalization proof. By "local" we mean a term-wise formalization of the statement "$M\mathsf{n}$ is normalizable for every Church numeral n" for each fixed term $M : \boldsymbol{N} \Rightarrow \boldsymbol{N}$.

While the above is definitely a great achievement, it is not completely satisfactory since they only prove *weak* normalization for terms of specific type $\boldsymbol{N}$, and only provides a *local* formalization. Also, the argument in [3] is *indirect* as it passes through intermediate systems of second-order Heyting arithmetic proposed in [2].

The purpose of this paper is to improve the current situation by showing:

1. A proof of *strong* normalization for $\mathbf{F}^p_n$, which is *fully* formalizable in $\mathbf{ID}_{n+1}$ (Section 3).
2. Another proof of strong normalization for $\mathbf{F}^p_n$, which is locally but *directly* formalizable in $\mathbf{ID}_n$ (Section 4).

The first one is based on what we call the *Joachimski-Matthes method* (JM method), which is pioneered by [18] and established by [12] as methodology. It fits the "predicative" spirit of inductive definitions very well, and leads to a sharp upper bound on the complexity of the normalization theorem for $\mathbf{F}^p_n$ (as we know $\mathbf{ID}_n \nvdash \mathsf{SN}(\mathbf{F}^p_n)$, our result $\mathbf{ID}_{n+1} \vdash \mathsf{SN}(\mathbf{F}^p_n)$ is best possible).

The second proof is based on the standard computability argument, which we call the *Gödel-Tait method*[5]. It is particularly suitable for local formalization. Combined with the

[3] Here we use term "predicative" to refer to a system without circular definitions. That is, predicativity in the sense of Martin-Löf's type theory, *not* in the sense of Feferman's ordinal analysis. Indeed, the proof-theoretic ordinals of both systems are far beyond Γ_0, the limit of predicative ordinals. Our usage also conforms to [1], in which strong normalization is proved in a "predicative" way for lambda calculus with interleaving inductive data types.

[4] Our system $\mathbf{F}^p_n$ corresponds to the $n+1$th system $\mathbf{F}^\times_{n+1}$ of [3]. We keep using our notation to have a better correspondence with systems of arithmetic.

[5] As is well known, Tait introduced his computability argument in [13]. However, Troelstra pointed out in [17] that Gödel already suggested a similar idea in his Princeton notes, and Tait himself admitted that Gödel knew essentially the same argument at that time [14, p.115]. Hence, it is not unfair to call it the *Gödel*-Tait method.

JM method, it provides a direct proof to the results in [5] and [3] that the representable functions in System $\mathbf{F}^p_n$ are provably total in $\mathbf{ID}_n$ (recall that $\mathbf{ID}_0 = \mathbf{PA}$).

Apart from the technical results themselves, this paper exhibits two apparently orthogonal methods for strong normalization in a comparable way. Both are very rare proofs of normalization for an impredicative system which do not rely on reducibility candidates in any sense. Candidates are replaced by the Ω-*rule* of Buchholz [6, 9, 7], a well-known technique in proof theory (see below for an intuition). It has been used for ordinal analyses of the theories of iterated inductive definitions and iterations of Π^1_1-$\mathbf{CA}_0$, where an essential ingredient is a *partial* cut-elimination theorem for arithmetical sequents. It is recently extended to a *complete* cut-elimination theorem for arbitrary sequents by the first author and Mints [4]. One of our real motivations in this work is to bring this important technique to the realm of lambda calculus, where we do not yet find any explicit use of it[6].

An intuition of the Ω-rule. Let us conclude the introduction by giving an intuition of the Ω-rule. While it was originally introduced in the context of arithmetic, its basic idea can be explained in terms of the (standard) sequent calculus for second-order propositional intuitionistic logic. In what follows, we assume that (i) any second-order formula $\forall\alpha.A(\alpha)$ has a quantifier-free body $A(\alpha)$, and (ii) $\mathsf{Fv}(A(\alpha)) \subseteq \{\alpha\}$ (*parameter-free*). This corresponds to the restriction imposed by [5].

Recall that one of the most innovative ideas of Gentzen is to replace *axioms* with *rules:*

$$C \Rightarrow D \quad\mapsto\quad \frac{D, \Gamma \Rightarrow \Pi}{C, \Gamma \Rightarrow \Pi} \qquad\qquad D, \Gamma \Rightarrow \Pi \quad\mapsto\quad \frac{\Delta \Rightarrow D}{\Delta, \Gamma \Rightarrow \Pi}$$

so that we obtain a good cut-elimination procedure.

According to our understanding, the essence of the Ω-rule lies in applying this replacement *twice* to the comprehension axiom $\forall\alpha.A(\alpha) \Rightarrow A(B)$:

$$\forall\alpha.A(\alpha) \Rightarrow A(B) \quad\mapsto\quad \frac{A(B), \Gamma \Rightarrow \Pi}{\forall\alpha.A(\alpha), \Gamma \Rightarrow \Pi}\ (\forall l) \quad\mapsto\quad \frac{\{\ \Delta, \Gamma \Rightarrow \Pi\ \}_{\Delta \Rightarrow A(B)}}{\forall\alpha.A(\alpha), \Gamma \Rightarrow \Pi}$$

where the last rule has a premise $\Delta, \Gamma \Rightarrow \Pi$ for each provable sequent $\Delta \Rightarrow A(B)$.

Unfortunately, the last rule is not useful to inductively define the set of provable sequents, since the indices of the premises themselves depend on provability. To break this circularity, we consider a two-layered setting. Let us write $\Gamma \Rightarrow^\alpha A(\alpha)$ if $\Gamma \Rightarrow A(\alpha)$ is provable and $\alpha \notin \mathsf{Fv}(\Gamma)$ (the *eigenvariable* condition). We also write $\Gamma \Rightarrow^\alpha_{\mathsf{fo}} A(\alpha)$ if furthermore Γ and $A(\alpha)$ are quantifier-free. Since second-order intuitionistic logic is conservative over first-order one, the provability $\Gamma \Rightarrow^\alpha_{\mathsf{fo}} A(\alpha)$ can be defined without recourse to the second-order part.

We are now ready to introduce the Ω-rule corresponding to $\forall\alpha.A(\alpha) \Rightarrow A(B)$:

$$\frac{\{\ \Delta, \Gamma \Rightarrow \Pi\ \}_{\Delta \Rightarrow^\alpha_{\mathsf{fo}} A(\alpha)}}{\forall\alpha.A(\alpha), \Gamma \Rightarrow \Pi}\ (\Omega)$$

This rule indeed admits a well-defined reduction step:

$$\dfrac{\dfrac{\Sigma \Rightarrow^\alpha A(\alpha)}{\Sigma \Rightarrow \forall\alpha.A(\alpha)}\ (\forall r) \qquad \dfrac{\{\ \Delta, \Gamma \Rightarrow \Pi\ \}_{\Delta \Rightarrow^\alpha_{\mathsf{fo}} A(\alpha)}}{\forall\alpha.A(\alpha), \Gamma \Rightarrow \Pi}\ (\Omega)}{\Sigma, \Gamma \Rightarrow \Pi}\ (cut) \quad\longrightarrow\quad \Sigma, \Gamma \Rightarrow \Pi$$

[6] Article [2] mentioned above uses the Ω-rule for subsystems of second-order Heyting arithmetic, not directly for lambda calculus.

provided that Σ *is quantifier-free*. Indeed, we have $\Sigma \Rightarrow^{\alpha}_{\mathsf{fo}} A(\alpha)$ by the premise of $(\forall r)$, hence $\Sigma, \Gamma \Rightarrow \Pi$ is a premise of the Ω-rule.

Moreover, the standard left rule $(\forall l)$ for $\forall$, inferring $\forall\alpha.A(\alpha), \Gamma \Rightarrow \Pi$ from $A(B), \Gamma \Rightarrow \Pi$ (see above), can be simulated by the Ω-rule. To see this, suppose that $\Delta \Rightarrow^{\alpha}_{\mathsf{fo}} A(\alpha)$ (an index of the Ω-rule). We then obtain $\Delta \Rightarrow A(B)$ by substituting B for α, hence $\Delta, \Gamma \Rightarrow \Pi$ by the premise of $(\forall l)$. By (Ω), we conclude $\forall\alpha.A(\alpha), \Gamma \Rightarrow \Pi$.

Thus provability is preserved by replacing $(\forall l)$ with (Ω) (called *Embedding* in traditional proof theory), while cut-elimination can be proved *predicatively* (without any semantic argument), provided that the conclusion sequent is quantifier-free and the requirements (i) and (ii) above are satisfied (called *Collapsing*). This technique can be further extended to the *parameter-free* fragment of second order intuitionistic logic, which correspond to the system studied in [3].

2 System $\mathbf{F}^p$

2.1 Syntax

Given a countable set of *type variables* $\alpha, \beta, \gamma, \ldots$, we define the set Tp_n of *types at level* n for each $n \in \mathbb{N} \cup \{-1\}$ as follows:

$$A_n, B_n \quad ::= \quad \alpha \mid A_n \Rightarrow B_n \mid \forall\alpha.A_{n-1}$$

with the proviso that there is no type at level -2, and type $\forall\alpha.A_{n-1}$ can be formed only when $\mathsf{Fv}(A_{n-1}) \subseteq \{\alpha\}$. Here $\mathsf{Fv}(A)$ denotes the set of free type variables in A. That is to say, a quantified type $\forall\alpha.A$ is always *parameter-free*, so that we may treat it as a self-standing entity (like a data type). Let $\mathsf{Tp} := \bigcup_{n\in\mathbb{N}\cup\{-1\}} \mathsf{Tp}_n$.

Since there is no type at level -2, Tp_{-1} just consists of simple types. As it is unpleasant to refer to a negative integer, we write simp to denote the number -1. Thus $\mathsf{Tp}_{\mathsf{simp}} = \mathsf{Tp}_{-1}$. Types in Tp_0 are built by arrow $\Rightarrow$ from type variables and $\forall\alpha.A$, where A is a simple type over single variable α. For instance:

$$\begin{array}{rcllr} \boldsymbol{N} & := & \forall\alpha.(\alpha \Rightarrow \alpha) \Rightarrow (\alpha \Rightarrow \alpha) & \in \mathsf{Tp}_0 & \text{(natural numbers)} \\ \boldsymbol{T} & := & \forall\alpha.(\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow (\alpha \Rightarrow \alpha) & \in \mathsf{Tp}_0 & \text{(binary trees)} \\ \boldsymbol{L}(\boldsymbol{N}) & := & \forall\alpha.(\boldsymbol{N} \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow (\alpha \Rightarrow \alpha) & \in \mathsf{Tp}_1 & \text{(lists of nat. numbers)} \\ \boldsymbol{O} & := & \forall\alpha.((\boldsymbol{N} \Rightarrow \alpha) \Rightarrow \alpha) \Rightarrow (\alpha \Rightarrow \alpha) \Rightarrow (\alpha \Rightarrow \alpha) & \in \mathsf{Tp}_1 & \text{(Brouwer ordinals)} \end{array}$$

On the other hand, $\boldsymbol{L}(\beta) := \forall\alpha.(\beta \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow (\alpha \Rightarrow \alpha)$ is not a type. Hence the polymorphic map function, whose type would be

$$\forall\beta.\forall\gamma.(\beta \Rightarrow \gamma) \Rightarrow \boldsymbol{L}(\beta) \Rightarrow \boldsymbol{L}(\gamma),$$

is not representable in our setting. A more striking example is $\forall\beta.(\boldsymbol{L}(\beta) \Rightarrow \beta) \Rightarrow \beta$, which is the type for finitely but arbitrarily branching trees. Thus *interleaving* inductive data types (cf. [1]) are out of scope and left to future work.

An important property is that
(*) $A, B \in \mathsf{Tp}_n$ implies $A[B/\alpha] \in \mathsf{Tp}_n$,
where $[B/\alpha]$ stands for a substitution (which is always capture-free).

We now introduce terms, which are explicitly typed à la Church. We presuppose that a countable set Var of symbols $x, y, z, \ldots$ together with a distinguished symbol $c \notin \mathsf{Var}$ is provided. A *variable* is a pair of $x \in \mathsf{Var}$ and a type A, written x^A. Likewise a *constant* is a pair c^A. We never use x^A and x^B with $A \neq B$ together in the same context. Type

$$\frac{}{x^A \in X}\,(\mathsf{var}) \qquad \frac{}{c^A \in X}\,(\mathsf{con}) \qquad \frac{M^B \in X}{(\lambda x^A.M)^{A\Rightarrow B} \in X}\,(\mathsf{abs})$$

$$\frac{M^{A\Rightarrow B} \in X \quad N^A \in X}{(MN)^B \in X}\,(\mathsf{app}) \qquad \frac{M^A \in X \cap \mathsf{Ec}(\alpha)}{(\Lambda\alpha.M)^{\forall\alpha.A} \in X}\,(\mathsf{Abs}) \qquad \frac{M^A \in X}{(MB)^{A[B/\alpha]} \in X}\,(\mathsf{App})$$

Figure 1 Term rules: $\mathsf{Ec}(\alpha) = \{M : x^B \in \mathsf{fv}(M) \text{ implies } \alpha \notin \mathsf{Fv}(B)\}$.

annotations are often omitted when they are irrelevant. We write M^A to indicate that expression M has type A, and $\mathsf{fv}(M)$ to denote the set of free (term) variables in M.

The set Tm of *terms* is defined to be the least set closed under the *term rules* in Figure 1, where $\mathsf{Ec}(\alpha)$ is the set of terms M subject to the *eigenvariable condition* with respect to α: for any $x^B \in \mathsf{fv}(M)$, $\alpha \notin \mathsf{Fv}(B)$.

As usual, we assume that terms are identified up to α-equivalence. The reduction relation $\to$ is defined to be the contextual closure of

$$(\lambda x^A.M)N \to M[N/x^A], \qquad (\Lambda\alpha.M)B \to M[B/\alpha],$$

where $[N/x^A]$ stands for a capture-free substitution.

This defines the *System* $\mathbf{F}^p$. For each $n \in \mathbb{N} \cup \{\mathsf{simp}\}$, the subsystem $\mathbf{F}^p_n$ is obtained by restricting types to Tp_n and terms to $\mathsf{Tm}_n \subseteq \mathsf{Tm}$, which is obtained by restricting the types to Tp_n when applying the term rules. It is a legitimate definition since Tm_n is closed under reduction by (*).

Below are additional terminology and notational conventions. A term is *closed* if it does not contain a free term variable x (it may contain a free type variable α). We write $\mathsf{type}(M) = A$ if M is of type A. Symbols $T, T_0, T_1, \dots$ stand for a term or a type, and $\overline{T}$ for a list $T_1, \dots, T_n$ $(n \geq 0)$. The following convention turns out quite useful: when we write $\overline{T} \in X$, it means that all *terms* among $T_1, \dots, T_n$ belong to X, leaving types aside. Finally we assume that *all terms are well typed* throughout this paper. This means that we write MN only when $\mathsf{type}(M) = A \Rightarrow B$ and $\mathsf{type}(N) = A$. Likewise, we write $M[N/x^A]$ only when $\mathsf{type}(N) = A$, and MB only when $\mathsf{type}(M) = \forall\alpha.A$.

▶ Remark. $\mathbf{F}^p_{\mathsf{simp}}$ is nothing but the simply typed lambda calculus, while $\mathbf{F}^p_0$ exactly corresponds to the system of [5]. If the product types are added, our $\mathbf{F}^p_n$ corresponds to System $\mathrm{F}^{\times}_{n+1}$ of [3][7]. As noted in the introduction, it is known that $\mathbf{F}^p_0 \propto \mathbf{PA}$ and $\mathbf{F}^p_n \propto \mathbf{ID}_n$ for $n \in \mathbb{N}$.

2.2 Strongly normalizable terms

A term M is *strongly normalizable* if there is $n \in \mathbb{N}$ which bounds the length of any reduction sequence $M \equiv M_0 \to M_1 \to M_2 \to \cdots$. Since $\to$ is finitely branching, it is equivalent to say that there is no infinite reduction sequence from M by König's lemma. We prefer the former definition, since it is *arithmetical* (i.e., definable by a first-order formula of Peano arithmetic). Let SN be the set of strongly normalizable terms in Tm.

As is well known, the set SN admits an alternative inductive definition (cf. [19]).

[7] The second-order definition of product type $A \times B := \forall\alpha.(A \Rightarrow B \Rightarrow \alpha) \Rightarrow \alpha$ is not quite useful in our setting, since it would raise the level by one. However, all the results in this paper can be easily extended to systems with product types. Also, $\mathbf{F}^p_n \propto \mathbf{ID}_n$ holds in absence of product types.

$$\frac{\overline{T} \in X}{x\overline{T} \in X}\,(\mathsf{vap}) \qquad \frac{\overline{T} \in X}{c\overline{T} \in X}\,(\mathsf{cap}) \qquad \frac{M[N/x^A]\overline{T} \in X \quad N \in X}{(\lambda x^A.M)N\overline{T} \in X}\,(\beta) \qquad \frac{M[B/\alpha]\overline{T} \in X}{(\Lambda\alpha.M)B\overline{T} \in X}\,(\mathsf{B})$$

Figure 2 SN rules.

▶ **Lemma 1.** SN *coincides with the least set X closed under* (vap)*,* (cap)*,* (abs)*,* (Abs)*,* (β) *and* (B) *(see Figures 1 and 2).*

Most importantly, SN is closed under (β) (as well as (B)), a fact known as the *fundamental lemma of perpetuality* [20][8].

Notice that (var) and (con) are special cases of (vap) and (cap) with $\overline{T}$ empty. Since SN also satisfies (abs) and (Abs), we could conclude Tm $\subseteq$ SN (the *strong normalization theorem for* $\mathbf{F}^p$), if SN would satisfy (app) and (App). Of course we do not know that *a priori*. Hence proofs of strong normalization usually proceed as follows:

1. Define a set X which *approximates* SN.
2. Prove Tm $\subseteq X$ by showing that X is closed under the term rules (*Embedding*).
3. Prove $X \subseteq$ SN (*Collapsing*).

We may then conclude Tm $\subseteq X \subseteq$ SN, the strong normalization theorem.

2.3 Freezing

When discussing normalization of a lambda term, it is reasonable to distinguish two kinds of variable. For instance, consider $K \equiv \Lambda\alpha.\lambda x^{A(\alpha)}.N^{\alpha\Rightarrow\alpha}$. We immediately notice that variables α and x are never replaced by another expression during normalization, so can be treated as if they were constants. On the other hand, K may contain variables for which terms/types are actually substituted. In terms of proof theory, this corresponds to the distinction between *explicit* and *implicit* formulas [16, 4]. The following operation, called *freezing*, allows us to dynamically replace explicit bound variables with constants.

Let o be a distinguished type variable which we think of as constant. It is clear that for every type A, there is a unique list $\overline{t}$ of constants c and o such that $M\overline{t}$ is of atomic type for any $M^A \in$ Tm. We write $M^\circ := M\overline{t}$. For instance, $K^\circ = K\,o\,c^{A(o)}\,c^o$, which is of type o.

The following lemma is obvious, since SN is closed under subterms, and the reduction rules do not make a distinction between free variables and constants.

▶ **Lemma 2.** *Let $\sigma = [\overline{o}/\alpha, \overline{c}/\overline{x}]$ be a substitution which replace some free type variables with o and free term variables with c. If $(M\sigma)^\circ \in$ SN, then $M \in$ SN.*

3 Joachimski-Matthes method

We now present the first proof of strong normalization. It is based on the *JM method* established by Joachimski and Matthes, who gave a remarkably simple proof to strong normalization for the simply typed lambda calculus and its extensions, including System **T** [12]. It has a precursor [18], and owes the inductive characterization of SN to [19].

[8] Proofs of the lemma often rely on the definition of SN as the set of terms without infinite reduction sequences. It does not matter for our purpose, however, since **PA** can be conservatively extended to $\mathbf{ACA}_0$, in which König's lemma is available. Thus **PA** proves the lemma. Also, a very careful argument based on the other definition of SN can be found in [12, p.68] (footnote 18).

$$\frac{\overline{M} \in X}{x\overline{M} \in X}\,(\mathsf{vap}^-) \qquad \frac{\overline{T^\circ} \in X}{c\overline{T} \in X}\,(\mathsf{cap}^\circ) \qquad \frac{M \in X}{\lambda x^A.M \in X}\,(\mathsf{abs})$$

$$\frac{M \in X \cap \mathsf{Ec}(\alpha)}{\Lambda\alpha.M \in X}\,(\mathsf{Abs}) \qquad \frac{M[N/x^A]\overline{T} \in X \quad N^\circ \in X}{(\lambda x^A.M)N\overline{T} \in X}\,(\beta^\circ) \qquad \frac{M[B/\alpha]\overline{T} \in X}{(\Lambda\alpha.M)B\overline{T} \in X}\,(\mathsf{B})$$

Figure 3 JM rules.

We begin with the simply typed lambda calculus $\mathbf{F}^p_{\mathsf{simp}}$ in 3.1. This will be useful for recalling the JM method, and also as the basis for higher level systems $\mathbf{F}^p_n$ with $n \geq 0$. The latter will be dealt with in 3.2. Finally we will informally discuss formalization in the theories of inductive definitions in 3.3.

3.1 Simply typed case

Let us begin with defining a suitable *term domain* for each $\mathbf{F}^p_n$, in which an approximating set $X \subseteq \mathsf{SN}$ is to be defined. The set Tm_n is not suitable. As it will turn out, it is crucially important to include as many terms as possible, while restricting the types of free variables and whole terms. Below is the right definition.

For each $n \in \mathbb{N} \cup \{\mathsf{simp}\}$, we define a set Dom_n as follows:

$$\mathsf{Dom}_n := \{M \in \mathsf{Tm} : \mathsf{type}(\mathsf{fv}(M)) \subseteq \mathsf{Tp}_n,\ \mathsf{type}(M) \in \forall\mathsf{Tp}_n\},$$

where $\forall\mathsf{Tp}_n := \mathsf{Tp}_n \cup \{\forall\alpha.A : A \in \mathsf{Tp}_n\}$.

Thus all free variables of $M \in \mathsf{Dom}_{\mathsf{simp}}$ have quantifier-free types, and $\mathsf{type}(M)$ is either quantifier-free or a quantified type $\forall\alpha.A \in \mathsf{Tp}_0$. Let us emphasize that the definition of Dom_n is only concerned with the types of free variables and whole terms, *not* with the internal structure of terms at all. Thus $\mathsf{Tm}_n \subsetneq \mathsf{Dom}_n$.

We now define the first approximating set $X \subseteq \mathsf{SN}$, which we call the *JM predicate at level* -1.

▶ **Definition 3.** Let $\mathbf{JM}_{\mathsf{simp}}$ be the least set $X \subseteq \mathsf{Dom}_{\mathsf{simp}}$ closed under the rules in Figure 3, called the *JM rules.*

Compared with the rules defining SN (Lemma 1), rule (vap) is restricted to (vap^-). This will be important in Lemma 6, where we argue by induction on the $\Rightarrow$-rank of a type. Another difference is that the freezing operator is employed in (cap°) and (β°). This results in a very pleasant property: for any $n \in \mathbb{N}$ and any JM rule, if the conclusion term belongs to Dom_n, so do the premise terms. For instance, look at (cap°). Even though $c\overline{T} \in \mathsf{Dom}_n$, the type of each term T_i may be quite complicated. Still, $\mathsf{type}(T_i^\circ)$ is atomic so that T_i° belongs to Dom_n. Observe that the same is true of (vap^-), because the type of each M_i in $x^A\overline{M}$ is a subtype of A.

▶ **Lemma 4** (Collapsing). $\mathbf{JM}_{\mathsf{simp}} \subseteq \mathsf{SN}$.

Proof. SN is closed under the JM rules by Lemma 1; notice that closure under (cap°) and (β°) is ensured by Lemma 2. ◀

We next proceed to Embedding ($\mathsf{Tm}_{\mathsf{simp}} \subseteq \mathbf{JM}_{\mathsf{simp}}$). We already have (var), (con), (abs) and (Abs) (though redundant), while (App) is not needed for $\mathbf{F}^p_{\mathsf{simp}}$. Hence it just remains to show that $\mathbf{JM}_{\mathsf{simp}}$ is closed under (app).

$$\frac{L \in X}{L^\circ \in X}\,(\mathsf{frz}) \qquad \frac{L \in X \quad B \in \mathsf{Tp}_n}{L[B/\alpha] \in X}\,(\mathsf{Sub}_n) \qquad \frac{L \in X \quad K^C \in X}{LK \in X}\,(\mathsf{app}) \qquad \frac{L \in X \quad K^C \in X}{L[K/y^C] \in X}\,(\mathsf{sub})$$

Figure 4 Additional rules.

▶ **Lemma 5.** $\mathbf{JM}_{\mathsf{simp}}$ *is closed under* (frz) *in Figure 4.*

Proof. We prove a more general claim: let $\sigma = [\overline{o}/\overline{\alpha}, \overline{c}/\overline{x}]$ be a substitution of o, c for free type variables and term variables, and $\overline{t}$ a list of o, c such that $(L\sigma)\overline{t}$ is well typed. Then $L \in \mathbf{JM}_{\mathsf{simp}}$ implies $(L\sigma)\overline{t} \in \mathbf{JM}_{\mathsf{simp}}$.

The proof proceeds by induction on the derivation of $L \in \mathbf{JM}_{\mathsf{simp}}$.

- $L \equiv xM_1 \cdots M_n$ is derived from $\overline{M} \in \mathbf{JM}_{\mathsf{simp}}$ by (vap^-). Then $\overline{t}$ does not contain o, since $\mathsf{type}(L)$ is quantifier-free (as x is). Suppose that $x\sigma = x$ (the case $x\sigma = c$ is similar). By the IH, we have $M_i\sigma \in \mathbf{JM}_{\mathsf{simp}}$. Hence $(L\sigma)\overline{t} \equiv x(M_1\sigma)\cdots(M_n\sigma)\overline{t} \in \mathbf{JM}_{\mathsf{simp}}$ by (vap^-).
- $L \equiv \Lambda\alpha.M \in \mathbf{JM}_{\mathsf{simp}}$ is derived from $M \in \mathbf{JM}_{\mathsf{simp}}$ by (Abs). We may assume that $\alpha\sigma = \alpha$ and $\overline{t}$ is of the form $o, \overline{u}$, if not empty. By the IH, we have $(M\sigma[o/\alpha])\overline{u} \in \mathbf{JM}_{\mathsf{simp}}$. Hence $(L\sigma)\overline{t} \equiv (\Lambda\alpha.M\sigma)o\overline{u} \in \mathbf{JM}_{\mathsf{simp}}$ by (B).

The other cases are similar. ◀

The next lemma is the highlight of the JM method. Given a type A, its $\Rightarrow$-*rank* is defined as follows:

$$\mathsf{rk}(\alpha) = \mathsf{rk}(\forall\alpha.A) := 0, \qquad \mathsf{rk}(A \Rightarrow B) := \max\{\mathsf{rk}(A) + 1, \mathsf{rk}(B)\}.$$

▶ **Lemma 6.** $\mathbf{JM}_{\mathsf{simp}}$ *is closed under* (app) *and* (sub) *in Figure 4.*

Proof. By main induction on $\mathsf{rk}(C)$ and side induction on the derivation of $L \in \mathbf{JM}_{\mathsf{simp}}$.

For (sub), we consider two cases; the first one is crucial, while the second one is a typical one, from which the other cases are easily understood.

- $L \equiv x^A\overline{M} \in \mathbf{JM}_{\mathsf{simp}}$ is derived from $\overline{M} \equiv M_1, \dots, M_n \in \mathbf{JM}_{\mathsf{simp}}$ by (vap^-). Suppose that $y^C \equiv x^A$ so that $L[K/y] \equiv K(M_1[K/y])\cdots(M_n[K/y])$ (the case $y \not\equiv x$ is easier). This means that C is of the form $B_1 \Rightarrow \cdots B_n \Rightarrow B_0$. By the side IH (sub) we have $M_i[K/y]^{B_i} \in \mathbf{JM}_{\mathsf{simp}}$. Since $\mathsf{rk}(B_i) < \mathsf{rk}(C)$ we may apply the main IH (app) (n times) to conclude that $L[K/y] \in \mathbf{JM}_{\mathsf{simp}}$.
- $L \equiv (\lambda x.M)N\overline{T} \in \mathbf{JM}_{\mathsf{simp}}$ is derived from $M[N/x]\overline{T} \in \mathbf{JM}_{\mathsf{simp}}$ and $N^\circ \in \mathbf{JM}_{\mathsf{simp}}$. Let us use a tentative notation $M' := M[K/y]$. Then $L[K/y]$ can be written as $(\lambda x.M')N'\overline{T'}$. By the side IH, we have $M'[N'/x]\overline{T'} \in \mathbf{JM}_{\mathsf{simp}}$ and $(N')^\circ \equiv (N^\circ)' \in \mathbf{JM}_{\mathsf{simp}}$, hence $L[K/y] \in \mathbf{JM}_{\mathsf{simp}}$ by (β°).

For (app), we again consider two cases.

- $L \equiv \lambda x.M \in \mathbf{JM}_{\mathsf{simp}}$ is derived from $M \in \mathbf{JM}_{\mathsf{simp}}$ by (abs). We have $M[K/x] \in \mathbf{JM}_{\mathsf{simp}}$ by the side IH (sub), hence $(\lambda x.M)K \in \mathbf{JM}_{\mathsf{simp}}$ by (β°), noting that $K^\circ \in \mathbf{JM}_{\mathsf{simp}}$ follows from $K \in \mathbf{JM}_{\mathsf{simp}}$ by the previous lemma.
- $L \equiv x\overline{M} \in \mathbf{JM}_{\mathsf{simp}}$ is derived from $\overline{M} \in \mathbf{JM}_{\mathsf{simp}}$ by (vap^-). We may add a new premise $K \in \mathbf{JM}_{\mathsf{simp}}$ to obtain $x\overline{M}K \in \mathbf{JM}_{\mathsf{simp}}$. ◀

Since $\mathbf{JM}_{\mathsf{simp}}$ satisfies (var), (con), (abs) and (app), we have $\mathsf{Tm}_{\mathsf{simp}} \subseteq \mathbf{JM}_{\mathsf{simp}} \subseteq \mathsf{SN}$ (Embedding). This completes the proof.

▶ **Theorem 7.** $\mathbf{F}^p_{\mathsf{simp}}$ *admits strong normalization.*

$$\frac{M^{\forall\alpha.A} \in X \quad \{\, K[B/\alpha]\overline{T} \in X \,\}_{K^A \in \mathbf{JM}_{n-1} \cap \mathsf{Ec}(\alpha)}}{MB\overline{T} \in X}\,(\Omega_n) \qquad \frac{M^{\forall\alpha.A} \in X \quad B \in \mathsf{Tp}_n}{MB \in X}\,(\mathsf{App}_n)$$

Figure 5 (Ω_n) and (App_n).

3.2 Inductive cases

Now a crucial question is how to extend $\mathbf{JM}_{\mathsf{simp}}$ so that it also accommodates (App). Extending (vap^-) to (vap) would totally spoil the fine-tuned structure of the JM method, as the use of induction on $\mathsf{rk}(C)$ in the proof of Lemma 6 would not work anymore. We will instead adopt a brilliant idea due to Buchholz: the Ω*-rule*.

▶ **Definition 8.** Let $\mathbf{JM}_0$ be the least set $X \subseteq \mathsf{Dom}_0$ closed under the JM rules (Figure 3) and (Ω_0) (Figure 5). More generally, $\mathbf{JM}_n$ $(n \in \mathbb{N})$ is defined to be the least set $X \subseteq \mathsf{Dom}_n$ closed under the JM rules and $(\Omega_0), \dots, (\Omega_n)$. $\mathbf{JM}_n$ is called the *JM predicate at level* n.

Rule (Ω_0) has a premise $K[B/\alpha]\overline{T} \in X$ for each $K \in \mathbf{JM}_{\mathsf{simp}} \cap \mathsf{Ec}(\alpha)$. Thus it depends on the set $\mathbf{JM}_{\mathsf{simp}}$, which has been already defined. In general, $\mathbf{JM}_n$ is obtained from $\mathbf{JM}_{n-1}$ by extending the term domain to Dom_n and by adding a new rule (Ω_n), which depends on $\mathbf{JM}_{n-1}$. Hence we have $\mathbf{JM}_{n-1} \subseteq \mathbf{JM}_n$ by definition. Notice that B is an *arbitrary* type in Tp; it is condition $K \in \mathsf{Ec}(\alpha)$ that ensures that $K[B/\alpha]\overline{T}$ belongs to Dom_n as far as $MB\overline{T}$ does.

▶ **Lemma 9.** $\mathbf{JM}_n$ *is closed under* (App_n) *(Figure 5).*

Proof. Suppose that $M^{\forall\alpha.A} \in \mathbf{JM}_n$ and $B \in \mathsf{Tp}_n$. For each $K^A \in \mathbf{JM}_{n-1} \cap \mathsf{Ec}(\alpha)$ we have $K \in \mathbf{JM}_n$ and so $K[B/\alpha] \in \mathbf{JM}_n$ by Lemma 10 below. Hence we obtain $MB \in \mathbf{JM}_n$ by (Ω_n). ◀

▶ Remark. The Ω-rule is often called an *impredicative* cut. In the current situation, it can be thought of as a *meta-cut* on derivations, rather than a redex occurring in a term M. Imagine that rule (Abs) is sort of an "introduction rule" in natural deduction. Then (Ω_n) provides a matching "elimination rule" with a notion of "reduction":

$$\frac{\dfrac{N^A \in \mathbf{JM}_n}{\Lambda\alpha.N \in \mathbf{JM}_n}\,(\mathsf{Abs}) \qquad \left\{\, \begin{array}{c} \vdots\ \pi_K \\ K[B/\alpha] \in \mathbf{JM}_n \end{array} \,\right\}_{K^A \in \mathbf{JM}_{n-1} \cap \mathsf{Ec}(\alpha)}}{(\Lambda\alpha.N)B \in \mathbf{JM}_n}\,(\Omega_n) \quad \Longrightarrow \quad \frac{\begin{array}{c} \vdots\ \pi_N \\ N[B/\alpha] \in \mathbf{JM}_n \end{array}}{(\Lambda\alpha.N)B \in \mathbf{JM}_n}\,(\mathsf{B})$$

which is triggered by showing $N^A \in \mathbf{JM}_{n-1}$.

The Ω-rule was first introduced by Buchholz [6] to give ordinal analyses of iterated inductive definitions. His main theorem called *Collapsing* amounts to a partial cut-elimination theorem for derivations of arithmetical sequents. Later it is extended to a complete cut-elimination theorem for the Ω-rule by the first author and Mints [4]. In these developments, it is always a crucial issue how to define or extend the "domain" of the Ω-rule. A technical contribution of this paper is that we have managed to include strongly normalizable terms in the domain, in contrast to the "proof theoretic" domains which consist of normal (cut-free) derivations.

Coming back to the formal argument, it is not hard to extend (frz), (app) and (sub) (Figure 4) to $\mathbf{JM}_n$. We also consider a new rule (Sub_n).

▶ **Lemma 10.** *For every* $n \in \mathbb{N} \cup \{\mathsf{simp}\}$, $\mathbf{JM}_n$ *is closed under* (frz), (Sub_n), (app) *and* (sub).

Hence $\mathbf{JM}_n$ satisfies all of (var), (con), (abs), (Abs), (app) and (App_n). We therefore conclude:

▶ **Lemma 11** (Embedding). *For every $n \in \mathbb{N} \cup \{\mathsf{simp}\}$, $\mathsf{Tm}_n \subseteq \mathbf{JM}_n$.*

Let us now move on to the Collapsing part. We first need an inversion lemma for (cap°).

▶ **Lemma 12.** *If $c\overline{T} \in \mathbf{JM}_n$, then $\overline{T^\circ} \in \mathbf{JM}_n$.*

Proof. By induction on the derivation. It is obvious if $c\overline{T} \in \mathbf{JM}_n$ is derived by (cap°). Otherwise, it is derived by (Ω_m) ($m \leq n$):

$$\frac{(c\overline{T_1})^{\forall\alpha.A} \in \mathbf{JM}_n \quad \{\ K[B/\alpha]\overline{T_2} \in \mathbf{JM}_n\ \}_{K^A \in \mathbf{JM}_{m-1} \cap \mathsf{Ec}(\alpha)}}{c\overline{T_1}B\overline{T_2} \in \mathbf{JM}_n}$$

Let $K := c^A$ to obtain $c^{A[B/\alpha]}\overline{T_2} \in \mathbf{JM}_n$. By the IH (twice), we have $\overline{T_1^\circ}, \overline{T_2^\circ} \in \mathbf{JM}_n$. ◀

The next lemma lies at the heart of the Ω-rule technique. It describes a "meta-cut elimination procedure" to eliminate (Ω_{n+1}) from a derivation in $\mathbf{JM}_{n+1}$.

▶ **Lemma 13.** *$\mathbf{JM}_n$ satisfies (Ω_{n+1}):*

$$\frac{M^{\forall\alpha.A} \in \mathbf{JM}_n \quad \{\ K[B/\alpha]\overline{T} \in \mathbf{JM}_n\ \}_{K^A \in \mathbf{JM}_n \cap \mathsf{Ec}(\alpha)}}{MB\overline{T} \in \mathbf{JM}_n}$$

Proof. By induction on the derivation of $M^{\forall\alpha.A} \in \mathbf{JM}_n$.

- $M \equiv \Lambda\alpha.N \in \mathbf{JM}_n$ is derived by (Abs). Then $N^A \in \mathbf{JM}_n \cap \mathsf{Ec}(\alpha)$, so let $K := N$ to obtain $N[B/\alpha]\overline{T} \in \mathbf{JM}_n$. Hence $MB\overline{T} \in \mathbf{JM}_n$ by (B).
- $M \equiv x^C\overline{N}$ is derived by (vap^-). $C \in \mathsf{Tp}_n$ implies $\forall\alpha.A \in \mathsf{Tp}_n$, so $A \in \mathsf{Tp}_{n-1}$. Moreover, $\mathbf{JM}_{n-1} \subseteq \mathbf{JM}_n$. Hence we may apply ($\Omega_n$) to obtain the same conclusion[9].
- $M \equiv c\overline{U} \in \mathbf{JM}_n$ is derived by (cap°). Let $K := c^A$ to obtain $c^{A[B/\alpha]}\overline{T} \in \mathbf{JM}_n$. By Lemma 12, we have $\overline{U^\circ}, \overline{T^\circ} \in \mathbf{JM}_n$. Hence we obtain $MB\overline{T} \equiv c\overline{U}B\overline{T} \in \mathbf{JM}_n$ by (cap°).
- $M \equiv NC\overline{U}$ is obtained by (Ω_m) with $m \leq n$:

$$\frac{N^{\forall\beta.D} \in \mathbf{JM}_n \quad \{\ L[C/\beta]\overline{U} \in \mathbf{JM}_n\ \}_{L^D \in \mathbf{JM}_{m-1} \cap \mathsf{Ec}(\beta)}}{NC\overline{U} \in \mathbf{JM}_n}$$

  For each $L^D \in \mathbf{JM}_{m-1} \cap \mathsf{Ec}(\beta)$ we have $(L[C/\beta]\overline{U})^{\forall\alpha.A} \in \mathbf{JM}_n$. So $L[C/\beta]\overline{U}B\overline{T} \in \mathbf{JM}_n$ by the IH. Hence we obtain $MB\overline{T} \equiv NC\overline{U}B\overline{T} \in \mathbf{JM}_n$ by (Ω_m).

It never happens that $M^{\forall\alpha.A} \in \mathbf{JM}_n$ is derived by (abs). The cases of (β°) and (B) easily follow from the IH. ◀

The next lemma follows immediately, since $\mathbf{JM}_{n+1}$ reduces to $\mathbf{JM}_n$ by restricting the term domain to Dom_n and eliminating (Ω_{n+1}).

▶ **Lemma 14.** *For every $n \in \mathbb{N} \cup \{\mathsf{simp}\}$, $\mathbf{JM}_{n+1} \cap \mathsf{Dom}_n = \mathbf{JM}_n$.*

As a consequence, we obtain:

▶ **Lemma 15** (Collapsing). *For every $n \in \mathbb{N} \cup \{\mathsf{simp}\}$, $\mathbf{JM}_n \subseteq \mathsf{SN}$.*

[9] Though it looks innocent, this is indeed the bottle neck of the whole argument. We restricted the term domain to Dom_n and introduced constants and freezing just for managing this case.

Proof. Given $M \in \mathbf{JM}_n$, let $N := M^\circ[\overline{c}/\overline{x}]$ be the closed term of atomic type obtained by freezing and constant substitution. Then $N \in \mathsf{Dom}_{\mathsf{simp}}$ and $N \in \mathbf{JM}_n$ by Lemma 10. Hence by Lemma 14 and Lemma 4, we obtain:

$$N \in \mathbf{JM}_n \cap \mathsf{Dom}_{\mathsf{simp}} \ = \ \mathbf{JM}_{n-1} \cap \mathsf{Dom}_{\mathsf{simp}} \ = \ \cdots \ = \ \mathbf{JM}_0 \cap \mathsf{Dom}_{\mathsf{simp}} \ = \ \mathbf{JM}_{\mathsf{simp}} \ \subseteq \ \mathsf{SN}.$$

From this, we conclude $M \in \mathsf{SN}$ by Lemma 2. ◀

▶ **Theorem 16.** *For each $n \in \mathbb{N} \cup \{\mathsf{simp}\}$, $\mathbf{F}^p_n$ admits strong normalization. Hence $\mathbf{F}^p$ admits strong normalization too.*

3.3 Formalization in $\mathbf{ID}_{<\omega}$

Let us recall the theories $\{\mathbf{ID}_n\}_{n\in\mathbb{N}}$ of finitely iterated inductive definitions. $\mathbf{ID}_0$ is just the standard first-order Peano arithmetic **PA**.

$\mathbf{ID}_1$ is obtained as follows. Let $A \equiv A(\boldsymbol{X}, x)$ be a first order arithmetical formula with (temporarily used) second-order variable $\boldsymbol{X}$ which occurs positively. x is a first-order variable, and we suppose that A does not contain any other free variables. For each such A, we extend the language with a new unary predicate $\boldsymbol{I}_A(x)$ together with the axioms

$$A[\boldsymbol{I}_A/\boldsymbol{X}] \subseteq \boldsymbol{I}_A, \qquad A[S/\boldsymbol{X}] \subseteq S \to \boldsymbol{I}_A \subseteq S,$$

where $S \equiv S(x, \overline{y})$ is an arbitrary formula and $B \subseteq C$ abbreviates $\forall x(B(x) \to C(x))$. Intuitively, A expresses a monotone operator $\wp(\mathbb{N}) \longrightarrow \wp(\mathbb{N})$ and $\boldsymbol{I}_A$ its least fixed point. This defines $\mathbf{ID}_1$.

$\mathbf{ID}_n$ with $n > 1$ is defined similarly, except that formula A can be taken from the language of $\mathbf{ID}_{n-1}$. Thus we are allowed to define a new fixed point making use of previous ones. Let $\mathbf{ID}_{<\omega} := \bigcup_{n\in\mathbb{N}} \mathbf{ID}_n$. It is known that $\mathbf{ID}_{<\omega}$ proves exactly the same arithmetical sentences as Π^1_1-$\mathbf{CA}_0$, the second-order Peano arithmetic with Π^1_1-comprehension.

Let us now discuss formalization of Theorem 16. We may assume a reasonable encoding of lambda terms as natural numbers and basic operations as primitive recursive functions (see [2]). The sets SN and $\mathbf{JM}_{\mathsf{simp}}$, as well as the associated induction principles, are available in **PA**, since these are defined by *finitary* rules. For instance, one can define $M \in \mathbf{JM}_{\mathsf{simp}}$ as "there exists a derivation d ending with statement $M \in \mathbf{JM}_{\mathsf{simp}}$," which is arithmetical since d, a finite object, is encodable by a natural number.

On the other hand, the definition of $\mathbf{JM}_0$ involves (Ω_0), which is *infinitary*. It is here that inductive definitions play a role. Indeed, $\mathbf{ID}_1$ allows us to define $\mathbf{JM}_0$ quite smoothly. For $n > 0$, recall that $\mathbf{JM}_n$ involves $(\Omega_0), \ldots, (\Omega_n)$, which depend on $\mathbf{JM}_0, \ldots, \mathbf{JM}_{n-1}$ (as well as $\mathbf{JM}_{\mathsf{simp}}$, which is arithmetical and thus negligible). Hence definition of $\mathbf{JM}_n$ requires $\mathbf{ID}_{n+1}$.

Once $\mathbf{JM}_n$ has been defined, the rest of argument proceeds by induction on the derivation and some inductions on natural numbers, all of which are available in $\mathbf{ID}_{n+1}$ (see also footnote 8). We therefore conclude:

▶ **Theorem 17.** *$\mathbf{ID}_{n+1}$ proves strong normalization for $\mathbf{F}^p_n$.*

Since $\mathbf{F}^p_n \propto \mathbf{ID}_n$ [2, 3], normalization for $\mathbf{F}^p_n$ implies consistency of $\mathbf{ID}_n$ (if $\mathbf{ID}_n$ were not consistent, it would prove totality of a partial function, and $\mathbf{F}^p_n$ would represent it by a lambda term of type $\boldsymbol{N} \Rightarrow \boldsymbol{N}$, contradicting normalization). Hence by the second incompleteness theorem, $\mathbf{ID}_n$ does not prove normalization for $\mathbf{F}^p_n$. Therefore Theorem 17 is the best possible we may obtain.

Turning to the whole system, normalization for $\mathbf{F}^p$ cannot be proved in $\mathbf{ID}_{<\omega}$ for the same reason. Still, $\mathbf{ID}_{<\omega}$ has a proper extension $\mathbf{ID}_\omega$ which corresponds to Π^1_1-$\mathbf{CA}_0 + \mathbf{BI}$ (bar induction). We have:

▶ **Theorem 18.** $\mathbf{ID}_\omega$ *proves strong normalization for* $\mathbf{F}^p$*.*

4 Gödel-Tait method

The previous proof, fully based on the JM method, works almost fine. It is, however, not amenable to local formalization, so cannot be used to prove that the representable functions in $\mathbf{F}^p_n$ are provably total in $\mathbf{ID}_n$, since it involves $n+1$ times iteration of inductive definitions in an unavoidable way. Hence we are led back to a more conventional approach, which we call the *Gödel-Tait method* for the reason explained in footnote 5. It works perfectly, when combined with the JM method. In this section, we give an alternative proof of strong normalization in 4.1 and discuss local formalization in 4.2.

4.1 Computability predicates

Throughout this section, we fix $n \in \mathbb{N} \cup \{\mathsf{simp}\}$ and assume that the JM predicate $\mathbf{JM}_n$ at level n has been defined. Our goal is to give an alternative proof to strong normalization for $\mathbf{F}^p_{n+1}$ by building a *computability predicate* on top of $\mathbf{JM}_n$. In the sequel, we write $\mathbf{JM} := \mathbf{JM}_n$ just for simplicity.

Anticipating local formalization later, we will work with a restricted set of types. Given a set $\mathcal{X} \subseteq \mathsf{Tp}_{n+1}$, let $\mathcal{X}_\downarrow$ be the least set containing $\mathcal{X}$ and satisfying the following conditions:

1. $A \Rightarrow B \in \mathcal{X}_\downarrow$ implies $A, B \in \mathcal{X}_\downarrow$.
2. $\forall\alpha.A \in \mathcal{X}_\downarrow$ and $D \in \mathcal{X}$ imply $A[D/\alpha] \in \mathcal{X}_\downarrow$.

It is clear that $\mathcal{X}_\downarrow$ is finite whenever $\mathcal{X}$ is.

Recall that $\mathbf{JM} = \mathbf{JM}_n \subseteq \mathsf{Dom}_n$. To address strong normalization for $\mathbf{F}^p_{n+1}$, we enlarge the domain Dom_n with terms of type $\mathcal{X}_\downarrow$. Let

$$\mathsf{Dom}(\mathcal{X}) := \{M \in \mathsf{Tm} : \mathsf{type}(\mathsf{fv}(M)) \subseteq \mathsf{Tp}_n,\ \mathsf{type}(M) \in \forall\mathsf{Tp}_n \cup \mathcal{X}_\downarrow\}.$$

In particular when $\mathcal{X}$ is finite, $\forall\mathsf{Tp}_n \cup \mathcal{X}_\downarrow$ consists of $\forall\mathsf{Tp}_n$ together with finitely many $\Rightarrow$-types in Tp_{n+1}. Given $X, Y \subseteq \mathsf{Dom}(\mathcal{X})$, let us write

$$X \Rightarrow Y := \{M \in \mathsf{Dom}(\mathcal{X}) : \forall N \in X.\ MN \in Y\}.$$

Also, let $\mathbf{JM}(A) := \{M \in \mathbf{JM} : \mathsf{type}(M) = A\}$ for each $A \in \forall\mathsf{Tp}_n$.

Our first observation is the following:

▶ **Lemma 19.** *If* $A, B \in \mathsf{Tp}_n$*,* $\mathbf{JM}(A \Rightarrow B) = \mathbf{JM}(A) \Rightarrow \mathbf{JM}(B)$*.*

Proof. The inclusion $\subseteq$ is due to (app) already established for $\mathbf{JM} = \mathbf{JM}_n$ by Lemma 10. For the other inclusion, let $M \in \mathbf{JM}(A) \Rightarrow \mathbf{JM}(B)$. Since $x \in \mathbf{JM}(A)$ by (var), we have $Mx \in \mathbf{JM}(B)$ with x a fresh variable. We can easily show that $M \in \mathbf{JM}$ by induction on the derivation of $Mx \in \mathbf{JM}$.

The only nontrivial case is when $(\lambda y.N)x \in \mathbf{JM}$ is derived from $N[x/y] \in \mathbf{JM}$ by (β°). In this case, we have $\lambda y.N \equiv \lambda x.N[x/y] \in \mathbf{JM}$ by (abs). ◀

Notice that this is a *consequence* of the definition of JM predicates. The Gödel-Tait method works the other way round; we *define* a predicate by the above property, and then derive JM-like properties as consequences.

▶ **Definition 20.** For each $C \in \forall\mathsf{Tp}_n \cup \mathcal{X}_\downarrow$, we define a set $\mathbf{CP}(C) \subseteq \mathsf{Dom}(\mathcal{X})$ as follows.

$$\begin{array}{rcll} \mathbf{CP}(C) & := & \mathbf{JM}(C) & (C \in \forall\mathsf{Tp}_n) \\ & := & \mathbf{CP}(A) \Rightarrow \mathbf{CP}(B) & (C \equiv A \Rightarrow B \in \mathcal{X}\backslash\forall\mathsf{Tp}_n) \end{array}$$

Let $\mathbf{CP} = \mathbf{CP}(\mathcal{X}) := \bigcup_{C \in \forall\mathsf{Tp}_n \cup \mathcal{X}_\downarrow} \mathbf{CP}(C)$. This defines the *computability predicate at level* $n+1$, relative to $\mathcal{X}$.

▶ **Lemma 21.** *The set* $\mathbf{CP}$ *satisfies* (app), (Abs), (β°), (B), (Ω_m) *($m \leq n+1$) as well as*

$$\frac{M^C \in X}{M^\circ \in \mathbf{JM}}\ (\mathsf{sn}^\circ) \qquad \frac{\overline{T^\circ} \in \mathbf{JM}}{(c\overline{T})^C \in X}\ (\mathsf{csn}^\circ)$$

Proof. (app) is a consequence of the definition and Lemma 19. (Abs) is trivial. Indeed, if $\forall\alpha.A \in \forall\mathsf{Tp}_n \cup \mathcal{X}_\downarrow$, then $\forall\alpha.A \in \forall\mathsf{Tp}_n$ and $A \in \mathsf{Tp}_n$. Hence we have $\mathbf{CP}(A) = \mathbf{JM}(A)$ and $\mathbf{CP}(\forall\alpha.A) = \mathbf{JM}(\forall\alpha.A)$ so that it boils down to (Abs) for $\mathbf{JM}$.

(sn°) and (csn°) are simultaneously verified by induction on $\mathsf{rk}(C)$. If $C \in \forall\mathsf{Tp}_n$, it amounts to (frz) and (cap°) for $\mathbf{JM}$. So suppose that $C \equiv A \Rightarrow B \in \mathcal{X}_\downarrow\backslash\forall\mathsf{Tp}_n$.

- (sn°) Assume $M \in \mathbf{CP}(A \Rightarrow B)$. By the IH ($\mathsf{csn}^\circ$) for A, we have $c \in \mathbf{CP}(A)$, so $Mc \in \mathbf{CP}(B)$ and thus $(Mc)^\circ \in \mathbf{JM}$ by the IH (sn°) for B. That is, $M^\circ \in \mathbf{JM}$.
- (csn°) Suppose that $\overline{T^\circ} \in \mathbf{JM}$. For any $N \in \mathbf{CP}(A)$, we have $N^\circ \in \mathbf{JM}$ by the IH (sn°) for A. Hence $c\overline{T}N \in \mathbf{CP}(B)$ by the IH (csn°) for B.

Finally (β°), (B) and (Ω_m) are verified by induction on $\mathsf{rk}(C)$, where C is the type of the term in conclusion. It is obvious if $C \in \forall\mathsf{Tp}_n$. Otherwise, suppose that the conclusion term is $M^{A\Rightarrow B}$. To prove $M \in \mathbf{CP}(A \Rightarrow B)$, it suffices to show $MN \in \mathbf{CP}(B)$ for any $N \in \mathbf{CP}(A)$. But it follows from the IH straightforwardly, since these rules are closed under term application. ◀

As an immediate consequence of (sn°), Lemma 15 and Lemma 2, we obtain:

▶ **Lemma 22** (Collapsing)**.** $\mathbf{CP} \subseteq \mathsf{SN}$.

We now proceed to the Embedding part. Since $\mathbf{CP}$ already satisfies (var), (con), (app) and (Abs), we only have to verify (abs) and (App_{n+1}). Let us begin with the latter. The basic idea is to use (Ω_{n+1}) as in Lemma 9, so we have to show that $\mathbf{CP}$ is closed under (Sub_{n+1}) as far as needed.

Consider a term substitution $\sigma = [N_1/x_1^{A_1}, \dots, N_k/x_k^{A_k}]$. We say that σ is a *cp-substitution* if $A_i \in \forall\mathsf{Tp}_n \cup \mathcal{X}_\downarrow$ and $N_i \in \mathbf{CP}(A_i)$ for every $1 \leq i \leq k$.

▶ **Lemma 23.** *Let* $B \in \mathcal{X}$ *and* σ *be a cp-substitution. Suppose that* $K^A \in \mathbf{JM}$ *satisfies:*
($\star$) $C[B/\alpha] \in \forall\mathsf{Tp}_n \cup \mathcal{X}_\downarrow$ *for any* $C \in \mathsf{type}(\mathsf{fv}(K)) \cup \{A\}$.
Then $K[B/\alpha]\sigma \in \mathbf{CP}$.

We are now ready to show:

▶ **Lemma 24.** $\mathbf{CP}$ *satisfies* (App_{n+1}) *for* $\forall\alpha.A \in \mathcal{X}_\downarrow$ *and* $B \in \mathcal{X}$.

Proof. Suppose that $M^{\forall\alpha.A} \in \mathbf{CP}$. We are going to use ($\Omega_{n+1}$). So let $K^A \in \mathbf{JM}_n \cap \mathsf{Ec}(\alpha)$. Any $C \in \mathsf{type}(\mathsf{fv}(K))$ does not contain α as free type variable, so that $C[B/\alpha] \equiv C \in \mathsf{Tp}_n$. Also, $A[B/\alpha] \in \mathcal{X}_\downarrow$. Hence K satisfies the condition of the previous lemma so that $K[B/\alpha] \in \mathbf{JM}_n$ follows. Therefore we obtain $MB \in \mathbf{CP}$ by (Ω_{n+1}). ◀

Given a lambda term $M \in \mathsf{Tm}_{n+1}$, let $\mathsf{subterm}(M)$ be the set of subterms of M and $\mathsf{subtype}(M)$ be defined by:

$$\mathsf{subtype}(M) := \{\mathsf{type}(N) : N \in \mathsf{subterm}(M)\} \cup \{B : NB \in \mathsf{subterm}(M)\}.$$

The next lemma, the *Basic Lemma* of logical relations, establishes Embedding. It can be proved by induction on the structure of M in a completely standard way.

▶ **Lemma 25.** *Suppose that* $\mathsf{subtype}(M) \subseteq \mathcal{X}$. *Then for any cp-substitution* σ, $M\sigma \in \mathbf{CP} = \mathbf{CP}(\mathcal{X})$.

In particular, any closed term $M \in \mathsf{Tm}_{n+1}$ belongs to $\mathbf{CP} = \mathbf{CP}(\mathcal{X})$ by letting $\mathcal{X} := \mathsf{subtype}(M)$. Hence in conjunction with Lemma 22, we conclude:

▶ **Theorem 26.** *System* $\mathbf{F}^p_{n+1}$ *admits strong normalization.*

4.2 Local formalization in $\mathbf{ID}_{n+1}$

In contrast to the JM predicates, the computability predicate $\mathbf{CP}$ does not fit the pattern of inductive definitions. Namely, it is not defined as the least fixed point of a monotone operator, due to the use of *non-monotone* operator $\Rightarrow$. Indeed, a naive formalization would require Π^1_1-comprehension, which is too high a price to pay.

On the other hand, $\mathbf{CP}$ is easily amenable to local formulation. For simplicity, let us write $m := n + 1$. If $\mathcal{X}$ is a finite set, $\mathbf{CP} = \mathbf{CP}(\mathcal{X})$ is definable from the JM predicate $\mathbf{JM}_{m-1}$ by a single formula so that it is definable in $\mathbf{ID}_m$. By formalizing the rest of argument, we obtain:

▶ **Theorem 27.** *Let* $\mathcal{X}$ *be a finite subset of* Tp_m. *Then* $\mathbf{ID}_m$ *proves that* M *is strongly normalizable for every closed term* $M \in \mathsf{Tm}_m$ *such that* $\mathsf{subtype}(M) \subseteq \mathcal{X}$.

A function $f : \mathbb{N} \longrightarrow \mathbb{N}$ is *representable* in System $\mathbf{F}^p_m$ if there is a lambda term M of type $\boldsymbol{N} \Rightarrow \boldsymbol{N}$ such that $M\mathsf{n} \to^* \mathsf{k}$ iff $f(n) = k$ for every $n, k \in \mathbb{N}$, where n is the Church numeral for n. By noting that $\mathsf{subtype}(\mathsf{n})$ is the same for any Church numeral n, we conclude from the previous theorem that $M\mathsf{n}$ normalizes to a Church numeral for every $n \in \mathbb{N}$, provably in $\mathbf{ID}_m$.

▶ **Theorem 28.** *Every representable function in* $\mathbf{F}^p_m$ *is provably total in* $\mathbf{ID}_m$.

▶ Remark. The above theorem, together with the converse direction, is already proved by Altenkirch and Coquand [5] for $n = 0$, and by Aehlig [2, 3] for an arbitrary $n \in \mathbb{N}$. The former proof uses a Heyting-valued computability predicate, while the latter consists of two steps: article [3] locally formalizes (weak) normalization (for terms of type $\boldsymbol{N}$) in a parameter-free system $\mathbf{HA}^2_{n+1,(1)}$ of second order Heyting arithmetic, and [2] gives a proof-theoretic reduction of the latter system to $\mathbf{ID}_n$. The reduction is done by encoding (recursive) infinitary derivations involving the Ω-rule into natural numbers, and then applying the computability argument. Though closely related, our proof is more direct in that it circumvents use of an intermediate system like $\mathbf{HA}^2_{n+1,(1)}$ and first-order encoding of infinitary derivations. More importantly, we prove *strong* normalization for *all* terms explicitly, in contrast to *weak* normalization for *specific* terms.

Acknowledgments. This work was supported by KAKENHI 16K16690 and KAKENHI 25330013.

References

1 A. Abel and T. Altenkirch. A predicative strong normalisation proof for a lambda-calculus with interleaving inductive types. In *Proceedings of TYPE'99*, pages 1–20, 1999.

2 K. Aehlig. Induction and inductive definitions in fragments of second order arithmetic. *Journal of Symbolic Logic*, 70:1087–1107, 2005.

3 K. Aehlig. Parameter-free polymorphic types. *Annals of Pure and Applied Logic*, 156:3–12, 2008.

4 R. Akiyoshi and G. Mints. An extension of the Omega-rule. *Archive for Mathematical logic*, 55(3):593–603, 2016.

5 T. Altenkirch and T. Coquand. A finitary subsystem of the polymorphic λ-calculus. *TLCA'01 Proceedings of the 5th international conference on Typed lambda calculi and applications*, pages 22–28. Springer, 2001.

6 Wilfried Buchholz. The $\Omega_{\mu+1}$-rule. In [8], pages 188–233, 1981.

7 W. Buchholz. Explaining the Gentzen-Takeuti reduction steps. *Archive for Mathematical Logic*, 40:255–272, 2001.

8 W. Buchholz, S. Feferman, W. Pohlers, and W. Sieg (editors). *Iterated Inductive Definitions and Subsystems of Analysis: Recent Proof-Theoretical Studies*, LNM 897, Springer, 1981.

9 W. Buchholz and K. Schütte. *Proof Theory of Impredicative Subsystems of Analysis*, Bibliopolis, 1988.

10 S. Feferman. What rests on what? The proof-theoretic analysis of mathematics. In *Proceedings of the 15th International Wittgenstein Symposium*, pages 141–171, 1993.

11 J.-Y. Girard. Une Extension de l'Interpretation de Gödel á l'Analyse, et son Application á l'Élimitation des Coupures dans l'Analyse et la Théorie des Types *Proceedings of the Second Scandinavian Logic Symposium.* Amsterdam, pages 63–92, 1971.

12 F. Joachimski and R. Matthes. Short proofs of normalization for the simply-typed lambda-calculus, permutative conversions and Gödel's T. *Archive for Mathematical Logic*, 42:59–87, 2003.

13 W. Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967.

14 W. Tait. Gödel's unpublished papers on foundations of mathematics. *Philosophia Mathematica*, 3(9):87–126, 2001.

15 G. Takeuti. Consistency proofs of subsystems of classical analysis. *The Annals of Mathematics*, 86(2):299–348, 1967.

16 G. Takeuti. *Proof Theory (2nd. ed.)*, North-Holland, 1987.

17 A.S. Troelstra. Introductory note to *1941. In *Kurt Gödel Collected Works, volume 3*, Oxford, pages 186–189, 1995.

18 D.T. van Daalen. *The Language Theory of Automath.* PhD thesis, Technische Universiteit Eindhoven, 1980.

19 F. van Raamsdonk and P. Severi. On Normalisation. *Technical Report CS-R9545*, CWI, 1995.

20 F. van Raamsdonk, P. Severi, M.H. Sorensen and H. Xi. Perpetual reductions in λ-calculus. *Information and Computation*, 149(2): 173–225, 1999.

Normalisation by Evaluation for Dependent Types[*]

Thorsten Altenkirch[1] and Ambrus Kaposi[2]

1 School for Computer Science, University of Nottingham, Nottingham, United Kingdom
txa@cs.nott.ac.uk

2 Department of Programming Languages and Compilers, Eötvös Loránd University, Budapest, Hungary
akaposi@inf.elte.hu

Abstract

We develop normalisation by evaluation (NBE) for dependent types based on presheaf categories. Our construction is formulated using internal type theory using quotient inductive types. We use a typed presentation hence there are no preterms or realizers in our construction. NBE for simple types is using a logical relation between the syntax and the presheaf interpretation. In our construction, we merge the presheaf interpretation and the logical relation into a proof-relevant logical predicate. We have formalized parts of the construction in Agda.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases normalisation by evaluation, dependent types, internal type theory, logical relations, Agda

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.6

1 Introduction

1.1 Specifying normalisation

Normalisation can be given the following specification.

We denote the type of well typed terms of type A in context Γ by $\mathsf{Tm}\,\Gamma\,A$. This type is defined as a quotient inductive inductive type (QIIT, see [10]): in addition to normal constructors for terms such as lam and app, it also has equality constructors e.g. expressing the β computation rule for functions. An equality $t \equiv_{\mathsf{Tm}\,\Gamma\,A} t'$ expresses that t and t' are convertible.

The type of normal forms is denoted $\mathsf{Nf}\,\Gamma\,A$ and there is an embedding from it to terms $\ulcorner - \urcorner : \mathsf{Nf}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A$. Normal forms are defined as a usual inductive type, decidability of equality is straightforward.

Normalisation is given by a function norm which takes a term to a normal form. It needs to be an isomorphism:

$$\text{completeness} \circlearrowleft \qquad \mathsf{norm} \downarrow \frac{\mathsf{Tm}\,\Gamma\,A}{\mathsf{Nf}\,\Gamma\,A} \uparrow \ulcorner - \urcorner \qquad \curvearrowright \text{stability}$$

If we normalise a term, we obtain a term which is convertible to it: $t \equiv \ulcorner \mathsf{norm}\,t \urcorner$. This is called completeness. The other direction is called stability: $n \equiv \mathsf{norm}\,\ulcorner n \urcorner$. It expresses that there is no redundancy in the type of normal forms. This property makes it possible to establish properties of the syntax by induction on normal forms.

[*] Supported by EPSRC grant EP/M016951/1 and USAF grant FA9550-16-1-0029.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 6; pp. 6:1–6:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$$\begin{tikzcd}
\mathsf{NE}_\Delta \arrow[r, "\mathsf{u}_\Delta"] \arrow[dr, "\ulcorner\_\urcorner"'] & \Sigma\,(\mathsf{TM}_\Delta \times [\![\Delta]\!])\,\mathsf{R}_\Delta \arrow[r, "\mathsf{q}_\Delta"] \arrow[d, "\mathsf{proj}"] & \mathsf{NF}_\Delta \arrow[dl, "\ulcorner\_\urcorner"] \\
& \mathsf{TM}_\Delta &
\end{tikzcd}$$

Figure 1 The type of quote and unquote for a context Δ in NBE for simple types.

Soundness, that is, if $t \equiv t'$ then $\mathsf{norm}\,t \equiv \mathsf{norm}\,t'$ is given by congruence of equality. The elimination rule for the QIIT of the syntax ensures that every function defined from the syntax respects the equality constructors.

1.2 NBE for simple type theory

Normalisation by evaluation (NBE) is one way to implement this specification. In this subsection, we summarize the approach of [6]. NBE works by evaluating the syntax in a presheaf model over the category of renamings REN and with normal forms as interpretation of the base type. The objects in REN are contexts and morphisms are lists of variables. Note that for any context Γ one can define the presheaves of terms, neutral terms (the subset of normal forms where an eliminator is applied to a variable) and normal forms. The action on objects is just returning substitutions, lists of neutral terms and lists of normal forms, respectively.

$$\begin{array}{lll}
\mathsf{TM}_\Delta : \mathsf{PSh}\,\mathsf{REN} & \mathsf{NE}_\Delta : \mathsf{PSh}\,\mathsf{REN} & \mathsf{NF}_\Delta : \mathsf{PSh}\,\mathsf{REN} \\
\mathsf{TM}_\Delta\,\Gamma := \mathsf{Tms}\,\Gamma\,\Delta & \mathsf{NE}_\Delta\,\Gamma := \mathsf{Nes}\,\Gamma\,\Delta & \mathsf{NF}_\Delta\,\Gamma := \mathsf{Nfs}\,\Gamma\,\Delta
\end{array}$$

To normalise a substitution with codomain Δ, one defines two natural transformations unquote u_Δ and quote q_Δ by induction on the structure of contexts and types such that the diagram in figure 1 commutes. $[\![\Delta]\!]$ denotes the interpretation of Δ in the presheaf model and R_Δ denotes the logical relation at context Δ between TM_Δ and $[\![\Delta]\!]$. The logical relation is equality at the base type.

Now a substitution σ can be normalised by quote: it needs the substitution itself, the interpretation $[\![\sigma]\!]$ and a proof that they are related. This is given by the fundamental theorem of the logical relation denoted by R_σ which also needs two related elements: these are given by unquoting the identity renaming (which is neutral).

$$\mathsf{norm}_\Delta\,(\sigma : \mathsf{TM}_\Delta\,\Gamma) : \mathsf{NF}_\Delta\,\Gamma := \mathsf{q}_{\Delta\,\Gamma}\,(\sigma, [\![\sigma]\!], \mathsf{R}_\sigma\,(\mathsf{u}_{\Gamma\,\Gamma}\,\mathsf{id}_\Gamma))$$

Completeness is given by commutativity of the right hand triangle. Stability can be proven by mutual induction on terms and normal forms.

A nice property of this approach is that the part of unquote and quote which gives $[\![\Delta]\!]$ can be defined separately from the part which gives relatedness, hence the normalisation function can be defined independently from the proof that it is complete.

1.3 NBE for type theory

In the case of simple type theory, types are closed, so they act like contexts. Quote at a type A is just a natural transformation.

$$\mathsf{q}_A : \Sigma\,(\mathsf{TM}_A \times [\![A]\!])\,\mathsf{R}_A \mathrel{\dot{\to}} \mathsf{NF}_A$$

$$\begin{tikzcd}
\mathsf{NE}_\Delta \arrow[r, "\mathsf{u}_\Delta"] \arrow[dr, "\ulcorner\_\urcorner"'] & \Sigma\,\mathsf{TM}_\Delta\,\llbracket\Delta\rrbracket \arrow[r, "\mathsf{q}_\Delta"] \arrow[d, "\mathsf{proj}"] & \mathsf{NF}_\Delta \arrow[dl, "\ulcorner\_\urcorner"] \\
& \mathsf{TM}_\Delta &
\end{tikzcd}$$

Figure 2 The type of quote and unquote for a context Δ in our proof.

In the case of (dependent) type theory, types depend on contexts, so $\mathsf{TM}_{\Gamma\vdash A}$ becomes a family of presheaves over TM_Γ, $\llbracket\Gamma\vdash A\rrbracket$ is a family over $\llbracket\Gamma\rrbracket$ and $\mathsf{R}_{\Gamma\vdash A}$ depends on R_Γ (and a term of that type and the interpretation of a term of that type).

$$\begin{aligned}
&\mathsf{TM}_{\Gamma\vdash A}, \mathsf{NE}_{\Gamma\vdash A}, \mathsf{NF}_{\Gamma\vdash A} : \mathsf{FamPSh}\,\mathsf{TM}_\Gamma\\
&\llbracket\Gamma\vdash A\rrbracket : \mathsf{FamPSh}\,\llbracket\Gamma\rrbracket\\
&\mathsf{R}_{\Gamma\vdash A} : \mathsf{FamPSh}\Big(\Sigma\,\big(\Sigma\,(\mathsf{TM}_\Gamma \times \llbracket\Gamma\rrbracket)\,\mathsf{R}_\Gamma\big)\,\big(\mathsf{TM}_{\Gamma\vdash A} \times \llbracket\Gamma\vdash A\rrbracket\big)\Big)
\end{aligned}$$

We can try to define quote and unquote for this type as a family of natural transformations. The type of quote and unquote omitting the naturality conditions would be the following. These types encode the commutativity of the triangles as well.

$$\begin{aligned}
&\mathsf{q}_{(\Gamma\vdash A)\,\Psi} : (p : \mathsf{R}_{\Gamma\,\Psi}\,\rho\,\alpha)(t : \mathsf{TM}_A\,\rho)(v : \llbracket A\rrbracket\,\alpha) \to \mathsf{R}_A\,p\,t\,v \to \Sigma(n : \mathsf{NF}_A\,\rho).t \equiv \ulcorner n\urcorner\\
&\mathsf{u}_{(\Gamma\vdash A)\,\Psi} : (p : \mathsf{R}_{\Gamma\,\Psi}\,\rho\,\alpha)(n : \mathsf{NE}_A\,\rho) \to \Sigma(v : \llbracket A\rrbracket\,\alpha).\mathsf{R}_A\,p\,\ulcorner n\urcorner\,v
\end{aligned}$$

However there seems to be no way to define quote and unquote this way because quote does not preserve the logical relation. The problem is that when defining unquote at Π we need to define a semantic function which works for arbitrary inputs, not only those which are related to a term. It seems that we need to restrict the presheaf model to only contain such functions.

We solve this problem by replacing the presheaf and the logical relation by a proof relevant logical predicate. We denote the logical predicate at a context Δ by $\llbracket\Delta\rrbracket$. We define normalisation following the diagram in figure 2.

In the presheaf model, the interpretation of the base type was normal forms of the base type, and the logical relation at the base type was equality of the term and the normal form. In our case, the logical predicate at the base type will say that there exists a normal form which is equal to the term.

1.4 Structure of the proof and the paper

In this section, we give a high level sketch of the proof. Sections 3, 4, 6 are fully formalised in Agda, sections 5, 7 and 8 are partially formalised [9].

In section 2 we briefly summarize the metatheory we are working in.

In section 3 we define the syntax for type theory as a quotient inductive inductive type (QIIT) [10]. The arguments of the eliminator for the QIIT form a model of type theory.

In section 4 we define the category of renamings REN: objects are contexts and morphisms are renamings (lists of variables).

In section 5 we define the proof-relevant Kripke logical predicate interpretation of the syntax. The interpretation has REN as the base category and two parameters for the interpretations of U and El. This interpretation can be seen as a dependent version of the presheaf model of type theory. E.g. a context in the presheaf model is interpreted as a presheaf. Now it is a family of presheaves dependent on a substitution into that context. The interpretations of base types can depend on the actual elements of the base types. The interpretation of substitutions and terms are the fundamental theorems.

In section 6 we define neutral terms and normal forms together with their renamings and embeddings into the syntax ($\ulcorner - \urcorner$). With the help of these, we define the interpretations of U and El. The interpretation of U at a term of type U will be a neutral term of type U which is equal to the term. Now we can interpret any term of the syntax in the logical predicate interpretation. We will denote the interpretation of a term t by $[\![t]\!]$.

In section 7 we mutually define the natural transformations quote and unquote. We define them by induction on contexts and types as shown in figure 2. Quote takes a term and a semantic value at that term into a normal term and a proof that the normal term is equal to it. Unquote takes a neutral term into a semantic value at the neutral term.

Finally, in section 8, we put together the pieces by defining the normalisation function and showing that it is complete and stable. Normalisation and completeness are given by interpreting the term in the logical predicate model at the identity semantic element and then quoting. Stability is proved by mutual induction on neutral terms and normal forms.

1.5 Related work

Normalisation by evaluation was first formulated by Schwichtenberg and Berger [11], subsequently a categorical account using presheaf categories was given [6] and this approach was extended to System F [7, 8] and coproducts [5]. The present work can be seen as a continuation of this line of research.

The term normalisation by evaluation is also more generally used to describe semantic based normalisation functions. E.g. Danvy is using semantic normalisation for partial evaluation [14]. Normalisation by evaluation using untyped realizers has been applied to dependent types by Abel et al [3, 1, 2]. Danielsson [13] has formalized NBE for dependent types but he doesn't prove soundness of normalisation.

2 Metatheory and notation

We are working in intensional Martin-Löf Type Theory with postulated extensionality principles using Agda as a vehicle [17, 4]. We extend Agda with quotient inductive inductive types (QIITs, see section 6 of [10]) using axioms. When defining an inductive type A, we first declare the type by $\mathsf{data}\, A : S$ where S is the sort, then we list the constructors. For inductive inductive types we first declare all the types, then following a second data keyword we list the constructors. We also postulate functional extensionality which is a consequence of having an interval QIIT anyway. We assume K, that is, we work in a strict type theory.

We follow Agda's convention of denoting the universe of types by Set, we write function types as $(x : A) \to B$ or $\forall x.B$, implicit arguments are written in curly braces $\{x : A\} \to B$ and can be omitted or given in lower index. If some arguments are omitted, we assume universal quantification, e.g. $(y : B\, x) \to C$ means $\forall\, x.(y : B\, x) \to C$ if x is not given in the context. We write $\Sigma(x : A).B$ for Σ types. We overload names e.g. the action on objects and morphisms of a functor is denoted by the same symbol.

The identity type is denoted $- \equiv -$ and its constructor is refl. Transport of a term $a : P\, x$ along an equality $p : x \equiv y$ is denoted ${}_{p*}a : P\, y$. We denote $({}_{p*}a) \equiv b$ by $a \equiv^p b$. We write ap for congruence, that is $\mathsf{ap}\, f\, p : f\, x \equiv f\, y$ if $p : x \equiv y$. For readability, we will omit transports most of the time (starting from section 5). This makes some terms non-well typed, e.g. we might write $f\,a$ where $f : A \to B$ and $a : A'$ but in this case there is an equality in scope which justifies $A \equiv A'$.

Sometimes we use Coq-style definitions: we write $\mathsf{d}\,(x : A) : B := t$ for defining d of type $(x : A) \to B$ by $\lambda x.t$. We also use Agda-style pattern matching definitions.

3 Object theory

The object theory is the same[1] as in [10], we present it as a quotient inductive inductive type (QIIT). A QIIT is presented by first declaring the types that we define mutually, and then listing all the constructors.

The syntax constituting of contexts, types, substitutions and terms is declared as follows.

$$\begin{aligned}
&\mathsf{data\,Con} &&: \mathsf{Set}\\
&\mathsf{data\,Ty} &&: \mathsf{Con} \to \mathsf{Set}\\
&\mathsf{data\,Tms} &&: \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}\\
&\mathsf{data\,Tm} &&: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set}
\end{aligned}$$

We use the convention of naming contexts Γ, Δ, Θ, types A, B, terms t, u, substitutions σ, ν, δ.

We define a basic type theory with an uninterpreted base type U, a family over this type El and dependent function space Π with constructor lam and eliminator app. Our type theory is given as an explicit substitution calculus, hence the QIIT needs constructors $-[-]$ for substituted types and terms. The constructors of the QIIT can be summarized as follows.

- Substitutions form a category with a terminal object. This includes the categorical substitution laws for types $[\mathsf{id}]$ and $[][]$.
- Substitution laws for types $\mathsf{U}[]$, $\mathsf{El}[]$, $\Pi[]$.
- The laws of comprehension which state that we have the natural isomorphism

  $$\pi_1\beta, \pi_2\beta \circlearrowleft \qquad -,- \downarrow \frac{\sigma : \mathsf{Tms}\,\Gamma\,\Delta \qquad \mathsf{Tm}\,\Gamma\,A[\sigma]}{\mathsf{Tms}\,\Gamma\,(\Delta, A)} \uparrow \pi_1, \pi_2 \qquad \curvearrowright \pi\eta$$

  where naturality[2] is given by $,\circ$.
- The laws for function space which are given by the natural isomorphism

  $$\Pi\beta \circlearrowleft \qquad \mathsf{lam} \downarrow \frac{\mathsf{Tm}\,(\Gamma, A)\,B}{\mathsf{Tm}\,\Gamma\,(\Pi\,A\,B)} \uparrow \mathsf{app} \qquad \curvearrowright \Pi\eta$$

  where naturality is given by $\mathsf{lam}[]$.

[1] Steven Schäfer pointed us to [18] which shows that in the presentation [10] the equalities $[\mathsf{id}]\mathsf{t}$ and $[][]\mathsf{t}$ (identity and associativity laws of term substitution) can be derived from the others. This is why we omitted these equalities from this presentation and the formal development.

[2] If one direction of an isomorphism is natural, so is the other. This is why it is enough to state naturality for $-,-$ and not for π_1, π_2.

We list the point constructors in the left column and the equality constructors in the right.

$$\begin{array}{ll}
\multicolumn{2}{l}{\mathsf{data}} \\
\cdot & : \mathsf{Con} \\
-,- & : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con} \\
-[-] & : \mathsf{Ty}\,\Delta \to \mathsf{Tms}\,\Gamma\,\Delta \to \mathsf{Ty}\,\Gamma \\
\mathsf{U} & : \mathsf{Ty}\,\Gamma \\
\mathsf{El} & : \mathsf{Tm}\,\Gamma\,\mathsf{U} \to \mathsf{Ty}\,\Gamma \\
\Pi & : (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma, A) \to \mathsf{Ty}\,\Gamma \\
\mathsf{id} & : \mathsf{Tms}\,\Gamma\,\Gamma \\
- \circ - & : \mathsf{Tms}\,\Theta\,\Delta \to \mathsf{Tms}\,\Gamma\,\Theta \to \mathsf{Tms}\,\Gamma\,\Delta \\
\epsilon & : \mathsf{Tms}\,\Gamma\,\cdot \\
-,- & : (\sigma : \mathsf{Tms}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,A[\sigma] \to \mathsf{Tms}\,\Gamma\,(\Delta, A) \\
\pi_1 & : \mathsf{Tms}\,\Gamma\,(\Delta, A) \to \mathsf{Tms}\,\Gamma\,\Delta \\
-[-] & : \mathsf{Tm}\,\Delta\,A \to (\sigma : \mathsf{Tms}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,A[\sigma] \\
\pi_2 & : (\sigma : \mathsf{Tms}\,\Gamma\,(\Delta, A)) \to \mathsf{Tm}\,\Gamma\,A[\pi_1\,\sigma] \\
\mathsf{lam} & : \mathsf{Tm}\,(\Gamma, A)\,B \to \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \\
\mathsf{app} & : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to \mathsf{Tm}\,(\Gamma, A)\,B
\end{array}$$

$$\begin{array}{ll}
\multicolumn{2}{l}{\mathsf{data}} \\
[\mathsf{id}] & : A[\mathsf{id}] \equiv A \\
[][] & : A[\sigma][\nu] \equiv A[\sigma \circ \nu] \\
\mathsf{U}[] & : \mathsf{U}[\sigma] \equiv \mathsf{U} \\
\mathsf{El}[] & : (\mathsf{El}\,\hat{A})[\sigma] \equiv \mathsf{El}\,({}_{\mathsf{U}[]*}\hat{A}[\sigma]) \\
\Pi[] & : (\Pi\,A\,B)[\sigma] \equiv \Pi\,(A[\sigma])\,(B[\sigma^A]) \\
\mathsf{id}\circ & : \mathsf{id} \circ \sigma \equiv \sigma \\
\circ\mathsf{id} & : \sigma \circ \mathsf{id} \equiv \sigma \\
\circ\circ & : (\sigma \circ \nu) \circ \delta \equiv \sigma \circ (\nu \circ \delta) \\
\epsilon\eta & : \{\sigma : \mathsf{Tms}\,\Gamma\,\cdot\} \to \sigma \equiv \epsilon \\
\pi_1\beta & : \pi_1\,(\sigma, t) \equiv \sigma \\
\pi\eta & : (\pi_1\,\sigma, \pi_2\,\sigma) \equiv \sigma \\
,\circ & : (\sigma, t) \circ \nu \equiv (\sigma \circ \nu), ({}_{[][]*}t[\nu]) \\
\pi_2\beta & : \pi_2\,(\sigma, t) \equiv^{\pi_1\beta} t \\
\Pi\beta & : \mathsf{app}\,(\mathsf{lam}\,t) \equiv t \\
\Pi\eta & : \mathsf{lam}\,(\mathsf{app}\,t) \equiv t \\
\mathsf{lam}[] & : (\mathsf{lam}\,t)[\sigma] \equiv^{\Pi[]} \mathsf{lam}\,(t[\sigma^A])
\end{array}$$

Note that the equality $\pi_2\beta$ lives over $\pi_1\beta$. Also, we had to use transport to typecheck $\mathsf{El}[]$ and $,\circ$. We used lifting of a substitution in the types of $\Pi[]$ and $\mathsf{lam}[]$. It is defined as follows.

$$(\sigma : \mathsf{Tms}\,\Gamma\,\Delta)^A : \mathsf{Tms}\,(\Gamma, A[\sigma])\,(\Delta, A) := (\sigma \circ \pi_1\,\mathsf{id}), ({}_{[][]*}\pi_2\,\mathsf{id})$$

We use the categorical app operator but the usual one $(-\$-)$ can also be derived.

$$\begin{array}{lll}
< (u : \mathsf{Tm}\,\Gamma\,A) > & : \mathsf{Tms}\,\Gamma\,(\Gamma, A) & := \mathsf{id}, {}_{[\mathsf{id}]^{-1}*}u \\
(t : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B))\$(u : \mathsf{Tm}\,\Gamma\,A) & : B[< u >] & := (\mathsf{app}\,t)[< u >]
\end{array}$$

When we define a function from the above syntax, we need to use the eliminator. The eliminator has 4 motives corresponding to what Con, Ty, Tms and Tm get mapped to and one method for each constructor including the equality constructors. The methods for point constructors are the elements of the motives to which the constructor is mapped. The methods for the equality constructors demonstrate soundness, that is, the semantic constructions respect the syntactic equalities. The eliminator comes in two different flavours: the non-dependent and dependent version. In our constructions we use the dependent version. The motives and methods for the non-dependent eliminator (recursor) collected together form a model of type theory, they are basically the same[3] as Dybjer's Categories with Families [15].

As an example we list the motives and a few methods of the dependent eliminator. An algorithm for deriving them from the constructors is given in [10]. As names we use the names of the constructors followed by an upper index M.

[3] Dybjer uses the usual application operator, we use the categorical one, the projections π_1, π_2 are defined differently and Dybjer lists some equations derivable from the others, we omit these. However all the operators and the laws are inter-derivable.

$$
\begin{aligned}
&\mathsf{Con}^{\mathsf{M}} &&: \mathsf{Con} \to \mathsf{Set}\\
&\mathsf{Ty}^{\mathsf{M}} &&: (\mathsf{Con}^{\mathsf{M}}\,\Gamma) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set}\\
&\mathsf{Tms}^{\mathsf{M}} &&: (\mathsf{Con}^{\mathsf{M}}\,\Gamma) \to (\mathsf{Con}^{\mathsf{M}}\,\Delta) \to \mathsf{Tms}\,\Gamma\,\Delta \to \mathsf{Set}\\
&\mathsf{Ty}^{\mathsf{M}} &&: (\Gamma^{\mathsf{M}} : \mathsf{Con}^{\mathsf{M}}\,\Gamma) \to \mathsf{Ty}^{\mathsf{M}}\,\Gamma^{\mathsf{M}}\,A \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Set}\\
&\mathsf{id}^{\mathsf{M}} &&: \mathsf{Tms}^{\mathsf{M}}\,\Gamma^{\mathsf{M}}\,\Gamma^{\mathsf{M}}\,\mathsf{id}\\
&- \circ^{\mathsf{M}} - &&: \mathsf{Tms}^{\mathsf{M}}\,\Theta^{\mathsf{M}}\,\Delta^{\mathsf{M}}\,\sigma \to \mathsf{Tms}^{\mathsf{M}}\,\Gamma^{\mathsf{M}}\,\Theta^{\mathsf{M}}\,\nu \to \mathsf{Tms}^{\mathsf{M}}\,\Gamma^{\mathsf{M}}\,\Delta^{\mathsf{M}}\,(\sigma \circ \nu)\\
&\circ\mathsf{id}^{\mathsf{M}} &&: \sigma^{\mathsf{M}} \circ^{\mathsf{M}} \mathsf{id}^{\mathsf{M}} \equiv^{\circ\mathsf{id}} \sigma^{\mathsf{M}}\\
&\pi_2\beta^{\mathsf{M}} &&: \pi_2^{\mathsf{M}}\,(\rho^{\mathsf{M}} ,^{\mathsf{M}} t^{\mathsf{M}}) \equiv^{\pi_1\beta^{\mathsf{M}},\pi_2\beta} t^{\mathsf{M}}
\end{aligned}
$$

Note that the method equality $\circ\mathsf{id}^{\mathsf{M}}$ lives over the constructor $\circ\mathsf{id}$ while the method equality $\pi_2\beta^{\mathsf{M}}$ lives both over the method equality $\pi_1\beta^{\mathsf{M}}$ and the equality constructor $\pi_2\beta$.

4 The category of renamings

In this section we define the category of renamings REN. Objects in this category are contexts, morphisms are renamings (Vars): lists of de Bruijn variables.

We define the types of variables Var and renamings Vars together with their embeddings into substitutions. This is an inductive-recursive definition as $\ulcorner - \urcorner$ for renamings needs to be defined mutually with renamings.

$$
\begin{aligned}
&\mathsf{data}\,\mathsf{Var} &&: (\Psi : \mathsf{Con}) \to \mathsf{Ty}\,\Psi \to \mathsf{Set}\\
&\quad \mathsf{vze} &&: \mathsf{Var}\,(\Psi, A)\,(A[\pi_1\,\mathsf{id}])\\
&\quad \mathsf{vsu} &&: \mathsf{Var}\,\Psi\,A \to \mathsf{Var}\,(\Psi, B)\,(A[\pi_1\,\mathsf{id}])\\
&\ulcorner - \urcorner &&: \mathsf{Vars}\,\Omega\,\Psi \to \mathsf{Tms}\,\Omega\,\Psi\\
&\mathsf{data}\,\mathsf{Vars} &&: \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}\\
&\quad \epsilon &&: \mathsf{Vars}\,\Psi\,\cdot\\
&\quad -,- &&: (\beta : \mathsf{Vars}\,\Omega\,\Psi) \to \mathsf{Var}\,\Omega\,A[\ulcorner\beta\urcorner] \to \mathsf{Vars}\,\Omega\,(\Psi, A)\\
&\ulcorner - \urcorner &&: \mathsf{Var}\,\Psi\,A \to \mathsf{Tm}\,\Psi\,A\\
&\ulcorner \mathsf{vze} \urcorner &&:= \pi_2\,\mathsf{id}\\
&\ulcorner \mathsf{vsu}\,x \urcorner &&:= \ulcorner x \urcorner[\pi_1\,\mathsf{id}]\\
&\ulcorner \epsilon \urcorner &&:= \epsilon\\
&\ulcorner \beta, x \urcorner &&:= \ulcorner \beta \urcorner, \ulcorner x \urcorner
\end{aligned}
$$

Variables are typed de Bruijn indices. vze projects out the last element of the context, vsu extends the context, and the type $A : \mathsf{Ty}\,\Psi$ needs to be weakened in both cases because we need to interpret it in Ψ extended by another type. Renamings are lists of variables with the appropriate types. Embedding of variables into terms uses the projections and the identity substitution, and embedding renamings is pointwise.

We use the names Ψ, Ω, Ξ for objects of REN, x, y for variables, β, γ for renamings.

We need identity and composition of renamings for the categorical structure. To define them, we also need weakening and renaming of variables together with laws relating their embeddings to terms. We only list the types as the definitions are straightforward inductions.

$$\begin{array}{lll}
\mathsf{wk} & : \mathsf{Vars}\,\Omega\,\Psi \to \mathsf{Vars}\,(\Omega, A)\,\Psi & \ulcorner\mathsf{wk}\urcorner : \ulcorner\beta\urcorner \circ \pi_1\mathsf{id} \equiv \ulcorner\mathsf{wk}\,\beta\urcorner \\
\mathsf{id} & : \mathsf{Vars}\,\Psi\,\Psi & \ulcorner\mathsf{id}\urcorner\ \ : \ulcorner\mathsf{id}\urcorner \equiv \mathsf{id} \\
- \circ - & : \mathsf{Vars}\,\Xi\,\Psi \to \mathsf{Vars}\,\Omega\,\Xi \to \mathsf{Vars}\,\Omega\,\Psi & \ulcorner\circ\urcorner\ \ : \ulcorner\beta\urcorner \circ \ulcorner\gamma\urcorner \equiv \ulcorner\beta \circ \gamma\urcorner \\
-[-] & : \mathsf{Var}\,\Psi\,A \to (\beta : \mathsf{Vars}\,\Omega\,\Psi) \to \mathsf{Var}\,\Omega\,A[\ulcorner\beta\urcorner] & \ulcorner[]\urcorner\ \ : \ulcorner x\urcorner[\ulcorner\beta\urcorner] \equiv \ulcorner x[\beta]\urcorner
\end{array}$$

Renamings form a category, we omit the statement and proofs of the categorical laws.

5 The logical predicate interpretation

In this section, after defining a few categorical notions, we define the proof-relevant Kripke logical predicate interpretation of the type theory given in section 3. It can also be seen as a dependent version of the presheaf model of type theory [16].

A contravariant presheaf over a category $\mathcal{C}$ is denoted $\Gamma : \mathsf{PSh}\,\mathcal{C}$. It is given by the following data: given $I : |\mathcal{C}|$, a set $\Gamma\,I$, and given $f : \mathcal{C}(J, I)$ a function $\Gamma\,f : \Gamma\,I \to \Gamma\,J$. Moreover, we have $\mathsf{idP}\,\Gamma : \Gamma\,\mathsf{id}\,\alpha \equiv \alpha$ and $\mathsf{compP}\,\Gamma : \Gamma\,(f \circ g)\,\alpha \equiv \Gamma\,g\,(\Gamma\,f\,\alpha)$ for $\alpha : \Gamma\,I$, $f : \mathcal{C}(J, I)$, $g : \mathcal{C}(K, J)$.

Given $\Gamma : \mathsf{PSh}\,\mathcal{C}$, a family of presheaves over Γ is denoted $A : \mathsf{FamPSh}\,\Gamma$. It is given by the following data [4] : given $\alpha : \Gamma\,I$, a set $A_I\,\alpha$ and given $f : \mathcal{C}(J, I)$, a function $A\,f : A_I\,\alpha \to A_J\,(\Gamma\,f\,\alpha)$. In addition, we have the functor laws $\mathsf{idF}\,A : A\,\mathsf{id}\,v \equiv^{\mathsf{idP}} v$ and $\mathsf{compF}\,A : A\,(f \circ g)\,v \equiv^{\mathsf{compP}} A\,g\,(A\,f\,v)$ for $\alpha : \Gamma\,I$, $v : A\,\alpha$, $f : \mathcal{C}(J, I)$, $g : \mathcal{C}(K, J)$.

A natural transformation between presheaves Γ and Δ is denoted $\sigma : \Gamma \mathrel{\dot{\to}} \Delta$. It is given by a function $\sigma : \{I : |\mathcal{C}|\} \to \Gamma\,I \to \Delta\,I$ together with the condition $\mathsf{natn}\,\sigma : \Delta\,f\,(\sigma_I\,\alpha) \equiv \sigma_J\,(\Gamma\,f\,\alpha)$ for $\alpha : \Gamma\,I$, $f : \mathcal{C}(J, I)$.

A section[5] from a presheaf Γ to a family of presheaves A over Γ is denoted $t : \Gamma \xrightarrow{\mathsf{s}} A$. It is given by a function $t : \{I : |\mathcal{C}|\} \to (\alpha : \Gamma\,I) \to A_I\,\alpha$ together with the naturality condition $\mathsf{natS}\,t\,\alpha\,f : A\,f\,(t\,\alpha) \equiv t\,(\Gamma\,f\,\alpha)$ for $f : \mathcal{C}(J, I)$.

Given $\Gamma : \mathsf{PSh}\,\mathcal{C}$ and $A : \mathsf{FamPSh}\,\Gamma$ we can define $\Sigma\,\Gamma\,A : \mathsf{PSh}\,\mathcal{C}$ by $(\Sigma\,\Gamma\,A)\,I := \Sigma(\alpha : \Gamma\,I).A_I\,\alpha$ and $(\Sigma\,\Gamma\,A)\,f\,(\alpha, a) := (\Gamma\,f\,\alpha, A\,f\,a)$.

Given $\sigma : \Gamma \mathrel{\dot{\to}} \Delta$ and $A : \mathsf{FamPSh}\,\Delta$, we define $A[\sigma] : \mathsf{FamPSh}\,\Gamma$ by $A[\sigma]_I\,\alpha := A_I\,(\sigma_I\,\alpha)$ and $A[\sigma]\,f\,a := {}_{\mathsf{natn}\,\sigma *}(A\,f\,a)$ for $\alpha : \Gamma\,I$, $a : A[\sigma]\,\alpha$ and $f : \mathcal{C}(J, I)$.

The weakening natural transformation $\mathsf{wk} : \Sigma\,\Gamma\,A \mathrel{\dot{\to}} \Gamma$ is defined by $\mathsf{wk}_I\,(\alpha, a) := \alpha$.

Lifting of a section $t : \Gamma \xrightarrow{\mathsf{s}} A$ by a family of presheaves $B : \mathsf{FamPSh}\,\Gamma$ is a natural transformation $t^B : \Sigma\,\Gamma\,B \mathrel{\dot{\to}} \Sigma\,(\Sigma\,(\Gamma\,A))\,B[\mathsf{wk}]$. It is defined as ${t^B}_I\,(\alpha, b) := (\alpha, t_I\,\alpha, b)$.

To define the logical predicate interpretation of the syntax, we need to give the motives and methods for the eliminator. We will denote the interpretation of a syntactic construct t by $[\![t]\!]$. The following table gives the motives of the eliminator.

$$\begin{array}{lllll}
\Gamma : \mathsf{Con} & \mathsf{TM}_\Gamma = \mathsf{Tms} - \Gamma & : \mathsf{PSh}\,\mathsf{REN} & [\![\Gamma]\!] & : \mathsf{FamPSh}\,\mathsf{TM}_\Gamma \\
A : \mathsf{Ty}\,\Gamma & \mathsf{TM}_A = \mathsf{Tm} - A[-] & : \mathsf{FamPSh}\,\mathsf{TM}_\Gamma & [\![A]\!] & : \mathsf{FamPSh}\left(\Sigma\,(\Sigma\,(\mathsf{TM}_\Gamma\,\mathsf{TM}_A))\,[\![\Gamma]\!][\mathsf{wk}]\right) \\
\sigma : \mathsf{Tms}\,\Gamma\,\Delta & \mathsf{TM}_\sigma = (\sigma \circ -) & : \mathsf{TM}_\Gamma \mathrel{\dot{\to}} \mathsf{TM}_\Delta & [\![\sigma]\!] & : \Sigma\,\mathsf{TM}_\Gamma\,[\![\Gamma]\!] \xrightarrow{\mathsf{s}} [\![\Delta]\!][\mathsf{TM}_\sigma][\mathsf{wk}] \\
t : \mathsf{Tm}\,\Gamma\,A & \mathsf{TM}_t = t[-] & : \mathsf{TM}_\Gamma \xrightarrow{\mathsf{s}} \mathsf{TM}_A & [\![t]\!] & : \Sigma\,\mathsf{TM}_\Gamma\,[\![\Gamma]\!] \xrightarrow{\mathsf{s}} [\![A]\!][{\mathsf{TM}_t}^{[\![\Gamma]\!]}]
\end{array}$$

[4] Indeed, this is equivalent to a presheaf over the category of elements $\int \Gamma$.

[5] $t : \Gamma \xrightarrow{\mathsf{s}} A$ is called a section because it can be viewed as a section of the first projection from $\Sigma\,\Gamma\,A$ to Γ but we define it without using the projection.

First we define the syntactic presheaf interpretation TM as given in the table. TM_Δ is a presheaf over REN, the action on morphisms is $\mathsf{TM}_\Delta\,(\beta : \mathsf{Vars}\,\Omega\,\Psi)\,\sigma := \sigma \circ \ulcorner\beta\urcorner$. TM_A is a family of presheaves over TM_Γ, TM_σ is a natural transformation and TM_t is a section. The action on morphisms and the functor laws for TM_A and the naturality laws for TM_σ and TM_t are straightforward. TM is not a presheaf model, it is just the syntax in a different structure so that it matches the motives of a presheaf model.

In the logical predicate interpretation, a context Δ is mapped to a family of presheaves over TM_Δ. That is, for every substitution $\rho : \mathsf{TM}_\Delta\,\Psi$ we have a set $[\![\Delta]\!]_\Psi\,\rho$ which expresses that the logical predicate holds for ρ. Moreover, we have the renaming $[\![\Delta]\!]\,\beta : [\![\Delta]\!]\,\rho \to [\![\Delta]\!]\,(\mathsf{TM}_\Delta\,\beta\,\rho)$.

$[\![A]\!]$ is the logical predicate at a type A. It depends on a substitution (for which the predicate needs to hold) and a term. $[\![A]\!]_\Psi\,(\rho, s, \alpha)$ expresses that the logical predicate holds for term $s : \mathsf{Tm}\,\Psi\,A[\rho]$. It is also stable under renamings.

$$\frac{A : \mathsf{Ty}\,\Gamma \qquad \Psi : |\mathsf{REN}| \qquad \rho : \mathsf{TM}_\Gamma\,\Psi \qquad s : \mathsf{TM}_A\,\rho \qquad \alpha : [\![\Gamma]\!]_\Psi\,\rho}{[\![A]\!]_\Psi\,(\rho, s, \alpha) : \mathsf{Set}}$$

$$[\![A]\!]\,\beta : [\![A]\!]\,(\rho, s, \alpha) \to [\![A]\!]\,(\mathsf{TM}_\Gamma\,\beta\,\rho, \mathsf{TM}_A\,\beta\,s, [\![\Gamma]\!]\,\beta\,\alpha)$$

A substitution σ is mapped to $[\![\sigma]\!]$ which expresses the fundamental theorem of the logical predicate for σ: for any other substitution ρ for which the predicate holds, we can compose it with σ and the predicate will hold for the composition. The fundamental theorem is also natural.

$$\frac{\sigma : \mathsf{Tms}\,\Gamma\,\Delta \qquad \Psi : |\mathsf{REN}| \qquad \rho : \mathsf{TM}_\Gamma\,\Psi \qquad \alpha : [\![\Gamma]\!]_\Psi\,\rho}{[\![\sigma]\!]_\Psi\,(\rho, \alpha) : [\![\Delta]\!]_\Psi\,(\sigma \circ \rho)}$$

$$[\![\Delta]\!]\,\beta\,([\![\sigma]\!]\,(\rho, \alpha)) \equiv [\![\sigma]\!]\,(\mathsf{TM}_\Gamma\,\beta\,\rho, [\![\Gamma]\!]\,\beta\,\alpha)$$

A term t is mapped to the fundamental theorem for the term: given a substitution ρ for which the predicate holds, it also holds for $t[\rho]$ in a natural way.

$$\frac{t : \mathsf{Tm}\,\Gamma\,A \qquad \Psi : |\mathsf{REN}| \qquad \rho : \mathsf{TM}_\Gamma\,\Psi \qquad \alpha : [\![\Gamma]\!]_\Psi\,\rho}{[\![t]\!]_\Psi\,(\rho, \alpha) : [\![A]\!]_\Psi\,(\rho, t[\rho], \alpha)}$$

$$[\![A]\!]\,\beta\,([\![t]\!]\,(\rho, \alpha)) \equiv [\![t]\!]\,(\mathsf{TM}_\Gamma\,\beta\,\rho, [\![\Gamma]\!]\,\beta\,\alpha)$$

We define the presheaf $\mathsf{TM}^\mathsf{U} : \mathsf{PSh}\,\mathsf{REN}$ and a family over it $\mathsf{TM}^\mathsf{El} : \mathsf{FamPSh}\,\mathsf{TM}^\mathsf{U}$. The actions on objects are $\mathsf{TM}^\mathsf{U}\,\Psi := \mathsf{Tm}\,\Psi\,\mathsf{U}$ and $\mathsf{TM}^\mathsf{El}{}_\Psi\,\hat{A} := \mathsf{Tm}\,\Psi\,(\mathsf{El}\,\hat{A})$. The action on a morphism β is just substitution $-[\ulcorner\beta\urcorner]$ for both.

Note that the base category of the logical predicate interpretation is fixed to REN. However we parameterise the interpretation by the predicate at the base type U and base family El. These are denoted by $\bar{\mathsf{U}}$ and $\bar{\mathsf{El}}$ and have the following types.

$$\bar{\mathsf{U}} \;: \mathsf{FamPSh}\,\mathsf{TM}^\mathsf{U}$$

$$\bar{\mathsf{El}} : \mathsf{FamPSh}\,\Big(\Sigma\,\big(\Sigma\,(\mathsf{TM}^\mathsf{U}\,\mathsf{TM}^\mathsf{El})\big)\,\bar{\mathsf{U}}[\mathsf{wk}]\Big)$$

Now we list the methods for each constructor in the same order as we have given them in section 3. We omit the proofs of functoriality/naturality only for reasons of space.

The logical predicate trivially holds at the empty context, and it holds at an extended context for ρ if it holds at the smaller context at $\pi_1\,\rho$ and if it holds at the type which extends the context for $\pi_2\,\rho$. The second part obviously depends on the first. The action on morphisms for context extension is pointwise. Here we omitted some usages of ${}_{-*}{-}$ e.g. $[\![\Gamma]\!]\,\beta\,\alpha$ is only well-typed in that position when we transport along the equality $\pi_1\,\rho \circ \ulcorner\beta\urcorner \equiv \pi_1\,(\rho \circ \ulcorner\beta\urcorner)$. From now on we will omit transports and the usages of $\ulcorner - \urcorner$ in most cases for readability.

$$\begin{aligned}
&\llbracket \cdot \rrbracket_\Psi\,(\rho : \mathsf{TM}.\,\Psi) &&:= \top\\
&\llbracket \Gamma, A \rrbracket_\Psi\,(\rho : \mathsf{TM}_{\Gamma,A}\,\Psi) &&:= \Sigma(\alpha : \llbracket \Gamma \rrbracket_\Psi\,(\pi_1\,\rho)).\llbracket A \rrbracket_\Psi\,(\pi_1\,\rho, \pi_2\,\rho, \alpha)\\
&\llbracket \Gamma, A \rrbracket\,(\beta : \mathsf{Vars}\,\Omega\,\Psi)\,(\alpha, a) &&:= (\llbracket \Gamma \rrbracket\,\beta\,\alpha, \llbracket A \rrbracket\,\beta\,a)
\end{aligned}$$

The logical predicate at a substituted type is the logical predicate at the type and we need to use the fundamental theorem at the substitution to lift the witness of the predicate for the substitution. Renaming a substituted type is the same as renaming in the original type. The logical predicate at the base type and family says what we have given as parameters. Renaming also comes from these parameters.

$$\begin{aligned}
&\llbracket A[\sigma] \rrbracket\,(\rho, s, \alpha) &&:= \llbracket A \rrbracket\,(\sigma \circ \rho, s, \llbracket \sigma \rrbracket\,(\rho, \alpha)) &\qquad& \llbracket A[\sigma] \rrbracket\,\beta\,a &&:= \llbracket A \rrbracket\,\beta\,a\\
&\llbracket \mathsf{U} \rrbracket\,(\rho, s, \alpha) &&:= \bar{\mathsf{U}}\,({}_{\mathsf{U}[]_*}s) && \llbracket \mathsf{U} \rrbracket\,\beta\,a &&:= \bar{\mathsf{U}}\,\beta\,a\\
&\llbracket \mathsf{El}\,\hat{A} \rrbracket\,(\rho, s, \alpha) &&:= \bar{\mathsf{El}}\,(\hat{A}[\rho], s, \llbracket \hat{A} \rrbracket\,(\rho, \alpha)) && \llbracket \mathsf{El}\,\hat{A} \rrbracket\,\beta\,a &&:= \bar{\mathsf{El}}\,\beta\,a
\end{aligned}$$

The logical predicate holds for a function s when we have that if the predicate holds for an argument u (at A, witnessed by v), so it holds for $s\$u$ at B. In addition, we have a Kripke style generalisation: this should be true for $s[\beta]$ given a morphism β in a natural way. Renaming a witness of the logical predicate at the function type is postcomposing the Kripke morphism by it.

$$\begin{aligned}
&\llbracket \Pi\,A\,B \rrbracket_\Psi\,(\rho : \mathsf{TM}_\Gamma\,\Psi, s, \alpha)\\
&\quad := \Sigma\Big(\mathsf{map} : \big(\beta : \mathsf{Vars}\,\Omega\,\Psi\big)\big(u : \mathsf{TM}_A\,(\rho \circ \beta)\big)\big(v : \llbracket A \rrbracket_\Omega\,(\rho \circ \beta, u, \llbracket \Gamma \rrbracket\,\beta\,\alpha)\big)\\
&\qquad\qquad\qquad \to \llbracket B \rrbracket_\Omega\,\big((\rho \circ \beta, u), s[\beta]\$u, (\llbracket \Gamma \rrbracket\,\beta\,\alpha, v)\big)\Big)\\
&\qquad\quad .\forall \beta, u, v, \gamma.\llbracket B \rrbracket\,\gamma\,(\mathsf{map}\,\beta\,u\,v) \equiv \mathsf{map}\,(\beta \circ \gamma)\,(u[\gamma])\,(\llbracket A \rrbracket\,\gamma\,v)\\
&\llbracket \Pi\,A\,B \rrbracket\,\beta'\,(\mathsf{map}, \mathsf{nat}) := (\lambda\beta.\mathsf{map}\,(\beta' \circ \beta), \lambda\beta.\mathsf{nat}\,(\beta' \circ \beta))
\end{aligned}$$

Now we list the methods for the substitution constructors, that is, we prove the fundamental theorem for substitutions. We omit the naturality proofs. The object theoretic constructs map to their metatheoretic counterparts: identity becomes identity, composition becomes composition, the empty substitution becomes the element of the unit type, comprehension becomes pairing, first projection becomes first projection.

$$\begin{aligned}
&\llbracket \mathsf{id} \rrbracket\,(\rho, \alpha) &&:= \alpha\\
&\llbracket \sigma \circ \nu \rrbracket\,(\rho, \alpha) &&:= \llbracket \sigma \rrbracket\,(\nu \circ \rho, \llbracket \nu \rrbracket\,(\rho, \alpha))\\
&\llbracket \epsilon \rrbracket\,(\rho, \alpha) &&:= \mathsf{tt}\\
&\llbracket \sigma, t \rrbracket\,(\rho, \alpha) &&:= \llbracket \sigma \rrbracket\,(\rho, \alpha), \llbracket t \rrbracket\,(\rho, \alpha)\\
&\llbracket \pi_1\,\sigma \rrbracket\,(\rho, \alpha) &&:= \mathsf{proj}_1\,(\llbracket \sigma \rrbracket\,(\rho, \alpha))
\end{aligned}$$

The fundamental theorem for substituted terms and the second projection is again just composition and second projection.

$$\begin{aligned}
&\llbracket t[\sigma] \rrbracket\,(\rho, \alpha) &&:= \llbracket t \rrbracket\,(\sigma \circ \rho, \llbracket \sigma \rrbracket\,(\rho, \alpha))\\
&\llbracket \pi_2\,\sigma \rrbracket\,(\rho, \alpha) &&:= \mathsf{proj}_2\,(\llbracket \sigma \rrbracket\,(\rho, \alpha))
\end{aligned}$$

The fundamental theorem for lam and app are more interesting. For lam, the map function is using the fundamental theorem for t which is in the context extended by the domain type $A : \mathsf{Ty}\,\Gamma$, so we need to supply an extended substitution and a witness of the predicate. Moreover, we need to rename the substitution ρ and the witness of the predicate α to account for the Kripke property. The naturality is given by the naturality of the term itself. Application uses the map part of the logical predicate and the identity renaming.

$$\begin{aligned}
\llbracket \mathsf{lam}\, t \rrbracket\,(\rho, \alpha) &:= \Big(\lambda \beta, u, v. \llbracket t \rrbracket\,\big((\rho \circ \beta, u), (\llbracket \Gamma \rrbracket\,\beta\,\alpha, v)\big) \\
&\qquad\quad , \lambda \beta, u, v, \gamma. \mathsf{natS}\,\llbracket t \rrbracket\,\big((\rho \circ \beta, u), (\llbracket \Gamma \rrbracket\,\beta\,\alpha, v)\big)\,\gamma\Big) \\
\llbracket \mathsf{app}\, t \rrbracket\,(\rho, \alpha) &:= \mathsf{map}\,\big(\llbracket t \rrbracket\,(\pi_1\,\rho, \mathsf{proj}_1\,\alpha)\big)\,\mathsf{id}\,(\pi_2\,\rho)\,(\mathsf{proj}_2\,\alpha)
\end{aligned}$$

Lastly, we need to provide methods for the equality constructors. We won't list all of these proofs as they are quite straightforward, but as examples we show the semantic versions of the laws $[][]$ and $\pi_2\beta$. For $[][]$, we have to show that the two families of presheaves $\llbracket A[\sigma][\nu] \rrbracket$ and $\llbracket A[\sigma \circ \nu] \rrbracket$ are equal. It is enough to show that their action on objects and morphisms coincides as the equalities will be equal by K. Note that we use function extensionality to show the equality of the presheaves from the pointwise equality of actions. When we unfold the definitions for the actions on objects we see that the results are equal by associativity.

$$\begin{aligned}
&\llbracket A[\sigma][\nu] \rrbracket\,(\rho, s, \alpha) \\
= {} & \llbracket A \rrbracket\,\Big(\sigma \circ (\nu \circ \rho), s, \llbracket \sigma \rrbracket\,\big(\nu \circ \rho, \llbracket \nu \rrbracket\,(\rho, \alpha)\big)\Big) \\
\equiv {} & \llbracket A \rrbracket\,\Big((\sigma \circ \nu) \circ \rho, s, \llbracket \sigma \rrbracket\,\big(\nu \circ \rho, \llbracket \nu \rrbracket\,(\rho, \alpha)\big)\Big) \\
= {} & \llbracket A[\sigma \circ \nu] \rrbracket\,(\rho, s, \alpha)
\end{aligned}$$

The actions on morphisms are equal by unfolding the definitions.

$$\llbracket A[\sigma][\nu] \rrbracket\,\beta\,a = \llbracket A \rrbracket\,\beta\,a = \llbracket A[\sigma \circ \nu] \rrbracket\,\beta\,a$$

For $\pi_2\beta$ we need to show that two sections $\llbracket \pi_2\,(\sigma, t) \rrbracket$ and $\llbracket t \rrbracket$ are equal, and again, the law parts of the sections will be equal by K.

$$\llbracket \pi_2\,(\sigma, t) \rrbracket\,(\rho, \alpha) = \mathsf{proj}_2\,\big(\llbracket \sigma, t \rrbracket\,(\rho, \alpha)\big) = \llbracket t \rrbracket\,(\rho, \alpha)$$

6 Normal forms

We define η-long β-normal forms mutually with neutral terms. Neutral terms are terms where a variable is in a key position which precludes the application of the rule $\Pi\beta$. Embeddings back into the syntax are defined mutually in the obvious way. Note that neutral terms and normal forms are indexed by types, not normal types.

$$\begin{aligned}
&\mathsf{data\ Ne} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set} \\
&\mathsf{data\ Nf}\ : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set} \\
&\ulcorner - \urcorner \quad\ : \mathsf{Nf}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \\
&\mathsf{data\ Ne} \\
&\quad \mathsf{var} \quad : \mathsf{Var}\,\Gamma\,A \to \mathsf{Ne}\,\Gamma\,A \\
&\quad \mathsf{app} \quad : \mathsf{Ne}\,\Gamma\,(\Pi\,A\,B) \to (v : \mathsf{Nf}\,\Gamma\,A) \\
&\qquad\qquad \to \mathsf{Ne}\,\Gamma\,(B[<\ulcorner v \urcorner>])
\end{aligned}$$

$$\begin{aligned}
&\mathsf{data\ Nf} \\
&\quad \mathsf{neuU}\ \ : \mathsf{Ne}\,\Gamma\,\mathsf{U} \to \mathsf{Nf}\,\Gamma\,\mathsf{U} \\
&\quad \mathsf{neuEl} : \mathsf{Ne}\,\Gamma\,(\mathsf{El}\,\hat{A}) \to \mathsf{Nf}\,\Gamma\,(\mathsf{El}\,\hat{A}) \\
&\quad \mathsf{lam} \quad\ : \mathsf{Nf}\,(\Gamma, A)\,B \to \mathsf{Nf}\,\Gamma\,(\Pi\,A\,B) \\
&\ulcorner - \urcorner \qquad : \mathsf{Ne}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A
\end{aligned}$$

We define lists of neutral terms and normal forms. X is a parameter of the list, it can stand for both Ne and Nfs.

$$\begin{array}{l}
\mathsf{data}\ {-}\mathsf{s}\,(X : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set}) : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set} \\
\ulcorner {-} \urcorner : X\mathsf{s}\,\Gamma\,\Delta \to \mathsf{Tms}\,\Gamma\,\Delta \\
\mathsf{data}\ X\mathsf{s} \\
\quad \epsilon \qquad : X\mathsf{s}\,\Gamma\,\cdot \\
\quad {-},{-} : (\tau : X\mathsf{s}\,\Gamma\,\Delta) \to X\,\Gamma\,A[\ulcorner\tau\urcorner] \to X\mathsf{s}\,\Gamma\,(\Delta, A)
\end{array}$$

We also need renamings of (lists of) normal forms and neutral terms together with lemmas relating their embeddings to terms. Again, X can stand for both Ne and Nf.

$$\begin{array}{ll}
{-}[{-}]\ \ : X\,\Gamma\,A \to (\beta : \mathsf{Vars}\,\Psi\,\Gamma) \to X\,\Psi\,A[\ulcorner\beta\urcorner] & \qquad \ulcorner[]\urcorner : \ulcorner n\urcorner[\ulcorner\beta\urcorner] \equiv \ulcorner n[\beta]\urcorner \\
{-} \circ {-} : X\mathsf{s}\,\Gamma\,\Delta \to \mathsf{Vars}\,\Psi\,\Gamma \to X\mathsf{s}\,\Psi\,\Delta & \qquad \ulcorner\tau\urcorner \circ \ulcorner\beta\urcorner \equiv \ulcorner\tau \circ \beta\urcorner
\end{array}$$

Now we can define the presheaf X_Γ and families of presheaves X_A for any $A : \mathsf{Ty}\,\Gamma$ where X is either NE or NF. The definitions follow that of TM.

$$\begin{array}{llll}
\Gamma : \mathsf{Con} & X_\Gamma\ : \mathsf{PSh}\,\mathsf{REN} & X_\Gamma\,\Psi \qquad\qquad := X\mathsf{s}\,\Psi\,\Gamma & X_\Gamma\,\beta\,\tau := \tau \circ \beta \\
A : \mathsf{Ty}\,\Gamma & X_A : \mathsf{FamPSh}\,\mathsf{TM}_\Gamma & X_A\,(\rho : \mathsf{TM}_\Gamma\,\Psi) := X\,\Psi\,A[\rho] & X_A\,\beta\,n := n[\beta]
\end{array}$$

We set the parameters of the logical predicate at the base type and family by defining $\bar{\mathsf{U}}$ and $\bar{\mathsf{El}}$. The predicate holds for a term if there is a neutral term of the corresponding type which is equal to the term. The action on morphisms is just renaming.

$$\begin{array}{l}
\bar{\mathsf{U}} : \mathsf{FamPSh}\,\mathsf{TM}^{\mathsf{U}} \\
\bar{\mathsf{U}}_\Psi\,(\hat{A} : \mathsf{Tm}\,\Psi\,\mathsf{U}) := \Sigma(n : \mathsf{Ne}\,\Psi\,\mathsf{U}).\hat{A} \equiv \ulcorner n\urcorner \\
\bar{\mathsf{El}} : \mathsf{FamPSh}\left(\Sigma\,(\Sigma\,(\mathsf{TM}^{\mathsf{U}}\,\mathsf{TM}^{\mathsf{El}}))\,\bar{\mathsf{U}}[\mathsf{wk}]\right) \\
\bar{\mathsf{El}}_\Psi\,(\hat{A}, t : \mathsf{Tm}\,\Psi\,(\mathsf{El}\,\hat{A}), p) := \Sigma(n : \mathsf{Ne}\,\Psi\,(\mathsf{El}\,\hat{A})).t \equiv \ulcorner n\urcorner
\end{array}$$

Now we can interpret any term in the logical predicate model over REN with base type interpretations $\bar{\mathsf{U}}$ and $\bar{\mathsf{El}}$. We denote the interpretation of t by $[\![t]\!]$.

7 Quote and unquote

By the logical predicate interpretation using $\bar{\mathsf{U}}$ and $\bar{\mathsf{El}}$ we have the following two things:

- terms at the base type and base family are equal to a normal form,
- this property is preserved by the other type formers — this is what the logical predicate says at function types and substituted types.

We make use of this fact to lift the first property to any type. We do this by defining a quote function by induction on the type. Quote takes a term which preserves the predicate and maps it to a normal form that it is equal to it. Because of function spaces, we need a function in the other direction as well, mapping neutral terms to the witness of the predicate.

More precisely, we define the quote function q and unquote u by induction on the structure of contexts and types. For this, we need to define a model of type theory in which only the motives for contexts and types are interesting.

First we define families of presheaves for contexts and types which express that there is an equal normal form. The actions on objects are given as follows.

$$\begin{array}{ll} \mathsf{NF}^{\equiv}{}_{\Delta} : \mathsf{FamPSh}\,\mathsf{TM}_{\Delta} & \mathsf{NF}^{\equiv}{}_{A} : \mathsf{FamPSh}\,(\Sigma\,\mathsf{TM}_{\Gamma}\,\mathsf{TM}_{A}) \\ \mathsf{NF}^{\equiv}{}_{\Delta}\,(\rho : \mathsf{TM}_{\Delta}\,\Psi) := \Sigma(\rho' : \mathsf{NF}_{\Delta}\,\Psi).\rho \equiv \ulcorner\rho'\urcorner & \mathsf{NF}^{\equiv}{}_{A}\,(\rho, s) := \Sigma(s' : \mathsf{NF}_{A}\,\rho).s \equiv \ulcorner s'\urcorner \end{array}$$

We use these to write down the motives for contexts and types. We use sections to express the commutativity of the diagram in figure 2. We only write Σ once for iterated usage.

$$\begin{array}{llll} \mathsf{u}_{\Delta} : \mathsf{NE}_{\Delta} & \xrightarrow{\mathsf{s}} \llbracket\Delta\rrbracket[\ulcorner - \urcorner] & \mathsf{u}_{A} : \Sigma\,\mathsf{TM}_{\Gamma}\,\mathsf{NE}_{A}\,(\llbracket\Gamma\rrbracket[\mathsf{wk}]) & \xrightarrow{\mathsf{s}} \llbracket A\rrbracket[\mathsf{id}, \ulcorner - \urcorner, \mathsf{id}] \\ \mathsf{q}_{\Delta} : \Sigma\,\mathsf{TM}_{\Delta}\,\llbracket\Delta\rrbracket & \xrightarrow{\mathsf{s}} \mathsf{NF}^{\equiv}{}_{\Delta}[\mathsf{wk}] & \mathsf{q}_{A} : \Sigma\,\mathsf{TM}_{\Gamma}\,\mathsf{TM}_{A}\,(\llbracket\Gamma\rrbracket[\mathsf{wk}])\,\llbracket A\rrbracket & \xrightarrow{\mathsf{s}} \mathsf{NF}^{\equiv}{}_{A}[\mathsf{wk}][\mathsf{wk}] \end{array}$$

Unquote for a context takes a neutral substitution and returns a proof that the logical predicate holds for it. Quote takes a substitution for which the predicate holds and returns a normal substitution together with a proof that the original substitution is equal (convertible) to the normal one (embedded into substitutions by $\ulcorner - \urcorner$). The types of unquote and quote for types are more involved as they depend on a substitution for which the predicate needs to hold. Unquote for a type takes such a substitution and a neutral term at the type substituted by this substitution and returns a proof that the predicate holds for this neutral term. The natural transformation $\mathsf{id}, \ulcorner - \urcorner, \mathsf{id}$ is defined in the obvious way, it just embeds the second component (the neutral term) into terms. Quote for a type takes a term of this type for which the predicate holds and returns a normal form at this type together with a proof that it is equal to the term. Here again, another substitution is involved.

The motives for substitutions and terms are the constant unit families.

We will list the methods for contexts and types excluding the naturality proofs for brevity.

Unquote and quote for the empty context are trivial, for extended contexts they are pointwise. $\mathsf{ap},$ is the congruence law of substitution extension $-,-$.

$$\begin{array}{l} \mathsf{u}_{\cdot}\,(\tau : \mathsf{NE}_{\cdot}\,\Psi) : \top := \mathsf{tt} \\ \mathsf{q}_{\cdot}\,\big((\sigma : \mathsf{TM}_{\cdot}\,\Psi), (\alpha : \top)\big) : \Sigma(\rho' : \mathsf{NF}_{\cdot}\,\Psi).\rho \equiv \ulcorner\rho'\urcorner := (\epsilon, \epsilon\eta) \\ \mathsf{u}_{\Delta,A}\,\big((\tau, n) : \mathsf{NE}_{\Delta,A}\,\Psi\big) : \Sigma\big(\alpha : \llbracket\Delta\rrbracket\,(\pi_1\,\ulcorner\tau, n\urcorner)\big).\llbracket A\rrbracket\,(\pi_1\,\ulcorner\tau, n\urcorner, \pi_2\,\ulcorner\tau, n\urcorner, \alpha) \\ \quad := \mathsf{u}_{\Delta}\,\tau, \mathsf{u}_{A}\,(\ulcorner\tau\urcorner, n, \mathsf{u}_{\Delta}\,\tau) \\ \mathsf{q}_{\Delta,A}\,\big((\rho : \mathsf{TM}_{\Delta}\,\Psi), (\alpha, a) : \llbracket\Delta, A\rrbracket\,\rho\big) : \Sigma(\rho' : \mathsf{NF}_{\Delta,A}\,\Psi).\rho \equiv \ulcorner\rho'\urcorner \\ \quad := \mathsf{let}\,(\tau, p) := \mathsf{q}_{\Delta}\,(\pi_1\,\rho, \alpha);\ (n, q) := \mathsf{q}_{A}\,(\pi_1\,\rho, \pi_2\,\rho, \alpha, a)\ \mathsf{in}\,\big((\tau, n), (\mathsf{ap},\, p\, q)\big) \end{array}$$

(Un)quoting a substituted type is the same as (un)quoting at the type and using the fundamental theorem at the substitution to lift the witness of the predicate α. As expected, unquoting at the base type is simply returning the neutral term itself and the witness of the predicate will be reflexivity, while quote just returns the witness of the predicate.

$$\begin{array}{lll} \mathsf{u}_{A[\sigma]}\,(\rho, n, \alpha) & : \llbracket A\rrbracket\,\big(\sigma \circ \rho, \ulcorner n\urcorner, \llbracket\sigma\rrbracket\,(\rho, \alpha)\big) & := \mathsf{u}_{A}\,\big(\sigma \circ \rho, n, \llbracket\sigma\rrbracket\,(\rho, \alpha)\big) \\ \mathsf{q}_{A[\sigma]}\,(\rho, s, \alpha, a) & : \Sigma(s' : \mathsf{NF}_{A[\sigma]}\,\rho).s \equiv \ulcorner s'\urcorner & := \mathsf{q}_{A}\,\big(\sigma \circ \rho, s, \llbracket\sigma\rrbracket\,(\rho, \alpha), a\big) \end{array}$$

$$\begin{array}{lll} \mathsf{u}_{\mathsf{U}}\,\big((\rho : \mathsf{TM}_{\Gamma}\,\Psi), (n : \mathsf{Ne}\,\Psi\,\mathsf{U}[\rho]), \alpha\big) & : \Sigma(n' : \mathsf{NF}_{\mathsf{U}}\,\mathsf{id}).\ulcorner n\urcorner \equiv \ulcorner n'\urcorner & := \mathsf{neuU}\,({}_{\mathsf{U}[]*}n), \mathsf{refl} \\ \mathsf{q}_{\mathsf{U}}\,\Big(\rho, t, \alpha, \big(a : \mathsf{NF}^{\equiv}{}_{\mathsf{U}}\,(\mathsf{id}, t)\big)\Big) & : \mathsf{NF}^{\equiv}{}_{\mathsf{U}}\,(\rho, t) & := {}_{\mathsf{U}[]*}a \end{array}$$

$$\begin{array}{lll} \multicolumn{3}{l}{\mathsf{u}_{\mathsf{El}\,\hat{A}}\,\big((\rho : \mathsf{TM}_{\Gamma}\,\Psi), (n : \mathsf{Ne}\,\Psi\,(\mathsf{El}\,\hat{A}[\rho])), \alpha\big) : \Sigma(n' : \mathsf{NF}_{\mathsf{El}\,\hat{A}}\,\mathsf{id}).\ulcorner n\urcorner \equiv \ulcorner n'\urcorner} \\ \multicolumn{3}{l}{\quad := \mathsf{neuEl}\,({}_{\mathsf{El}[]*}n), \mathsf{refl}} \\ \mathsf{q}_{\mathsf{El}\,\hat{A}}\,\Big(\rho, t, \alpha, \big(a : \mathsf{NF}^{\equiv}{}_{\mathsf{El}\,\hat{A}}\,(\mathsf{id}, t)\big)\Big) & : \mathsf{NF}^{\equiv}{}_{\mathsf{El}\,\hat{A}}\,(\rho, t) & := {}_{\mathsf{El}[]*}a \end{array}$$

We only show the mapping part of unquoting a function. To show that n preserves the predicate, we show that it preserves the predicate for every argument u for which the predicate holds (by v). We quote the argument, thereby getting it in normal form (m), and now we can unquote the neutral term ($\mathsf{app}\, n[\beta]\, m$) to get the result. We also need to transport the result along the proof p that $u \equiv \ulcorner m \urcorner$.

$$\begin{aligned}
&\mathsf{map}\Big(\mathsf{u}_{\Pi\, A\, B}\,((\rho : \mathsf{TM}_{\Gamma}\, \Psi), (n : \mathsf{NE}_{\Pi\, A\, B}\, \rho), \alpha)\Big)\,(\beta : \mathsf{Vars}\,\Omega\,\Psi)\,\big(u : \mathsf{TM}_A\,(\rho \circ \ulcorner \beta \urcorner)\big)\\
&\qquad \big(v : \llbracket A \rrbracket_{\Omega}\,(\rho \circ \ulcorner \beta \urcorner, u, \llbracket \Gamma \rrbracket\, \beta\, \alpha)\big) \;:\; \llbracket B \rrbracket_{\Omega}\,\big((\rho \circ \ulcorner \beta \urcorner, u), (\ulcorner n \urcorner[\ulcorner \beta \urcorner])\$u, (\llbracket \Gamma \rrbracket\, \beta\, \alpha, v)\big)\\
&\quad := \mathsf{let}\,(m, p) := \mathsf{q}_A\,(\rho \circ \ulcorner \beta \urcorner, u, \llbracket \Gamma \rrbracket\, \beta\, \alpha, v)\;\mathsf{in}\;\mathsf{u}_B\,\big((\rho \circ \ulcorner \beta \urcorner, u), ({}_{p*}\mathsf{app}\, n[\beta]\, m), (\llbracket \Gamma \rrbracket\, \beta\, \alpha, v)\big)
\end{aligned}$$

The normal form of a function t is $\mathsf{lam}\, n$ for some normal form n which is in the extended context. We get this n by quoting $\mathsf{app}\, t$ in the extended context. f is the witness that t preserves the relation for any renaming, and we use the renaming $\mathsf{wk}\,\mathsf{id}$ to use f in the extended context. The argument of f in this case will be the zero de Bruijn index vze and we need to unquote it to get the witness that it preserves the logical predicate. This is the place where the Kripke property of the logical relation is needed: the base category of the Kripke logical relation needs to minimally include the morphism $\mathsf{wk}\,\mathsf{id}$.

$$\begin{aligned}
&\mathsf{q}_{\Pi\, A\, B}\,(\rho, t, \alpha, f) : \Sigma(t' : \mathsf{NF}_{\Pi\, A\, B}\, \rho).t \equiv \ulcorner t' \urcorner\\
&\quad := \mathsf{let}\, a \qquad := \mathsf{u}_A\,(\rho \circ \ulcorner \mathsf{wk}\,\mathsf{id} \urcorner, \mathsf{var}\,\mathsf{vze}, \llbracket \Gamma \rrbracket\,(\mathsf{wk}\,\mathsf{id})\,\alpha)\\
&\qquad\qquad (n, p) := \mathsf{q}_B\,\big(\rho^A, \mathsf{app}\, t, (\llbracket \Gamma \rrbracket\,(\mathsf{wk}\,\mathsf{id})\,\alpha, a), \mathsf{map}\, f\,(\mathsf{wk}\,\mathsf{id})\,\ulcorner \mathsf{vze} \urcorner\, a\big)\\
&\qquad\quad \mathsf{in}\;\;(\mathsf{lam}\, n, \Pi\eta^{-1} \centerdot \mathsf{ap}\,\mathsf{lam}\, p)
\end{aligned}$$

We have to verify the equality laws for types. Note that we use function extensionality to show that the corresponding quote and unquote functions are equal. The naturality proofs will be equal by K.

(Un)quote preserves [id] by the left identity law.

$$\begin{aligned}
\mathsf{u}_{A[\mathsf{id}]}\,(\rho, n, \alpha) &= \mathsf{u}_A\,(\mathsf{id} \circ \rho, n, \alpha) &&\equiv \mathsf{u}_A\,(\rho, n, \alpha)\\
\mathsf{q}_{A[\mathsf{id}]}\,(\rho, s, \alpha, a) &= \mathsf{q}_A\,(\mathsf{id} \circ \rho, s, \alpha, a) &&\equiv \mathsf{q}_A\,(\rho, s, \alpha, a)
\end{aligned}$$

(Un)quote preserves [][] by associativity for substitutions.

$$\begin{aligned}
&\mathsf{u}_{A[\sigma][\nu]}\,(\rho, n, \alpha)\\
=\,&\mathsf{u}_A\,\big(\sigma \circ (\nu \circ \rho), n, \llbracket \sigma \rrbracket\,(\nu \circ \rho, \llbracket \nu \rrbracket\,(\rho, \alpha))\big)\\
\equiv\,&\mathsf{u}_A\,\big((\sigma \circ \nu) \circ \rho, n, \llbracket \sigma \rrbracket\,(\nu \circ \rho, \llbracket \nu \rrbracket\,(\rho, \alpha))\big)\\
=\,&\mathsf{u}_{A[\sigma \circ \nu]}\,(\rho, n, \alpha)
\end{aligned}$$

$$\begin{aligned}
&\mathsf{q}_{A[\sigma][\nu]}\,(\rho, s, \alpha, a)\\
=\,&\mathsf{q}_A\,\big(\sigma \circ (\nu \circ \rho), s, \llbracket \sigma \rrbracket\,(\nu \circ \rho, \llbracket \nu \rrbracket\,(\rho, \alpha)), a\big)\\
\equiv\,&\mathsf{q}_A\,((\sigma \circ \nu) \circ \rho, s, \llbracket \sigma \rrbracket\,(\nu \circ \rho, \llbracket \nu \rrbracket\,(\rho, \alpha)), a)\\
=\,&\mathsf{q}_{A[\sigma \circ \nu]}\,(\rho, s, \alpha, a)
\end{aligned}$$

The semantic counterparts of $\mathsf{U}[]$ and $\mathsf{El}[]$ are verified as follows.

$$\begin{aligned}
\mathsf{u}_{\mathsf{U}[\sigma]}\,(\rho, n, \alpha) &= \mathsf{u}_{\mathsf{U}}\,\big(\sigma \circ \rho, n, \llbracket \sigma \rrbracket\,(\rho, \alpha)\big) &&= (n, \mathsf{refl}) &&= \mathsf{u}_{\mathsf{U}}\,(\rho, n, \alpha)\\
\mathsf{q}_{\mathsf{U}[\sigma]}\,(\rho, t, \alpha, a) &= \mathsf{q}_{\mathsf{U}}\,\big(\sigma \circ \rho, t, \llbracket \sigma \rrbracket\,(\rho, \alpha), a\big) &&= a &&= \mathsf{u}_{\mathsf{U}}\,(\rho, t, \alpha, a)\\
\mathsf{u}_{(\mathsf{El}\,\hat{A})[\sigma]}\,(\rho, n, \alpha) &= \mathsf{u}_{\mathsf{El}\,\hat{A}}\,\big(\sigma \circ \rho, n, \llbracket \sigma \rrbracket\,(\rho, \alpha)\big) &&= (n, \mathsf{refl}) &&= \mathsf{u}_{\mathsf{El}\,(\hat{A}[\sigma])}\,(\rho, n, \alpha)\\
\mathsf{q}_{(\mathsf{El}\,\hat{A})[\sigma]}\,(\rho, t, \alpha, a) &= \mathsf{q}_{\mathsf{El}\,\hat{A}}\,\big(\sigma \circ \rho, t, \llbracket \sigma \rrbracket\,(\rho, \alpha), a\big) &&= a &&= \mathsf{u}_{\mathsf{El}\,(\hat{A}[\sigma])}\,(\rho, t, \alpha, a)
\end{aligned}$$

For reasons of space, we only state what we need to verify for $\Pi[]$. It is enough to show that the mapping parts of the unquoted functions are equal and that the first components of the results of quote are equal because the other parts are equalities.

$$\begin{aligned}
\mathsf{map}\left(\mathsf{u}_{(\Pi\,A\,B)[\sigma]}\,(\rho, n, \alpha)\right) &\equiv \mathsf{map}\left(\mathsf{u}_{\Pi\,A[\sigma]\,B[\sigma^A]}\,(\rho, n, \alpha)\right)\\
\mathsf{proj}_1\left(\mathsf{q}_{(\Pi\,A\,B)[\sigma]}\,(\rho, t, \alpha, f)\right) &\equiv \mathsf{proj}_1\left(\mathsf{q}_{\Pi\,A[\sigma]\,B[\sigma^A]}\,(\rho, t, \alpha, f)\right)
\end{aligned}$$

The methods for substitutions and terms (including the equality methods) are all trivial.

8 Normalisation

Now we can define the normalisation function and show that it is complete as follows.

$$\begin{aligned}
&\mathsf{norm}_A\ \ (t : \mathsf{Tm}\,\Gamma\,A) : \mathsf{Nf}\,\Gamma\,A && := \mathsf{proj}_1\left(\mathsf{q}_A\left(\mathsf{id}, {}_{[\mathsf{id}]^{-1}*}t, \mathsf{u}_\Gamma\,\mathsf{id}, [\![t]\!]\,(\mathsf{id}, \mathsf{u}_\Gamma\,\mathsf{id})\right)\right)\\
&\mathsf{compl}_A\,(t : \mathsf{Tm}\,\Gamma\,A) : t \equiv \ulcorner\mathsf{norm}_A\,t\urcorner && := \mathsf{proj}_2\left(\mathsf{q}_A\left(\mathsf{id}, {}_{[\mathsf{id}]^{-1}*}t, \mathsf{u}_\Gamma\,\mathsf{id}, [\![t]\!]\,(\mathsf{id}, \mathsf{u}_\Gamma\,\mathsf{id})\right)\right)
\end{aligned}$$

We prove stability by mutual induction on neutral terms and normal forms.

$$\frac{n : \mathsf{Ne}\,\Gamma\,A}{[\![\ulcorner n\urcorner]\!]\,(\mathsf{id}, \mathsf{u}_\Gamma\,\mathsf{id}) \equiv \mathsf{u}_A\,(\mathsf{id}, n, \mathsf{u}_\Gamma\,\mathsf{id})} \qquad \frac{n : \mathsf{Nf}\,\Gamma\,A}{\mathsf{norm}_A\,\ulcorner n\urcorner \equiv v}$$

As our normal forms are indexed by types, we need decidability of equality of types to show decidability of equality of normal forms. For this, we need to define a model of normal forms where types are mapped to normal types (which exclude substituted types). We leave this as future work.

9 Conclusions and further work

We proved normalisation for a basic type theory with dependent types by the technique of NBE. We evaluate terms into a proof relevant logical predicate model. The model is depending on the syntax, we need to use the dependent eliminator of the syntax. Our approach can be seen as merging the presheaf model and the logical relation used in NBE for simple types [6] into a single logical predicate. This seems to be necessary because of the combination of type indexing and dependent types: the well-typedness of normalisation depends on completeness. Another property to note is that we don't normalise types, we just index normal terms by not necessarily normal types.

We are currently working on completing the formalisation[6] [9]. Most of the work here is equality reasoning. QIITs make it possible to define the syntax of type theory in a very concise way, however because of missing computation rules, reasoning with them involves lots of boilerplate. We expect that a cubical metatheory [12] with its systematic way of expressing equalities depending on equalities and its additional computation rules would significantly reduce the amount of boilerplate.

Another challenge is to extend our basic type theory with inductive types, universes and large elimination. Also, it would be interesting to see how the work fits into the setting of homotopy type theory (without assuming K). We would also like to investigate whether the logical predicate interpretation can be generalised to work over arbitrary presheaf models and how the syntactic model fits here.

[6] The current status of formalisation is that we formalised most main constructions but the functoriality and naturality properties are left as holes.

Acknowledgements. We would like to thank the anonymous reviewers for their helpful comments and suggestions.

References

1 Andreas Abel. Towards normalization by evaluation for the $\beta\eta$-calculus of constructions. In *Functional and Logic Programming*, pages 224–239. Springer, 2010.

2 Andreas Abel. *Normalization by Evaluation: Dependent Types and Impredicativity*. PhD thesis, Habilitation, Ludwig-Maximilians-Universität München, 2013.

3 Andreas Abel, Thierry Coquand, and Peter Dybjer. Normalization by evaluation for Martin-Löf type theory with typed equality judgements. In *Logic in Computer Science, 2007. LICS 2007. 22nd Annual IEEE Symposium on*, pages 3–12. IEEE, 2007.

4 The Agda Wiki, 2015. Available online.

5 Thorsten Altenkirch, Peter Dybjer, Martin Hofmann, and Phil Scott. Normalization by evaluation for typed lambda calculus with coproducts. In *16th Annual IEEE Symposium on Logic in Computer Science*, pages 303–310, 2001.

6 Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Categorical reconstruction of a reduction free normalization proof. In David Pitt, David E. Rydeheard, and Peter Johnstone, editors, *Category Theory and Computer Science*, LNCS 953, pages 182–199, 1995.

7 Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Reduction-free normalisation for a polymorphic system. In *11th Annual IEEE Symposium on Logic in Computer Science*, pages 98–106, 1996.

8 Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Reduction-free normalisation for system F. Manuscript, 1997. URL: `http://www.cs.nott.ac.uk/~txa/publ/f97.pdf`.

9 Thorsten Altenkirch and Ambrus Kaposi. Agda formalisation for the paper Normalisation by Evaluation for Dependent Types, 2016. Available online at the second author's website.

10 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 2016, pages 18–29, New York, NY, USA, 2016. ACM. `doi:10.1145/2837614.2837638`.

11 Ulrich Berger and Helmut Schwichtenberg. An inverse of the evaluation functional for typed λ-calculus. In *Logic in Computer Science, 1991. LICS'91., Proceedings of Sixth Annual IEEE Symposium on*, pages 203–211. IEEE, 1991.

12 Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom. Manuscript, December 2015.

13 Nils Anders Danielsson. A formalisation of a dependently typed language as an inductive-recursive family. In *Types for Proofs and Programs*, pages 93–109. Springer, 2006.

14 Olivier Danvy. *Type-directed partial evaluation*. Springer, 1999.

15 Peter Dybjer. Internal type theory. In *Types for Proofs and Programs*, pages 120–134. Springer, 1996.

16 Martin Hofmann. Syntax and semantics of dependent types. In *Extensional Constructs in Intensional Type Theory*, pages 13–54. Springer, 1997.

17 Ulf Norell. *Towards a practical programming language based on dependent type theory*. PhD thesis, Chalmers University of Technology, 2007.

18 Steven Schäfer, Gert Smolka, and Tobias Tebbi. Completeness and decidability of de Bruijn substitution algebra in Coq. In *Proceedings of the 2015 Conference on Certified Programs and Proofs, CPP 2015, Mumbai, India, January 15-17, 2015*, pages 67–73. ACM, 2015.

Minimal Paradefinite Logics for Reasoning with Incompleteness and Inconsistency*

Ofer Arieli[1] and Arnon Avron[2]

1 School of Computer Science, The Academic College of Tel-Aviv, Israel
oarieli@mta.ac.il

2 School of Computer Science, Tel-Aviv University, Israel
aa@math.tau.il

Abstract

Paradefinite ('beyond the definite') logics are logics that can be used for handling contradictory or partial information. As such, paradefinite logics should be both paraconsistent and paracomplete. In this paper we consider the simplest semantic framework for defining paradefinite logics, consisting of four-valued matrices, and study the better accepted logics that are induced by these matrices.

1998 ACM Subject Classification I.2.3 Deduction and Theorem Proving, I.2.4 Knowledge Representation Formalisms and Methods, F.4.1 Mathematical Logic

Keywords and phrases Paraconsistecy, Paracompleteness, 4-valued logics

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.7

1 Introduction

Uncertainty in commonsense reasoning and AI involves inconsistent and incomplete information. *Paradefinite logics* (called 'non-alethic' by da Costa and 'paranormal' by Béziau [11]) are logics that successfully handle these two types of indefinite data, and so they have the following two properties:

- *Paraconsistency* [13]: The ability to properly handle contradictory data by rejecting the principle of explosion, in which any proposition can be inferred from an inconsistent set of assumptions.
- *Paracompleteness*: The ability to properly handle incomplete data by rejecting the law of excluded middle, in which for any proposition, either that proposition is 'true' or its negation is 'true'.

Apart of these two primary requirements, a 'decent' logic for reasoning with indefinite data should have some further characteristics, like being expressive enough, faithful to classical logic as much as possible (in the sense that entailments in the logic should also hold in classical logic), and having some maximality properties (which may be intuitively interpreted by the aspiration to retain as much of classical logic as possible, while preserving paraconsistency and paracompleteness).

In this paper we are interested in the 'simplest' paradefinite logics (in terms of the number of the truth values of their semantics) that have the above properties. Obviously, two-valued logics are not adequate for this, as they cannot handle either of the two types of uncertainty. Likewise, three-valued logics can be used for handling just one type of uncertainty (see,

* This work is supported by the Israel Science Foundation (grant no. 817/15).

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 7; pp. 7:1–7:15
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

e.g., [5]), but they cannot simultaneously handle both of them. On the other hand, as shown e.g. in [10] and [2], four truth values *are* enough for reasoning with incompleteness and inconsistency.

This paper is a largely extended study of the work on 4-valued logics mentioned above. Among others, we characterize the 4-valued paradefinite matrices, consider the induced logics, examine them according to the criteria in [3], investigate their relative strengths, and introduce corresponding sound and complete Hilbert-type and Gentzen-type proof systems.

2 Preliminaries

2.1 Propositional Logics

In what follows we denote by $\mathcal{L}$ a propositional language with a set $\mathsf{Atoms}(\mathcal{L}) = \{P_1, P_2, \ldots\}$ of atomic formulas and use p, q, r to vary over this set. The set of the well-formed formulas of $\mathcal{L}$ is denoted by $\mathcal{W}(\mathcal{L})$ and $\varphi, \psi, \phi, \sigma$ will vary over its elements. The set $\mathsf{Atoms}(\varphi)$ denotes the atomic formulas occurring in φ. Sets of formulas in $\mathcal{W}(\mathcal{L})$ are called *theories* and are denoted by $\mathcal{T}$ or $\mathcal{T}'$. Finite theories are denoted by Γ or Δ. We shall abbreviate $\mathcal{T} \cup \{\psi\}$ by $\mathcal{T}, \psi$ and write $\mathcal{T}, \mathcal{T}'$ instead of $\mathcal{T} \cup \mathcal{T}'$. A *rule* in a language $\mathcal{L}$ is a pair $\langle \Gamma, \psi \rangle$, where $\Gamma \cup \{\psi\}$ is a finite theory. We shall henceforth denote such a rule by Γ/ψ.

▶ **Definition 2.1.** A (propositional) *logic* is a pair $\mathbf{L} = \langle \mathcal{L}, \vdash \rangle$, such that $\mathcal{L}$ is a propositional language, and $\vdash$ is a binary relation between theories in $\mathcal{W}(\mathcal{L})$ and formulas in $\mathcal{W}(\mathcal{L})$, satisfying the following conditions:

- *Reflexivity:* if $\psi \in \mathcal{T}$ then $\mathcal{T} \vdash \psi$.
- *Monotonicity:* if $\mathcal{T} \vdash \psi$ and $\mathcal{T} \subseteq \mathcal{T}'$, then $\mathcal{T}' \vdash \psi$.
- *Transitivity:* if $\mathcal{T} \vdash \psi$ and $\mathcal{T}', \psi \vdash \phi$ then $\mathcal{T}, \mathcal{T}' \vdash \phi$.
- *Structurality:* for every substitution θ and every $\mathcal{T}$ and ψ, if $\mathcal{T} \vdash \psi$ then $\{\theta(\varphi) \mid \varphi \in \mathcal{T}\} \vdash \theta(\psi)$.
- *Non-Triviality:* there is a non-empty theory $\mathcal{T}$ and a formula ψ such that $\mathcal{T} \nvdash \psi$.

A logic $\langle \mathcal{L}, \vdash \rangle$ is *finitary* if for every theory $\mathcal{T}$ and every formula ψ such that $\mathcal{T} \vdash \psi$ there is a *finite* theory $\Gamma \subseteq \mathcal{T}$ such that $\Gamma \vdash \psi$.

Note that the languages that are considered in the sequel are all *propositional*, as this is the heart of every paraconsistent and paracomplete logic ever studied so far. Also, we confine ourselves to paradefinite *logics*, thus no form of non-monotonic reasoning is considered in this paper.

▶ **Definition 2.2.** Let $\mathbf{L} = \langle \mathcal{L}, \vdash \rangle$ be a logic, and let S be a set of rules in $\mathcal{L}$. The *finitary* $\mathbf{L}$*-closure* $C_{\mathbf{L}}(S)$ of S is inductively defined as follows:

- $\langle \theta(\Gamma), \theta(\psi) \rangle \in C_{\mathbf{L}}(S)$, where θ is an $\mathcal{L}$-substitution, Γ is a *finite* theory in $\mathcal{W}(\mathcal{L})$, and either $\Gamma \vdash \psi$ or $\Gamma/\psi \in S$.
- If the pairs $\langle \Gamma_1, \varphi \rangle$ and $\langle \Gamma_2 \cup \{\varphi\}, \psi \rangle$ are both in $C_{\mathbf{L}}(S)$, then so is the pair $\langle \Gamma_1 \cup \Gamma_2, \psi \rangle$.

Next we define what an extension of a logic means.

▶ **Definition 2.3.** Let $\mathbf{L} = \langle \mathcal{L}, \vdash \rangle$ be a logic, and let S be a set of rules in $\mathcal{L}$.

- A logic $\mathbf{L}' = \langle \mathcal{L}, \vdash' \rangle$ is an *extension* of $\mathbf{L}$ (in the same language) if $\vdash \subseteq \vdash'$. We say that $\mathbf{L}'$ is a *proper extension* of $\mathbf{L}$, if $\vdash \subsetneq \vdash'$.
- The *extension of* $\mathbf{L}$ *by* S is the pair $\mathbf{L}^* = \langle \mathcal{L}, \vdash^* \rangle$, where $\vdash^*$ is the binary relation between theories in $\mathcal{W}(\mathcal{L})$ and formulas in $\mathcal{W}(\mathcal{L})$, defined by: $\mathcal{T} \vdash^* \psi$ if there is a finite $\Gamma \subseteq \mathcal{T}$ such that $\langle \Gamma, \psi \rangle \in C_{\mathbf{L}}(S)$.
- Extending $\mathbf{L}$ by an axiom schema φ means extending it by the rule $\emptyset/\varphi$.

The usefulness of a logic strongly depends on the question what kind of connectives are available in it. Three particularly important types of connectives are defined next.

▶ **Definition 2.4.** Let $\mathbf{L} = \langle \mathcal{L}, \vdash \rangle$ be a propositional logic.

- A binary connective $\supset$ of $\mathcal{L}$ is an *implication for* $\mathbf{L}$, if the classical deduction theorem holds for $\supset$ and $\vdash$, that is: $\mathcal{T}, \varphi \vdash \psi$ iff $\mathcal{T} \vdash \varphi \supset \psi$.
- A binary connective $\wedge$ of $\mathcal{L}$ is a *conjunction for* $\mathbf{L}$, if $\mathcal{T} \vdash \psi \wedge \varphi$ iff $\mathcal{T} \vdash \psi$ and $\mathcal{T} \vdash \varphi$.
- A binary connective $\vee$ of $\mathcal{L}$ is a *disjunction for* $\mathbf{L}$, if $\mathcal{T}, \psi \vee \varphi \vdash \sigma$ iff $\mathcal{T}, \psi \vdash \sigma$ and $\mathcal{T}, \varphi \vdash \sigma$.

We say that $\mathbf{L}$ is *semi-normal* if it has (at least) one of the three basic connectives defined above. We say that $\mathbf{L}$ is *normal* if it has *all* these three connectives.

2.2 Many-Valued Matrices

The most standard semantic way of defining many-valued logics is by using the following type of structures (see, e.g., [19, 20, 25]).

▶ **Definition 2.5.** A (multi-valued) *matrix* for a language $\mathcal{L}$ is a triple $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$, where

- $\mathcal{V}$ is a non-empty set of truth values,
- $\mathcal{D}$ is a non-empty proper subset of $\mathcal{V}$, called the *designated* elements of $\mathcal{V}$, and
- $\mathcal{O}$ is a function that associates an n-ary function $\widetilde{\diamond}_{\mathcal{M}} : \mathcal{V}^n \to \mathcal{V}$ with every n-ary connective $\diamond$ of the language $\mathcal{L}$.

In what follows, we shall denote by $\overline{\mathcal{D}}$ the elements in $\mathcal{V} \setminus \mathcal{D}$. The set $\mathcal{D}$ is used for defining satisfiability and validity as defined below:

▶ **Definition 2.6.** Let $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$ be a matrix for $\mathcal{L}$.

- An $\mathcal{M}$-*valuation* for $\mathcal{L}$ is a function $\nu : \mathcal{W}(\mathcal{L}) \to \mathcal{V}$ such that for every n-ary connective $\diamond$ of $\mathcal{L}$ and every $\psi_1, \ldots, \psi_n \in \mathcal{W}(\mathcal{L})$, $\nu(\diamond(\psi_1, \ldots, \psi_n)) = \widetilde{\diamond}_{\mathcal{M}}(\nu(\psi_1), \ldots, \nu(\psi_n))$. We denote by $\Lambda_{\mathcal{M}}$ the set of all the $\mathcal{M}$-valuations.
- A valuation $\nu \in \Lambda_{\mathcal{M}}$ is an $\mathcal{M}$-*model* of a formula ψ (alternatively, ν $\mathcal{M}$-*satisfies* ψ), if it belongs to the set $mod_{\mathcal{M}}(\psi) = \{\nu \in \Lambda_{\mathcal{M}} \mid \nu(\psi) \in \mathcal{D}\}$. The $\mathcal{M}$-models of a theory $\mathcal{T}$ are the elements of the set $mod_{\mathcal{M}}(\mathcal{T}) = \cap_{\psi \in \mathcal{T}}\, mod_{\mathcal{M}}(\psi)$.
- A formula ψ is $\mathcal{M}$-*satisfiable* if $mod_{\mathcal{M}}(\psi) \neq \emptyset$. A theory $\mathcal{T}$ is $\mathcal{M}$-satisfiable if $mod_{\mathcal{M}}(\mathcal{T}) \neq \emptyset$.

In the sequel, when it is clear from the context, we shall sometimes omit the subscript '$\mathcal{M}$' and the tilde sign from $\widetilde{\diamond}_{\mathcal{M}}$, and the prefix '$\mathcal{M}$' from the notions above.

▶ **Definition 2.7.** Let $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$ be a matrix for a language $\mathcal{L}$, and let $\mathcal{L} \subseteq \mathcal{L}'$. A matrix $\mathcal{M}' = \langle \mathcal{V}', \mathcal{D}', \mathcal{O}' \rangle$ for $\mathcal{L}'$ is called an *expansion* of $\mathcal{M}$ to $\mathcal{L}'$, if $\mathcal{V} = \mathcal{V}'$, $\mathcal{D} = \mathcal{D}'$, and $\mathcal{O}'(\diamond) = \mathcal{O}(\diamond)$ for every connective $\diamond$ of $\mathcal{L}$.

▶ **Definition 2.8.** Given a matrix $\mathcal{M}$, the consequence relation $\vdash_{\mathcal{M}}$ that is *induced by* (or associated with) $\mathcal{M}$, is defined by: $\mathcal{T} \vdash_{\mathcal{M}} \psi$ if $mod_{\mathcal{M}}(\mathcal{T}) \subseteq mod_{\mathcal{M}}(\psi)$. We denote by $\mathbf{L}_{\mathcal{M}}$ the pair $\langle \mathcal{L}, \vdash_{\mathcal{M}} \rangle$, where $\mathcal{M}$ is a matrix for $\mathcal{L}$ and $\vdash_{\mathcal{M}}$ is the consequence relation induced by $\mathcal{M}$.

▶ **Theorem 2.9** ([21, 22]). *For every propositional language $\mathcal{L}$ and finite matrix $\mathcal{M}$ for $\mathcal{L}$, $\mathbf{L}_{\mathcal{M}} = \langle \mathcal{L}, \vdash_{\mathcal{M}} \rangle$ is a propositional logic. If $\mathcal{M}$ is finite, then $\vdash_{\mathcal{M}}$ is also finitary.*

We conclude this section with some simple, easily verified properties of the basic connectives (Definition 2.4).

▶ **Definition 2.10.** Let $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$ be a matrix for $\mathcal{L}$ and let $\mathcal{A} \subseteq \mathcal{V}$.

- An n-ary connective $\diamond$ of $\mathcal{L}$ is called $\mathcal{A}$-*closed* if $\tilde{\diamond}(a_1, \ldots, a_n) \in \mathcal{A}$ for every $a_1, \ldots, a_n \in \mathcal{A}$.
- An n-ary connective $\diamond$ of $\mathcal{L}$ is called $\mathcal{A}$-*limited* if for every $a_1, \ldots, a_n \in \mathcal{V}$, if $\tilde{\diamond}(a_1, \ldots, a_n) \in \mathcal{A}$ then $a_1, \ldots, a_n \in \mathcal{A}$.

▶ **Definition 2.11.** Let $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$ be a matrix for $\mathcal{L}$.

- A connective $\wedge$ in $\mathcal{L}$ is called an $\mathcal{M}$-*conjunction* if it is $\mathcal{D}$-closed and $\mathcal{D}$-limited, i.e., for every $a, b \in \mathcal{V}$, $a \wedge b \in \mathcal{D}$ iff $a \in \mathcal{D}$ and $b \in \mathcal{D}$.
- A connective $\vee$ in $\mathcal{L}$ is called an $\mathcal{M}$-*disjunction* if it is $\overline{\mathcal{D}}$-closed and $\overline{\mathcal{D}}$-limited, i.e., for every $a, b \in \mathcal{V}$, $a \vee b \in \mathcal{D}$ iff $a \in \mathcal{D}$ or $b \in \mathcal{D}$.
- A connective $\supset$ in $\mathcal{L}$ is called an $\mathcal{M}$-*implication* if for every $a, b \in \mathcal{V}$, $a \supset b \in \mathcal{D}$ iff either $a \notin \mathcal{D}$ or $b \in \mathcal{D}$.

Using the terminologies in Definitions 2.4 and 2.11, we have:

▶ **Theorem 2.12.** *Let $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$ be a matrix for $\mathcal{L}$.*

1. *A connective is an $\mathcal{M}$-conjunction iff it is a conjunction for $\mathbf{L}_{\mathcal{M}}$.*
2. *An $\mathcal{M}$-disjunction is also a disjunction for $\mathbf{L}_{\mathcal{M}}$.*
3. *An $\mathcal{M}$-implication is also an implication for $\mathbf{L}_{\mathcal{M}}$.*

▶ **Corollary 2.13.** *Let $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$ be a matrix for $\mathcal{L}$, and let $\mathcal{M}'$ be an expansion of $\mathcal{M}$.*

1. *An $\mathcal{M}$-conjunction (respectively: $\mathcal{M}$-disjunction, $\mathcal{M}$-implication) is also a conjunction (respectively: disjunction, implication) for $\mathbf{L}_{\mathcal{M}'}$.*
2. *If $\mathcal{M}$ has either an $\mathcal{M}$-conjunction, or an $\mathcal{M}$-disjunction, or an $\mathcal{M}$-implication, then $\mathbf{L}_{\mathcal{M}'}$ is semi-normal. If $\mathcal{M}$ has all of them then $\mathbf{L}_{\mathcal{M}'}$ is normal.*

3 Paradefinite Logics

In this section we define in precise terms what paradefinite logics are, and consider some related desirable properties.

▶ **Definition 3.1.** Let $\mathcal{L}$ be a propositional language with a unary connective $\neg$, and let $\mathbf{L} = \langle \mathcal{L}, \vdash_{\mathbf{L}} \rangle$ be a logic for $\mathcal{L}$.

- $\mathbf{L}$ is called *pre-$\neg$-paraconsistent* if there are formulas ψ, φ such that $\psi, \neg\psi \nvdash_{\mathbf{L}} \varphi$.
- $\mathbf{L}$ is called *pre-$\neg$-paracomplete* if there is a theory $\mathcal{T}$ and formulas ψ, φ such that $\mathcal{T}, \psi \vdash_{\mathbf{L}} \varphi$ and $\mathcal{T}, \neg\psi \vdash_{\mathbf{L}} \varphi$, but $\mathcal{T} \nvdash_{\mathbf{L}} \varphi$.

The first property above intends to capture the idea that a contradictory set of premises should not entail every formula, and the second property indicates that it may happen that a certain statement and its negation do not hold. Both of these intuitions make sense only if the underlying connective $\neg$ somehow represents a 'negation' operation. This is assured by the condition of 'coherence with classical logic', which is defined next. Intuitively, this condition states that a logic that has such a connective should not admit entailments that do not hold in classical logic.

▶ **Definition 3.2.** Let $\mathcal{L}$ be a language with a unary connective $\neg$. A *bivalent $\neg$-interpretation for $\mathcal{L}$* is a function $\mathbf{F}$ that associates a two-valued truth table with each connective of $\mathcal{L}$, such that $\mathbf{F}(\neg)$ is the classical truth table for negation. We denote by $\mathcal{M}_{\mathbf{F}}$ the two-valued matrix for $\mathcal{L}$ induced by $\mathbf{F}$, that is, $\mathcal{M}_{\mathbf{F}} = \langle \{t, f\}, \{t\}, \mathbf{F} \rangle$ (see Definition 2.5).

▶ **Definition 3.3.** Let $\mathbf{L} = \langle \mathcal{L}, \vdash_{\mathbf{L}} \rangle$ be a propositional logic where $\mathcal{L}$ contains a unary connective $\neg$.

- Let $\mathbf{F}$ be a bivalent $\neg$-interpretation for $\mathcal{L}$. We say that $\mathbf{L}$ is $\mathbf{F}$*-contained in classical logic* if for every $\varphi_1, \ldots, \varphi_n, \psi \in \mathcal{W}(\mathcal{L})$, if $\varphi_1, \ldots \varphi_n \vdash_{\mathbf{L}} \psi$ then $\varphi_1, \ldots, \varphi_n \vdash_{\mathcal{M}_{\mathbf{F}}} \psi$.
- $\mathbf{L}$ is $\neg$*-contained in classical logic* [3], if it is $\mathbf{F}$-contained in it for a bivalent $\neg$-interpretation $\mathbf{F}$.
- $\mathbf{L}$ is $\neg$*-coherent with classical logic*, if it has a semi-normal fragment (Definition 2.4) which is $\neg$-contained in classical logic.

▶ **Definition 3.4.** Let $\mathbf{L} = \langle \mathcal{L}, \vdash_{\mathbf{L}} \rangle$ be a propositional logic where $\mathcal{L}$ contains a unary connective $\neg$. We say that $\neg$ is a *negation* for $\mathbf{L}$, if $\mathbf{L}$ is $\neg$-coherent with classical logic.

▶ Remark. If $\neg$ is a negation for $\mathbf{L} = \langle \mathcal{L}, \vdash_{\mathbf{L}} \rangle$, then for every atom $p \in \mathsf{Atoms}(\mathcal{L})$ it holds that $p \nvdash_{\mathbf{L}} \neg p$ and $\neg p \nvdash_{\mathbf{L}} p$.

▶ **Definition 3.5.** Let $\mathcal{L}$ be a language with a unary connective $\neg$, and $\mathbf{L} = \langle \mathcal{L}, \vdash_{\mathbf{L}} \rangle$ a logic for $\mathcal{L}$.

- $\mathbf{L}$ is called $\neg$*-paraconsistent* if it is pre-$\neg$-paraconsistent and $\neg$ is a negation of $\mathbf{L}$.
- $\mathbf{L}$ is called $\neg$*-paracomplete* if it is pre-$\neg$-paracomplete and $\neg$ is a negation of $\mathbf{L}$.
- $\mathbf{L}$ is called $\neg$*-paradefinite* if it is $\neg$-paraconsistent and $\neg$-paracomplete.

Henceforth we shall frequently omit the $\neg$ sign (if it is clear from the context), and simply refer to paradefinite [paraconsistent, paracomplete] logics.

4 Four-Valued Paradefinite Matrices

We now show that the availability of at least four different truth values is needed for developing paradefinite logics in the framework of matrices. We then characterize the structure of four-valued paradefinite matrices.

In what follows we suppose that $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$ is a matrix for a language with $\neg$. We say that $\mathcal{M}$ is paradefinite [paraconsistent, paracomplete] if so is $\mathbf{L}_{\mathcal{M}}$ (Definition 2.8).

▶ **Theorem 4.1.**

1. *$\mathcal{M}$ is pre-paraconsistent iff there is an element $\top \in \mathcal{D}$, such that $\tilde{\neg}\top \in \mathcal{D}$.*
2. *If $\mathcal{M}$ is paraconsistent then there are three different elements t, f, and $\top$ in $\mathcal{V}$ such that $f = \tilde{\neg}t$, $f \notin \mathcal{D}$, and $\{t, \tilde{\neg}f, \top, \tilde{\neg}\top\} \subseteq \mathcal{D}$.*

Proof. $\mathcal{M}$ is pre-paraconsistent iff $p, \neg p \nvdash_{\mathcal{M}} q$. Obviously, this happens iff $\{p, \neg p\}$ has an $\mathcal{M}$-model. The latter, in turn, is possible iff there is some $\top \in \mathcal{D}$, such that $\tilde{\neg}\top \in \mathcal{D}$, as indicated in the first item of the theorem. For the second item we may assume without loss of generality that $\mathcal{M}$ is $\neg$-contained in classical logic. We let $\mathbf{F}$ be a bivalent $\neg$-interpretation such that $\mathbf{L}_{\mathcal{M}}$ is $\mathbf{F}$-contained in classical logic. Since $p, \neg\neg p \nvdash_{\mathcal{M}_{\mathbf{F}}} \neg p$, also $p, \neg\neg p \nvdash_{\mathcal{M}} \neg p$, and so there is some $t \in \mathcal{D}$, such that $\tilde{\neg}t \notin \mathcal{D}$, while $\tilde{\neg}\tilde{\neg}t \in \mathcal{D}$. Let $f = \tilde{\neg}t$. Then t and f have the required properties, and together with the first item we are done. ◀

▶ **Theorem 4.2.**

1. *If $\mathcal{M}$ is pre-paracomplete then there is an element $\bot \in \mathcal{V}$ such that $\bot \notin \mathcal{D}$ and $\tilde{\neg}\bot \notin \mathcal{D}$.*
2. *If $\mathcal{M}$ has an $\mathcal{M}$-disjunction and there is an element $\bot \in \mathcal{V}$ such that $\bot \notin \mathcal{D}$ and $\tilde{\neg}\bot \notin \mathcal{D}$, then $\mathcal{M}$ is pre-paracomplete.*

Proof. Suppose first that $\mathcal{M}$ is pre-paracomplete. Then there is a set of formulas Γ and formulas ψ, ϕ, such that (i) $\Gamma, \psi \vdash_{\mathcal{M}} \phi$, (ii) $\Gamma, \neg\psi \vdash_{\mathcal{M}} \phi$, and (iii) $\Gamma \nvdash_{\mathcal{M}} \phi$. From (iii) it follows that there is a valuation $\nu \in mod_{\mathcal{M}}(\Gamma) \setminus mod_{\mathcal{M}}(\phi)$. Thus, in order to satisfy conditions (i) and (ii), necessarily $\nu(\psi) \notin \mathcal{D}$ and $\tilde{\neg}\nu(\psi) = \nu(\neg\psi) \notin \mathcal{D}$. Hence $\nu(\psi)$ is the element $\bot$ as required.

For the second item, suppose that its two conditions are satisfied. Let $\vee$ be an $\mathcal{M}$-disjunction. Then by Theorem 2.12, $p \vdash_{\mathcal{M}} \neg p \vee p$ and $\neg p \vdash_{\mathcal{M}} \neg p \vee p$. However, if $\nu(p) = \bot$ then $\nu(\neg p \vee p) \notin \mathcal{D}$ by the definitions of $\bot$ and of an $\mathcal{M}$-disjunction. Hence $\mathcal{M}$ is pre-paracomplete. ◀

By the theorems above, no 2-valued matrix can be paraconsistent or paracomplete, and no 3-valued matrix can be paradefinite. Also, by Theorem 4.1, every paraconsistent (and so every paradefinite) matrix should have at least two designated elements. The structures of the minimally-valued paradefinite matrices is considered next.

▶ **Theorem 4.3.** *If $\mathcal{M} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$ is a $\neg$-paradefinite matrix then there are four elements t,f,$\top$, and $\bot$ in $\mathcal{V}$ such that: (1) $t \in \mathcal{D}$ and $\tilde{\neg}t \notin \mathcal{D}$, (2) $f \notin \mathcal{D}$ and $\tilde{\neg}f \in \mathcal{D}$, (3) $\top \in \mathcal{D}$ and $\tilde{\neg}\top \in \mathcal{D}$, (4) $\bot \notin \mathcal{D}$ and $\tilde{\neg}\bot \notin \mathcal{D}$, (5) $\tilde{\neg}t = f$.*

Proof. This follows from Theorems 4.1 and 4.2. ◀

▶ **Corollary 4.4.** *Let $\mathcal{M}$ be a $\neg$-paradefinite four-valued matrix. Then $\mathcal{M}$ is isomorphic to a matrix of the form $\mathcal{M}' = \langle \{t, f, \top, \bot\}, \{t, \top\}, \mathcal{O} \rangle$, in which $\tilde{\neg}t = f$, $\tilde{\neg}f = t$, $\tilde{\neg}\top \in \{t, \top\}$, and $\tilde{\neg}\bot \in \{f, \bot\}$.*

In the rest of this paper we shall assume that the 4-valued matrices we study have the form described in Corollary 4.4,

5 Dunn-Belnap's Matrix $\mathcal{FOUR}$

Theorem 4.4 leaves exactly four possible interpretations for $\neg$ in four-valued paradefinite matrices. However, the next theorem and its corollary show that the Dunn-Belnap negation ([9, 10, 14, 15]) is by far more natural than the others.[1]

▶ **Theorem 5.1.** *Let $\mathcal{M}$ be a $\neg$-paradefinite 4-valued matrix. Then:*

1. *If $\neg$ is left involutive for $\mathbf{L}_{\mathcal{M}}$ (that is, $\neg\neg p \vdash_{\mathbf{L}_{\mathcal{M}}} p$) then $\neg\bot = \bot$.*
2. *If $\neg$ is right involutive for $\mathbf{L}_{\mathcal{M}}$ (that is, $p \vdash_{\mathbf{L}_{\mathcal{M}}} \neg\neg p$) then $\neg\top = \top$.*

Proof. Suppose that $\neg$ is left involutive. Then $\neg\neg p \vdash_{\mathcal{M}} p$, and so $\neg\bot \neq f$ (otherwise, by Corollary 4.4 $\nu(p) = \bot$ would have been a counter-model). It follows that $\neg\bot = \bot$. Suppose now that $\neg$ is right involutive. Then $p \vdash_{\mathcal{M}} \neg\neg p$, and so $\neg\top \neq t$ (otherwise, by Corollary 4.4 again, $\nu(p) = \top$ would have been a counter-model). Thus $\neg\top = \top$. ◀

▶ **Corollary 5.2.** *The only involutive negation of paradefinite 4-valued logics is Dunn-Belnap negation, defined by $\neg t = f$, $\neg f = t$, $\neg\top = \top$ and $\neg\bot = \bot$.*

Concerning the interpretations of the other connectives, we again follow Belnap's motivation in [9] and [10], where he suggested a four-valued framework for collecting and processing information (this work was later generalized in [7]): Assume a set of sources, each one of them can indicate that an atom p is true (i.e., it assigns p the truth-value 1), false (i.e., it assigns p the truth-value 0), or that it has no knowledge about p. In turn, a mediator assigns

[1] For convenience, we shall denote the interpretation of $\neg$ by $\neg$ as well. A similar convention will be usually used for any other connective.

to an atomic formula p a subset $d(p)$ of $\{0,1\}$ as follows: $1 \in d(p)$ iff some source claims that p is true, and $0 \in d(p)$ iff some source claims that p is false. The mediator's evaluation of complex formulas over $\{\neg, \vee\}$ is then derived as follows:

$0 \in d(\neg\varphi)$ iff $1 \in d(\varphi)$,
$1 \in d(\neg\varphi)$ iff $0 \in d(\varphi)$,
$1 \in d(\varphi \vee \psi)$ iff $1 \in d(\varphi)$ or $1 \in d(\psi)$,
$0 \in d(\varphi \vee \psi)$ iff $0 \in d(\varphi)$ and $0 \in d(\psi)$.

In this model, $\nu(\varphi) = \{0,1\}$ means that φ is known to be true and also known to be false (i.e., the information about φ is inconsistent). $\nu(\varphi) = \{1\}$ means that φ is only known to be true, while $\nu(\varphi) = \{0\}$ means that φ is only known to be false. Finally, $\nu(\varphi) = \emptyset$ means that there is no information about φ. This observation leads to the following identification of the four truth-values with the subsets of $\{0,1\}$: $t = \{1\}, f = \{0\}, \top = \{0,1\}, \bot = \emptyset$.

Accordingly, the truth tables for $\neg$ and $\vee$ that the above principles lead to are the following (where the connective $\wedge$ is defined by: $\varphi \wedge \psi =_{Df} \neg(\neg\varphi \vee \neg\psi)$):

$\tilde{\vee}$	t	f	$\top$	$\bot$
t	t	t	t	t
f	t	f	$\top$	$\bot$
$\top$	t	$\top$	$\top$	t
$\bot$	t	$\bot$	t	$\bot$

$\tilde{\wedge}$	t	f	$\top$	$\bot$
t	t	f	$\top$	$\bot$
f	f	f	f	f
$\top$	$\top$	f	$\top$	f
$\bot$	$\bot$	f	f	$\bot$

$\tilde{\neg}$	
t	f
f	t
$\top$	$\top$
$\bot$	$\bot$

▶ **Definition 5.3.** The *Dunn-Belnap basic matrix* for the language $\mathcal{L}_{\mathcal{FOUR}} = \{\neg, \vee, \wedge\}$ (or just $\{\neg, \vee\}$) is the matrix $\mathcal{FOUR} = \langle \mathcal{V}, \mathcal{D}, \mathcal{O} \rangle$, where $\mathcal{V} = \{t, f, \top, \bot\}$, $\mathcal{D} = \{t, \top\}$, and the interpretations of the connectives are given by the truth tables above.

▶ Remark. Another, dual representation of $\mathcal{FOUR}$ uses pairs from $\{1,0\} \times \{1,0\}$. Given such a pair $\langle a, b \rangle$, the first component intuitively represents the information about the truth of a formula, and the second one represents the information about its falsity. According to this representation, we have that $t = \langle 1, 0 \rangle$, $f = \langle 0, 1 \rangle$, $\top = \langle 1, 1 \rangle$, $\bot = \langle 0, 0 \rangle$, $\langle a_1, b_1 \rangle \vee \langle a_2, b_2 \rangle = \langle max(a_1, b_1), min(a_2, b_2) \rangle$, $\langle a_1, b_1 \rangle \wedge \langle a_2, b_2 \rangle = \langle min(a_1, b_1), max(a_2, b_2) \rangle$, and $\neg\langle a, b \rangle = \langle b, a \rangle$. This representation is useful for a number of applications (see, e.g., [1, 4, 8, 17]).

A common way of defining and understanding the disjunction, conjunction and negation of $\mathcal{FOUR}$ is with respect to the partial order $\leq_t$ on $\{t, f, \top, \bot\}$, in which t is the maximal element, f is the minimal element, and $\top, \bot$ are intermediate $\leq_t$-incomparable elements. This order may be intuitively understood as reflecting differences in the amount of *truth* that each element exhibits. Here, $\tilde{\wedge}$ and $\tilde{\vee}$ are the meet and the join (respectively) of $\leq_t$, and $\tilde{\neg}$ is order reversing with respect to $\leq_t$. Note that this interpretation of $\neg$ coincides with that of the unique involutive negation of paradefinite four-valued logics given in Corollary 5.2.

For characterizing the expressive power of the languages of $\mathcal{FOUR}$ it is convenient to order the truth-values in the partial order $\leq_k$ that intuitively reflects differences in the amount of *knowledge* (or *information*) that the truth values convey. According to this relation $\top$ is the maximal element, $\bot$ is the minimal element, and t, f are intermediate $\leq_k$-incomparable elements.

Together, the lattices $\langle \{t, f, \top, \bot\}, \leq_t \rangle$ and $\langle \{t, f, \top, \bot\}, \leq_k \rangle$ form a single four-valued structure (denoted again by $\mathcal{FOUR}$), known as Belnap's bilattice ([9, 10]), which is represented in the double-Hasse diagram of Figure 1.

Following Fitting's notations (see [16]), we shall denote the join and the meet of $\leq_k$ by $\oplus$ and $\otimes$ (respectively). The $\leq_k$-reversing function on $\{t, f, \top, \bot\}$ which is dual to $\tilde{\neg}$ is called *conflation* [16], and the corresponding connective is usually denoted by $-$. The truth tables of these $\leq_k$-connectives are given below.

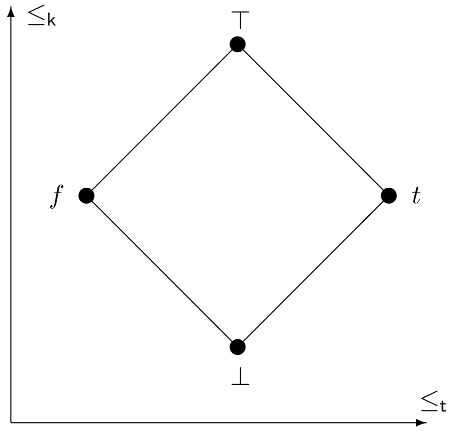


Figure 1 The bilattice $\mathcal{FOUR}$

$\tilde{\oplus}$	t	f	$\top$	$\bot$
t	t	$\top$	$\top$	t
f	$\top$	f	$\top$	f
$\top$	$\top$	$\top$	$\top$	$\top$
$\bot$	t	f	$\top$	$\bot$

$\tilde{\otimes}$	t	f	$\top$	$\bot$
t	t	$\bot$	t	$\bot$
f	$\bot$	f	f	$\bot$
$\top$	t	f	$\top$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

$\tilde{\neg}$	
t	t
f	f
$\top$	$\bot$
$\bot$	$\top$

6 Important Expansions of $\mathcal{FOUR}$

As noted before, the logic $\mathbf{L}_{\mathcal{FOUR}}$ (also denoted **4Basic**), induced by $\mathcal{FOUR}$, has some appealing applications in the context of logics for AI. Also, it has some desirable properties, like being semi-normal (it is easy to verify that $\vee$ is a $\mathcal{FOUR}$-disjunction and that $\wedge$ is a $\mathcal{FOUR}$-conjunction), paradefinite, and $\neg$-contained in classical logic. However, **4Basic** also has some drawbacks, one of which is considered next.

▶ **Theorem 6.1.** **4Basic** *is not normal (since no implication is definable in it).*

Sketch of proof. Note, first, that every n-valued function $g : \{t, f, \top, \bot\}^n \to \{t, f, \top, \bot\}$ which is representable in the language of $\{\vee, \wedge, \neg\}$ [2] must be $\leq_k$-monotonic, i.e., if $a_i \leq_k b_i$ for every $1 \leq i \leq n$ then $g(a_1, \ldots, a_n) \leq_k g(b_1, \ldots, b_n)$ as well. This implies that only $\leq_k$-monotonic connectives are definable in this language. Now, suppose for contradiction that $\supset$ is a definable implication for **4Basic**. In particular, one may verify that (i) $\vdash_{\mathbf{4Basic}} p \supset p$, and (ii) $p, p \supset q \vdash_{\mathbf{4Basic}} q$. Now, (i) entails that $\tilde{\supset}(f, f) \in \{t, \top\}$. Therefore, it follows from the $\leq_k$-monotonicity of $\supset$ that $\tilde{\supset}(\top, f) \in \{t, \top\}$. This contradicts (ii), since it is refuted by any assignment ν such that $\nu(p) = \top$ and $\nu(q) = f$. ◀

The last theorem, together with the fact that definable functions in the language of $\{\vee, \wedge, \neg\}$ are $\{\bot\}$-closed (and so no tautologies are available in this language), imply that the language of $\mathcal{FOUR}$ is rather limited, even if we add to it propositional constants for

[2] That is, there is a formula ψ in $\{\vee, \wedge, \neg\}$, such that $\mathsf{Atoms}(\psi) \subseteq \{P_1, \ldots, P_n\}$, and for every $a_1, \ldots, a_n \in \{t, f, \top, \bot\}$ it holds that $g(a_1, \ldots, a_n) = \nu(\psi)$, where $\nu \in \Lambda_{\mathcal{FOUR}}$ is defined by $\nu(P_i) = a_i$ for all $1 \leq i \leq n$.

the two classical truth-values. Therefore, we now introduce several other useful and natural connectives on $\{t, f, \top, \bot\}$ that cannot be defined in the language of $\mathcal{FOUR}$.

- The following connective is an $\mathcal{M}$-implication for every paradefinite four-valued matrix $\mathcal{M}$ (of the form considered in Corollary 4.4), in which it is definable:

$$a \tilde{\supset} b = \begin{cases} b & \text{if } a \in \{t, \top\}, \\ t & \text{if } a \in \{f, \bot\}. \end{cases}$$

- We denote by $\mathsf{t}, \mathsf{f}, \mathsf{c}$ (contradictory) and u (unknown) the propositional constants to be interpreted, respectively, by the truth-values t, f, $\top$, and $\bot$ (thus, for instance, $\forall \nu \in \Lambda_{\mathcal{M}}\ \nu(\mathsf{c}) = \top$).

Using the connectives above, in the following sections we shall consider some important expansions of the matrix $\mathcal{FOUR}$.[3]

6.1 A Maximal Expansion

First, we consider expansions of $\mathcal{FOUR}$ in which all the operations on $\{t, f, \top, \bot\}$ are definable.

▶ **Definition 6.2.** Let $\mathcal{L}_{All} = \{\neg, \vee, \wedge, -, \oplus, \otimes, \supset, \mathsf{f}, \mathsf{t}, \mathsf{c}, \mathsf{u}\}$. The matrix $\mathcal{M}_{All}$ is the expansion of $\mathcal{FOUR}$ to $\mathcal{L}_{All}$. The logic that is induced by $\mathcal{M}_{All}$ is denoted by **4All** (or, as before, $\mathbf{L}_{\mathcal{M}_{All}}$).

As the next theorem shows, the set of connectives in $\mathcal{L}_{All}$ (and actually a proper subset of it) is indeed sufficient for defining any operation on $\{t, f, \top, \bot\}$.

▶ **Theorem 6.3.** *The language of* $\{\neg, \vee, \wedge, \supset, \mathsf{c}, \mathsf{u}\}$ *is functionally complete for* $\{t, f, \top, \bot\}$.

▶ Remark. Since $\bot = f \otimes \neg f$ while $\top = f \oplus \neg f$, the language of $\{\neg, \vee, \wedge, \supset, \otimes, \oplus, \mathsf{f}\}$ is also functionally complete for $\{t, f, \top, \bot\}$. The use of this language has a certain advantage of modularity over the use of $\{\neg, \vee, \wedge, \supset, \mathsf{c}, \mathsf{u}\}$, since it has been proved in [6] that if Ξ is a subset of $\{\otimes, \oplus, \mathsf{f}\}$, then a function $g : \{t, f, \top, \bot\}^n \to \{t, f, \top, \bot\}$ is representable in $\{\neg, \wedge, \supset\} \cup \Xi$ iff it is S-closed for every $S \in \{\{\top\}, \{t, f, \top\}, \{t, f, \bot\}\}$ for which all the (functions that directly correspond to the) connectives in Ξ are S-closed. In other words:

▶ Theorem 6.4.

- *g is representable in $\{\neg, \vee, \wedge, \supset\}$ iff it is $\{\top\}$-closed, $\{t, f, \bot\}$-closed, and $\{t, f, \top\}$-closed.*
- *g is representable in $\{\neg, \vee, \wedge, \supset, \mathsf{f}\}$ iff it is $\{t, f, \bot\}$-closed and $\{t, f, \top\}$-closed.*
- *g is representable in $\{\neg, \vee, \wedge, \supset, \oplus\}$ iff it is $\{\top\}$-closed and $\{t, f, \top\}$-closed.*
- *g is representable in $\{\neg, \vee, \wedge, \supset, \otimes\}$ iff it is $\{\top\}$-closed and $\{t, f, \bot\}$-closed.*
- *g is representable in $\{\neg, \vee, \wedge, \supset, \otimes, \mathsf{f}\}$ iff it is $\{t, f, \bot\}$-closed.*
- *g is representable in $\{\neg, \vee, \wedge, \supset, \oplus, \otimes\}$ iff it is $\{\top\}$-closed.*
- *g is representable in $\{\neg, \vee, \wedge, \supset, \oplus, \mathsf{f}\}$ iff it is $\{t, f, \top\}$-closed.*
- *g is representable in $\{\neg, \vee, \wedge, \supset, \oplus, \otimes, \mathsf{f}\}$.*

[3] Due to lack of space, proofs in the rest of this paper are omitted. Complete proofs will be provided in the full version of the paper.

It is also worth noting that it is easy to find examples that show that the eight fragments in the theorem above are different from each other (see [2] and [6]).

Note that $\mathcal{M}_{All}$, like any other 4-valued matrix where the $\leq_{\mathsf{k}}$-meet $\otimes$, the $\leq_{\mathsf{k}}$-join $\oplus$, or either of the propositional constants c and u is definable in its language, is not $\{t, f\}$-closed (indeed, $a \oplus b \notin \{t, f\}$ and $a \otimes b \notin \{t, f\}$ for any $a \neq b \in \{t, f\}$). This implies that **4All** is only $\neg$-*coherent* with classical logic but not $\neg$-*included* in it.

▶ **Theorem 6.5.** *The logic* **4All** *is paradefinite and normal.*

The next theorem follows from the fact that **4All** has no proper extensions (in the same language).

▶ **Theorem 6.6.** *The logic* **4All** *(unlike the logic* **4Basic**!*) is* maximally paraconsistent *in the sense that every proper extension of* **4All** *(Definition 2.3) is not pre-paraconsistent.*

6.2 A Maximal Monotonic Expansion

In [10] Belnap suggested to use the sources-mediator model described previously only for languages with monotonic interpretations of the connectives. The reason was to achieve stability in the sense that the arrival of new data from new sources does not change previous knowledge about truth and falsity. From Belnap's point of view an optimal language for information processing is therefore a language in which it is possible to represent *all* monotonic functions, and only monotonic functions. Next we show that not much should be added to the basic language of $\{\neg, \vee, \wedge\}$ (or just $\{\neg, \vee\}$) in order to obtain such a language.

▶ **Definition 6.7.** Let $\mathcal{L}_{Mon} = \{\neg, \vee, \wedge, \mathsf{c}, \mathsf{u}\}$. We denote by $\mathcal{M}_{Mon}$ the expansion of $\mathcal{FOUR}$ to $\mathcal{L}_{Mon}$. The logic that is induced by $\mathcal{M}_{Mon}$ is denoted by **4Mon**.

▶ **Theorem 6.8.** *A function $g : \{t, f, \top, \bot\}^n \to \{t, f, \top, \bot\}$ is representable in $\mathcal{L}_{Mon}$ iff it is $\leq_{\mathsf{k}}$-monotonic (i.e., if $a_i \leq_{\mathsf{k}} b_i$ for every $1 \leq i \leq n$ then $g(a_1, \ldots, a_n) \leq_{\mathsf{k}} g(b_1, \ldots, b_n)$).*

▶ **Corollary 6.9.** *The logic* **4Mon** *contains every logic which is induced by a matrix of the form of Corollary 4.4 that employs only monotonic functions.*

▶ **Theorem 6.10.** *The logic* **4Mon** *is paradefinite. It has no proper extensions in its language, and so it is maximally paraconsistent (see Theorem 6.6).*

6.3 A Maximal Classically Closed Expansion

We now examine the maximal expansions of $\mathcal{FOUR}$ by connectives that are $\{t, f\}$-closed.

▶ **Definition 6.11.** Let $\mathcal{L}_{CC} = \{\neg, -, \vee, \wedge, \supset\}$. We denote by $\mathcal{M}_{CC}$ the expansion of $\mathcal{FOUR}$ to $\mathcal{L}_{CC}$. The logic that is induced by $\mathcal{M}_{CC}$ is denoted by **4CC**.

▶ **Theorem 6.12.** *A function $g : \{t, f, \top, \bot\}^n \to \{t, f, \top, \bot\}$ is representable in $\mathcal{L}_{CC}$ iff it is $\{t, f\}$-closed.*

▶ **Corollary 6.13.** *The logic* **4CC** *contains every logic which is induced by a matrix of the form of Corollary 4.4 and is $\neg$-contained in classical logic.*

▶ **Theorem 6.14.** *The logic* **4CC** *is paradefinite and normal. It is $\neg$-contained in classical logic and has no proper extensions in its language, thus it is maximally paraconsistent (in the sense described in Theorem 6.6).*

We note, in addition to the properties considered in Proposition 6.14, that **4CC** is also maximal relative to classical logic. This means, intuitively, that any attempt to add to it a tautology of classical logic which is not provable in **4CC** should necessarily end-up with classical logic (see [3] for the exact definition of this property).

6.4 A Maximal Non-Exploding Expansion

Next, we consider the maximal expansions of $\mathcal{FOUR}$ which are non-exploding in the following sense:

▶ **Definition 6.15.** A logic $\langle \mathcal{L}, \vdash \rangle$ is *non-exploding*, if for every theory $\mathcal{T}$ in $\mathcal{L}$ such that $\mathsf{Atoms}(\mathcal{T}) \neq \mathsf{Atoms}(\mathcal{L})$ there is a formula ψ in $\mathcal{L}$ such that $\mathcal{T} \not\vdash \psi$.

▶ **Definition 6.16.** Let $\mathcal{L}_{Nex} = \{\neg, \vee, \wedge, \supset, \oplus, \otimes\}$. We denote by $\mathcal{M}_{Nex}$ the expansion of $\mathcal{FOUR}$ to $\mathcal{L}_{Nex}$. The logic that is induced by $\mathcal{M}_{Nex}$ is denoted by **4Nex**.

▶ **Theorem 6.17.** *A function $g : \{t, f, \top, \bot\}^n \to \{t, f, \top, \bot\}$ is representable in $\mathcal{L}_{Nex}$ iff it is $\{\top\}$-closed.*

▶ **Corollary 6.18.** *The logic* **4Nex** *contains every logic which is induced by a matrix of the form of Corollary 4.4 and is non-exploding.*

▶ **Theorem 6.19.** *The logic* **4Nex** *is paradefinite. It is non-exploding but not $\neg$-contained in classical logic. Also,* **4Nex** *has no proper extensions in its language, thus it is maximally paraconsistent.*

6.5 A Maximal Flexible Expansion

The combination of $\{t, f, \top\}$-closure and $\{t, f, \bot\}$-closure is a very desirable property, since it allows flexibility in the use of the four basic truth-values. Obviously, there is no point in using c in case no contradiction is expected, while in the dual case there is no point in using u. The use of connectives which have both of the above properties ensures that one can easily switch from the use of the four-valued framework to the use of the appropriate 3-valued framework. Also, this combination is a natural strengthening of the condition of classical closure. These considerations motivate the four-valued logic introduced next.

▶ **Definition 6.20.** A function $g : \{t, f, \top, \bot\}^n \to \{t, f, \top, \bot\}$ is called *flexible* iff it is both $\{t, f, \top\}$-closed and $\{t, f, \bot\}$-closed.

Obviously, every flexible function is classically closed, but the converse is not true.

▶ **Definition 6.21.** Let $\mathcal{L}_{Flex} = \{\neg, \vee, \wedge, \supset, \mathsf{f}\}$. We denote by $\mathcal{M}_{Flex}$ the expansion of $\mathcal{FOUR}$ to $\mathcal{L}_{Flex}$. The logic that is induced by $\mathcal{M}_{Flex}$ is denoted by **4Flex**.

▶ **Theorem 6.22.** *A function $g : \{t, f, \top, \bot\}^n \to \{t, f, \top, \bot\}$ is representable in $\mathcal{L}_{Flex}$ iff it is flexible.*

▶ **Corollary 6.23.** *The logic* **4Flex** *contains every logic that is induced by a matrix of the form of Corollary 4.4 and employs only flexible connectives.*

▶ **Theorem 6.24.** *The logic* **4Flex** *is a paradefinite and normal. It is $\neg$-contained in classical logic and not non-exploding. This logic is neither maximally paraconsistent nor maximally paraconsistent relative to classical logic.*

6.6 The Classical Expansion

The last expansion of $\mathcal{FOUR}$ we present is the maximal one which is *both* non-explosive and flexible.

▶ **Definition 6.25.** Let $\mathcal{L}_{CL} = \{\neg, \vee, \wedge, \supset\}$. We denote by $\mathcal{M}_{4CL}$ is the expansion of $\mathcal{FOUR}$ to $\mathcal{L}_{CL}$. The logic that is induced by $\mathcal{M}_{4CL}$ is denoted by **4CL**.

▶ **Theorem 6.26.** *A function $g : \{t, f, \top, \bot\}^n \to \{t, f, \top, \bot\}$ is representable in $\mathcal{L}_{CL}$ iff it is flexible and $\{\top\}$-closed.*

▶ **Corollary 6.27.** *The logic* **4CL** *contains every non-exploding logic which is induced by a matrix of the form of Corollary 4.4 and employs only flexible connectives.*

▶ **Theorem 6.28.** *The logic* **4CL** *is paradefinite and normal. It is $\neg$-contained in classical logic and non-exploding. This logic is neither maximally paraconsistent nor maximally paraconsistent relative to classical logic.*

7 Proof Theory

We conclude by considering proof systems for the $\neg$-paradefinite logics presented in this paper.

7.1 Gentzen-type Systems

First, we consider Gentzen-type systems [18]. We show that each of the logics considered here has a corresponding cut-free, sound and complete sequent calculus, which is a fragment of the sequent calculus $G_{\mathbf{4All}}$, presented in Figure 2.

For each $\mathbf{L} \in \{\mathbf{4All}, \mathbf{4Mon}, \mathbf{4CC}, \mathbf{4Nex}, \mathbf{4Flex}, \mathbf{4CL}, \mathbf{4Basic}\}$ we denote by $G_{\mathbf{L}}$ the restriction of $G_{\mathbf{4All}}$ to the language of $\mathbf{L}$ (i.e., the Gentzen-type system in the language of $\mathbf{L}$ whose axioms and rules are the axioms and rules of $G_{\mathbf{4All}}$ which are relevant to that language). Also, we denote by $\vdash_{G_{\mathbf{L}}}$ the consequence relation induced by $G_{\mathbf{L}}$, that is: $\mathcal{T} \vdash_{G_{\mathbf{L}}} \varphi$, if there exists a finite $\Gamma \subseteq \mathcal{T}$ such that $\Gamma \Rightarrow \varphi$ is provable in $G_{\mathbf{L}}$ from the empty set of sequents (see, e.g., [23] and [24]).

▶ **Theorem 7.1.** *For each $\mathbf{L} \in \{\mathbf{4All}, \mathbf{4Mon}, \mathbf{4CC}, \mathbf{4Nex}, \mathbf{4Flex}, \mathbf{4CL}, \mathbf{4Basic}\}$ $G_{\mathbf{L}}$ is sound and complete for* **L***: $\mathcal{T} \vdash_{G_{\mathbf{L}}} \psi$ iff $\mathcal{T} \vdash_{\mathbf{L}} \psi$. Moreover, $G_{\mathbf{L}}$ admits cut-elimination.*

7.2 Hilbert-type Systems

Next, we consider sound and complete Hilbert-type systems for $\neg$-paradefinite logics which have an implication connective. Again, we show that these are fragments of the same proof system, which has Modus Ponens [MP] as its sole rule of inference.

Consider the proof system $H_{\mathbf{4All}}$ in Figure 3. For $\mathbf{L} \in \{\mathbf{4All}, \mathbf{4CC}, \mathbf{4Nex}, \mathbf{4Flex}, \mathbf{4CL}\}$ we denote by $H_{\mathbf{L}}$ the restriction of $H_{\mathbf{4All}}$ to the language of $\mathbf{L}$ (i.e., the Hilbert-type system in the language of $\mathbf{L}$ whose axioms and rules are the axioms and rules of $H_{\mathbf{4All}}$ which are relevant to that language). We denote by $\vdash_{H_{\mathbf{L}}}$ the consequence relation induced by $H_{\mathbf{L}}$.

▶ **Theorem 7.2.** *For every $\mathbf{L} \in \{\mathbf{4All}, \mathbf{4CC}, \mathbf{4Nex}, \mathbf{4Flex}, \mathbf{4CL}\}$ we have that $\vdash_{H_{\mathbf{L}}} = \vdash_{G_{\mathbf{L}}}$.*

By Theorems 7.1 and 7.2 we also have the following result.

Axioms: $\psi \Rightarrow \psi$

Structural Rules:

Weakening: $\dfrac{\Gamma \Rightarrow \Delta}{\Gamma, \Gamma' \Rightarrow \Delta, \Delta'}$ Cut: $\dfrac{\Gamma_1 \Rightarrow \Delta_1, \psi \quad \Gamma_2, \psi \Rightarrow \Delta_2}{\Gamma_1, \Gamma_2 \Rightarrow \Delta_1, \Delta_2}$

Logical Rules:

[∧⇒]	$\dfrac{\Gamma, \psi, \varphi \Rightarrow \Delta}{\Gamma, \psi \wedge \varphi \Rightarrow \Delta}$	[⇒∧]	$\dfrac{\Gamma \Rightarrow \Delta, \psi \quad \Gamma \Rightarrow \Delta, \varphi}{\Gamma \Rightarrow \Delta, \psi \wedge \varphi}$
[∨⇒]	$\dfrac{\Gamma, \psi \Rightarrow \Delta \quad \Gamma, \varphi \Rightarrow \Delta}{\Gamma, \psi \vee \varphi \Rightarrow \Delta}$	[⇒∨]	$\dfrac{\Gamma \Rightarrow \Delta, \psi, \varphi}{\Gamma \Rightarrow \Delta, \psi \vee \varphi}$
[⊃⇒]	$\dfrac{\Gamma \Rightarrow \psi, \Delta \quad \Gamma, \varphi \Rightarrow \Delta}{\Gamma, \psi \supset \varphi \Rightarrow \Delta}$	[⇒⊃]	$\dfrac{\Gamma, \psi \Rightarrow \varphi, \Delta}{\Gamma \Rightarrow \psi \supset \varphi, \Delta}$
[⊗⇒]	$\dfrac{\Gamma, \psi, \phi \Rightarrow \Delta}{\Gamma, \psi \otimes \phi \Rightarrow \Delta}$	[⇒⊗]	$\dfrac{\Gamma \Rightarrow \Delta, \psi \quad \Gamma \Rightarrow \Delta, \phi}{\Gamma \Rightarrow \Delta, \psi \otimes \phi}$
[⊕⇒]	$\dfrac{\Gamma, \psi \Rightarrow \Delta \quad \Gamma, \phi \Rightarrow \Delta}{\Gamma, \psi \oplus \phi \Rightarrow \Delta}$	[⇒⊕]	$\dfrac{\Gamma \Rightarrow \Delta, \psi, \phi}{\Gamma \Rightarrow \Delta, \psi \oplus \phi}$
[¬¬⇒]	$\dfrac{\Gamma, \varphi \Rightarrow \Delta}{\Gamma, \neg\neg\varphi \Rightarrow \Delta}$	[⇒¬¬]	$\dfrac{\Gamma \Rightarrow \Delta, \varphi}{\Gamma \Rightarrow \Delta, \neg\neg\varphi}$
[−⇒]	$\dfrac{\Gamma, \Rightarrow \Delta, \neg\psi}{\Gamma, -\psi \Rightarrow \Delta}$	[⇒−]	$\dfrac{\Gamma, \neg\psi \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, -\psi}$
[¬−⇒]	$\dfrac{\Gamma, \Rightarrow \Delta, \psi}{\Gamma, \neg-\psi \Rightarrow \Delta}$	[⇒¬−]	$\dfrac{\Gamma, \psi \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg-\psi}$
[¬∧⇒]	$\dfrac{\Gamma, \neg\varphi \Rightarrow \Delta \quad \Gamma, \neg\psi \Rightarrow \Delta}{\Gamma, \neg(\varphi \wedge \psi) \Rightarrow \Delta}$	[⇒¬∧]	$\dfrac{\Gamma \Rightarrow \Delta, \neg\varphi, \neg\psi}{\Gamma \Rightarrow \Delta, \neg(\varphi \wedge \psi)}$
[¬∨⇒]	$\dfrac{\Gamma, \neg\varphi, \neg\psi \Rightarrow \Delta}{\Gamma, \neg(\varphi \vee \psi) \Rightarrow \Delta}$	[⇒¬∨]	$\dfrac{\Gamma \Rightarrow \Delta, \neg\varphi \quad \Gamma \Rightarrow \Delta, \neg\psi}{\Gamma \Rightarrow \Delta, \neg(\varphi \vee \psi)}$
[¬⊃⇒]	$\dfrac{\Gamma, \varphi, \neg\psi \Rightarrow \Delta}{\Gamma, \neg(\varphi \supset \psi) \Rightarrow \Delta}$	[⇒¬⊃]	$\dfrac{\Gamma \Rightarrow \varphi, \Delta \quad \Gamma \Rightarrow \neg\psi, \Delta}{\Gamma \Rightarrow \neg(\varphi \supset \psi), \Delta}$
[¬⊗⇒]	$\dfrac{\Gamma, \neg\psi, \neg\phi \Rightarrow \Delta}{\Gamma, \neg(\psi \otimes \phi) \Rightarrow \Delta}$	[⇒¬⊗]	$\dfrac{\Gamma \Rightarrow \Delta, \neg\psi \quad \Gamma \Rightarrow \Delta, \neg\phi}{\Gamma \Rightarrow \Delta, \neg(\psi \otimes \phi)}$
[¬⊕⇒]	$\dfrac{\Gamma, \neg\psi \Rightarrow \Delta \quad \Gamma, \neg\phi \Rightarrow \Delta}{\Gamma, \neg(\psi \oplus \phi) \Rightarrow \Delta}$	[⇒¬⊕]	$\dfrac{\Gamma \Rightarrow \Delta, \neg\psi, \neg\phi}{\Gamma \Rightarrow \Delta, \neg(\psi \oplus \phi)}$
[f⇒]	$\Gamma, \mathsf{f} \Rightarrow \Delta$	[⇒¬f]	$\Gamma \Rightarrow \Delta, \neg\mathsf{f}$
[⇒ c]	$\Gamma \Rightarrow \Delta, \mathsf{c}$	[⇒¬c]	$\Gamma \Rightarrow \Delta, \neg\mathsf{c}$
[u⇒]	$\Gamma, \mathsf{u} \Rightarrow \Delta$	[¬u⇒]	$\Gamma, \neg\mathsf{u} \Rightarrow \Delta$

Figure 2 The proof system $G_{\mathbf{4All}}$.

▶ **Corollary 7.3.** *For every* $\mathbf{L} \in \{\mathbf{4All}, \mathbf{4CC}, \mathbf{4Nex}, \mathbf{4Flex}, \mathbf{4CL}\}$*,* $H_{\mathbf{L}}$ *is sound and complete for* $\mathbf{L}$.

▶ Remark. Other proof systems for paradefinite logics have been considered in the literature, and in many cases it is possible to show that they are equivalent to some of the proof systems considered here. For instance, Bou and Rivieccio's Hilbert-style proof system introduced in [12] has 23 rules for the language of $\{\neg, \vee, \wedge, \otimes, \oplus\}$, and no axioms. In [12] it is shown that this system is equivalent to the corresponding fragment of $G_{\mathbf{4All}}$, and it is not difficult to see that it is obtained by a straightforward translation of that system.

Inference Rule: [MP] $\dfrac{\psi \quad \psi \supset \varphi}{\varphi}$

Axioms:

[⇒⊃1]	$\psi \supset (\varphi \supset \psi)$		
[⇒⊃2]	$(\psi \supset (\varphi \supset \tau)) \supset ((\psi \supset \varphi) \supset (\psi \supset \tau))$		
[⇒⊃3]	$((\psi \supset \varphi) \supset \psi) \supset \psi$		
[⇒∧⊃]	$\psi \wedge \varphi \supset \psi, \quad \psi \wedge \varphi \supset \varphi$	[⇒⊃∧]	$\psi \supset (\varphi \supset \psi \wedge \varphi)$
[⇒⊃∨]	$\psi \supset \psi \vee \varphi, \quad \varphi \supset \psi \vee \varphi$	[⇒ ∨⊃]	$(\psi \supset \tau) \supset ((\varphi \supset \tau) \supset (\psi \vee \varphi \supset \tau))$
[¬¬⇒]	$\neg\neg\varphi \supset \varphi$	[⇒¬¬]	$\varphi \supset \neg\neg\varphi$
[¬⊃⇒1]	$\neg(\varphi \supset \psi) \supset \varphi$	[¬⊃⇒2]	$\neg(\varphi \supset \psi) \supset \neg\psi$
[⇒¬⊃]	$(\varphi \wedge \neg\psi) \supset \neg(\varphi \supset \psi)$		
[¬∨⇒1]	$\neg(\varphi \vee \psi) \supset \neg\varphi$	[¬∨⇒2]	$\neg(\varphi \vee \psi) \supset \neg\psi$
[⇒¬∨]	$(\neg\varphi \wedge \neg\psi) \supset \neg(\varphi \vee \psi)$		
[¬∧⇒]	$\neg(\varphi \wedge \psi) \supset (\neg\varphi \vee \neg\psi)$		
[⇒¬∧1]	$\neg\varphi \supset \neg(\varphi \wedge \psi)$	[⇒¬∧2]	$\neg\psi \supset \neg(\varphi \wedge \psi)$
[⇒⊗]	$\psi \supset \varphi \supset \psi \otimes \varphi$	[⊗⇒]	$\psi \otimes \varphi \supset \psi, \quad \psi \otimes \varphi \supset \varphi$
[⇒⊕]	$\psi \supset \psi \oplus \varphi, \quad \varphi \supset \psi \oplus \varphi$	[⊕⇒]	$(\psi \supset \tau) \supset (\varphi \supset \tau) \supset (\psi \oplus \varphi \supset \tau)$
[⇒¬⊕]	$\neg\psi \oplus \neg\varphi \supset \neg(\psi \oplus \varphi)$	[¬⊕⇒]	$\neg(\psi \oplus \varphi) \supset \neg\psi \oplus \neg\varphi$
[⇒¬⊗]	$\neg\psi \otimes \neg\varphi \supset \neg(\psi \otimes \varphi)$	[¬⊗⇒]	$\neg(\psi \otimes \varphi) \supset \neg\psi \otimes \neg\varphi$
[f⇒]	$\mathsf{f} \supset \psi$		
[u⇒]	$\mathsf{u} \supset \psi$	[⇒c]	$\psi \supset \mathsf{c}$
[¬u⇒]	$\neg\mathsf{u} \supset \psi$	[⇒¬c]	$\psi \supset \neg\mathsf{c}$

Figure 3 The proof system $H_{\mathbf{4All}}$.

References

1 O. Arieli. Paraconsistent reasoning and preferential entailments by signed quantified Boolean formulae. *ACM Transactions on Computational Logic*, 8(3), 2007.

2 O. Arieli and A. Avron. The value of the four values. *Artificial Intelligence*, 102(1):97–141, 1998.

3 O. Arieli, A. Avron, and A. Zamansky. Ideal paraconsistent logics. *Studia Logica*, 99(1–3):31–60, 2011.

4 O. Arieli and M. Denecker. Reducing preferential paraconsistent reasoning to classical entailment. *Logic and Computation*, 13(4):557–580, 2003.

5 A. Avron. Simple consequence relations. *Information and Computation*, 92(1):105–140, 1991.

6 A. Avron. On the expressive power of three-valued and four-valued languages. *Journal of Logic and Computation*, 9(6):977–994, 1999.

7 A. Avron, J. Ben-Naim, and B. Konikowska. Processing information from a set of sources. In *Towards Mathematical Philosophy*, Trends in Logic, volume 10, pages 165–186. Springer, 2009.

8 A. Avron and B. Konikowska. Multi-valued calculi for logics based on non-determinism. *Logic Journal of the IGPL*, 13(4):365–387, 2005.

9 N. Belnap. How a computer should think. In G. Ryle, editor, *Contemporary Aspects of Philosophy*, pages 30–56. Oriel Press, 1977.

10 N. Belnap. A useful four-valued logic. In J. M. Dunn and G. Epstein, editors, *Modern Uses of Multiple-Valued Logics*, pages 7–37. Reidel Publishing Company, 1977.

11 J. Y. Béziau. Bivalent semantics for De Morgan logic (The uselessness of four-valuedness). In W. A. Carnielli, M. E. Coniglio, and I. M. L. D'Ottaviano, editors, *The many sides of logic*, pages 391–402. College Publication, 2009.

12 F. Bou and U. Rivieccio. The logic of distributive bilattices. *Logic Journal of the IGPL*, 19(1):183–216, 2010.

13 N. da Costa. On the theory of inconsistent formal systems. *Notre Dame Journal of Formal Logic*, 15:497–510, 1974.

14 J. M. Dunn. *The Algebra of Intensional Logics.* PhD thesis, University of Pittsburgh, Ann Arbor (UMI), 1966.

15 J. M. Dunn. Intuitive semantics for first-degree entailments and coupled trees. *Philosophical Studies*, 29:149–168, 1976.

16 M. Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11(2):91–116, 1991.

17 M. Fitting. The family of stable models. *Journal of Logic Programming*, 17:197–225, 1993.

18 G. Gentzen. Investigations into logical deduction, 1934. In German. An English translation appears in 'The Collected Works of Gerhard Gentzen', edited by M. E. Szabo, North-Holland, 1969.

19 S. Gottwald. A treatise on many-valued logics. In *Studies in Logic and Computation*, volume 9. Research Studies Press, Baldock, 2001.

20 G. Malinowski. *Many-Valued Logics.* Clarendon Press, 1993.

21 D. J. Shoesmith and T. J. Smiley. Deducibility and many-valuedness. *Journal of Symbolic Logic*, 36:610–622, 1971.

22 D. J. Shoesmith and T. J. Smiley. *Multiple Conclusion Logic.* Cambridge University Press, 1978.

23 G. Takeuti. *Proof Theory.* Elsevier, 1975.

24 A.S. Troelstra and H. Schwichtenberg. *Basic Proof Theory.* Cambridge University Press, 2000.

25 A. Urquhart. Many-valued logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II, pages 249–295. Kluwer, 2001. Second edition.

Structural Interactions and Absorption of Structural Rules in BI Sequent Calculus

Ryuta Arisaka

National Institute of Informatics, Tokyo, Japan
ryutaarisaka@gmail.com

Abstract

Development of a contraction-free BI sequent calculus, be the contraction-freeness implicit or explicit, has not been successful in the literature. We address this problem by presenting such a sequent system. Our calculus involves no structural rules. It should be an insight into non-formula contraction absorption in other non-classical logics. Contraction absorption in sequent calculus is associated to simpler cut elimination and to efficient proof searches.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases cut-elimination, contraction-free, sequent calculus, proof theory, BI, logic combination

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.8

1 Introduction

Propositional BI [22] is a combined logic formed from propositional intuitionistic logic IL and propositional multiplicative fragment of intuitionistic linear logic MILL. Recall that IL, and respectively MILL, have the following logical connectives: $\{\top_0, \bot_0, \wedge_2, \vee_2, \supset_2\}$ (Cf. any standard text on the mathematical logic for intuitionistic logic; [16] for instance), and respectively, $\{\mathbf{1}_0, \otimes_2, \multimap_2\}$ (Cf. [11] for linear logic).[1] A rough intuition about BI is that a BI expression is any expression that is constructable from $(\mathcal{P}, \{\top_0, \bot_0, \wedge_2, \vee_2, \supset_2, \mathbf{1}_0, \otimes_2, \multimap_2\})$. $\mathcal{P}$ denotes some set of propositional letters. Following the popular convention in BI, we use the symbol $*$ in place of $\otimes$, and $-\!\!*$ in place of $\multimap$. In place of $\mathbf{1}$, we use $^*\!\top$, emphasising some link of its to $\top$, as to be shortly stated. It holds true that what IL or MILL considers a theorem, BI also does [22]. To this extent BI is a conservative extension of the two propositional logics.

Now, one may contemplate the converse. Is it the case that what BI considers a theorem, IL or MILL also does, i.e. is it the case that every BI formula is reducible either into an IL formula or into a MILL formula? It is stated in [22] that that is not so.

Analysis of the way logics combine is itself an interest. When one combines two logics, it is possible - depending on how the chosen methodology combines the logics - that some logical connective in one of them collapses onto some logical connective in the other. A notable example is the case of classical logic and intuitionistic logic [6, 5]. There, intuitionistic implication can become classical implication. If another approach is chosen, classical implication can also become intuitionistic implication. In order to prevent these from occurring, one must prepare the combined logic domain in such a way that, within the domain, the classical logic domain is sufficiently independent of the intuitionistic logic domain. The reason pertains to the difference in their viewpoint of what an infinity is. Similarly in the combination of IL

[1] The subscripts denote the arity.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 8; pp. 8:1–8:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

with MILL, some sort of the merging of logical connectives could occur. In BI, one that is intentionally avoided is the conflict between the two implications. The following example in the BI proof theory is taken from [22].

$$\frac{\Gamma; F \vdash G}{\Gamma \vdash F \supset G} \supset R \quad \frac{\Gamma, F \vdash G}{\Gamma \vdash F \mathrel{-\!\!*} G} \mathrel{-\!\!*} R$$

F and G are assumed to be some arbitrary BI formula. The semi-colon and the comma are the two structural connectives acting as the structural counterparts of $\wedge$ and respectively $*$, which can nest over one another. Γ denotes some arbitrary BI structure.[2] BI achieves separation of the two implications by the two structural connectives.[3] Here the basic axioms of IL and MILL can be recalled: that if $(F \wedge G) \supset H$ for some formulas F, G and H is a theorem in IL, then so is $F \supset (G \supset H)$ (which structurally translates into $\supset R$ above); and that if $(F * G) \mathrel{-\!\!*} H$ is a theorem in MILL, then so is $F \mathrel{-\!\!*} (G \mathrel{-\!\!*} H)$ (which structurally translates into $\mathrel{-\!\!*} R$ above). On the other hand, there is certain glueing between $\top$ and $\top^*$: in BI, F is a true expression iff $F * \top^*$ is iff $F \wedge \top$ is. This connection is chosen not to be eliminated, although it could be eliminated if one so desires.

Under the particular combination that forms BI, there is no free distribution of ";" over "," or of "," over ";". This implies that a BI structure is, as we just stated, a nesting of structures in the form of $\Gamma_1; \Gamma_2$, called additive structures, and those in the form of Γ_1, Γ_2, called multiplicative structures. There is a proof theoretical asymmetry among them by the availability of structural inference rules. Consider for example the following familiar structural rules (in sequent calculi) that come from IL:

$$\frac{\Gamma(\Gamma_1; \Gamma_1) \vdash F}{\Gamma(\Gamma_1) \vdash F} \text{Contraction} \quad \frac{\Gamma(\Gamma_1) \vdash F}{\Gamma(\Gamma_1; \Gamma_2) \vdash F} \text{Weakening}$$

Here $\Gamma(\cdots)$ abstracts any other structures surrounding the focused ones in the sequents. These are available in BI sequent calculus LBI [24]. On the other hand, neither of the inferences below is - as a rule - permitted.

$$\frac{\Gamma(\Gamma_1, \Gamma_1) \vdash F}{\Gamma(\Gamma_1) \vdash F} \qquad \frac{\Gamma(\Gamma_1) \vdash F}{\Gamma(\Gamma_1, \Gamma_2) \vdash F}$$

'As a rule' because there are some exceptions to the guideline.

$$\frac{\Gamma(\Gamma_1, \top^*) \vdash F}{\Gamma(\Gamma_1) \vdash F} \qquad \frac{\Gamma(\Gamma_1) \vdash F}{\Gamma(\Gamma_1, \top^*) \vdash F}$$

1.1 Research problems and contributions

In $\Gamma_1; \Gamma_1$ on the premise of Contraction, or in $\Gamma_1; \Gamma_2$ on the conclusion of Weakening, neither Γ_1 nor Γ_2 must be additive. Consider then the following inferences, each of which is an instance of Contraction:

$$\frac{\Gamma((F; F), G) \vdash H}{\Gamma(F, G) \vdash H} Ctr_1 \quad \frac{\Gamma(F, (G; G)) \vdash H}{\Gamma(F, G) \vdash H} Ctr_2 \quad \frac{\Gamma((\Gamma_a, \Gamma_b); (\Gamma_a, \Gamma_b)) \vdash H}{\Gamma(\Gamma_a, \Gamma_b) \vdash H} Ctr_3$$

[2] These and other orthodox proof-theoretical terms are assumed to be familiar to the readers. They are found for example in [25, 16]. The formal definitions that we will need for our technical discussions will be found in the next section.

[3] The need for more than one structural connectives in proof systems was recognised in display calculus [3] as well as in other studies, e.g. in the multi-modal categorial type logics [20] and in relevant logics [18, 19], which were developed prior to the appearance of BI.

Observe it is a formula that duplicates upwards in the first two inferences. These are simply adaptations of the usual structural contraction available in G1i [25], the standard IL sequent calculus. It is a well-known fact that, as far as G1i is concerned, elimination of the structural weakening requires hardly any effort, and that the structural contraction goes admissible once the left implication rule is modified in the weakening-free IL sequent calculus; Cf. [25, 16] for the results but also [13] for the idea of eliminating the structural contraction rule. Given that the same elimination technique has been shown to be applicable to many other extensions of IL, it is expected on a reasonable ground that handling these formula contractions (and weakenings) is straightforward also in LBI. As can be seen in Ctr_3, however, the scope of Contraction is not restricted to the formula contractions. The degree of the nesting of additive/multiplicative structures in Γ_1 in Contraction can be arbitrarily large.

One pertinent question to ask is if it is possible at all to eliminate the non-formula contractions from LBI, eliminating Contraction as the result. Actually, it is not very difficult to postpone answering this question, if replacement of Contraction with a set of alternative new structural rules is permitted. The Contraction can be then emulated in the new structural rules. Such replacement strategies work particularly well if one retains the cut rule in the sequent system. Knowing, however, that they rather relocate the issue that was expressed in the original question into the new structural rules, we may just as well strengthen the question and ask, instead, if a BI sequent calculus without structural rules is derivable at all, this way precluding any miscommunication.

In setting for the investigation, it seems there are two major sources of difficulty one must face. The first difficulty comes from the equivalences $\Gamma, ^*\top = \Gamma = \Gamma; \top$, structural counterparts of the above-mentioned equivalences, which imply bidirectional inference rules.

$$\frac{\Gamma(\Gamma_1) \vdash F}{\Gamma(\Gamma_1; \top) \vdash F} \qquad \frac{\Gamma(\Gamma_1; \top) \vdash F}{\Gamma(\Gamma_1) \vdash F} \qquad \frac{\Gamma(\Gamma_1) \vdash F}{\Gamma(\Gamma_1, ^*\top) \vdash F} \qquad \frac{\Gamma(\Gamma_1, ^*\top) \vdash F}{\Gamma(\Gamma_1) \vdash F}$$

As well as being obvious sources of non-termination, they obscure the core mechanism of the interactions between additive and multiplicative structures, since they imply a free transformation of an additive structure into a multiplicative one and vice versa. The second difficulty is the difficulty of isolating the effect of the structural contraction from that of the structural weakening. Donnelly *et al* [7] succeeded in eliminating structural weakening; however, they had to absorb contraction into the structural weakening as well as into logical rules. Absorption of one structural rule into another structural rule is a little problematic, since - as we have already mentioned - the former still occurs indirectly through the latter which is a structural rule. It is also not so straightforward to know whether either weakening or contraction is immune to the effect of the structural equivalences.

Despite the technical obstacles, we show the answer to the above-posed question to be in the affirmative by presenting a structural-rule-free BI sequent calculus. What it is to LBI is what G3i is to G1i. As far as can be gathered from the literature, the elimination of contraction from BI sequent calculus has not been previously successful, be the sense of contraction-freeness according to the sense in G3i or the sense in G4i [8]. The following are some motivations for presenting such a sequent calculus.

1. The current status of the knowledge of structural interactions within BI proof systems is not very satisfactory. From the perspective of theorem proving for example, the presence of the bidirectional rules and contraction as explicit structural rules in LBI means that it is difficult to actually prove that an invalid BI formula is underivable within the calculus. This is because LBI by itself does not provide termination conditions save when a (backward) derivation actually terminates: the only case in which no more backward derivation on a LBI sequent is possible is when the sequent is empty; the only case in

which it is empty is when it is the premise of an axiom. The contraction-free BI sequent calculus is a step forward in this respect.

2. There are other sequent calculi that necessarily require a non-formula structural contraction rule (or else alternative structural rules that emulate the effect). Sequent systems of the relevant logics closely related to BI [10] are good examples. Sequent systems of some constructive modal logics [23] also require non-formula contractions; Cf. [1]. It tends to be almost always the case that the presence of a structural contraction rule increases the technical complexity of a cut elimination proof (see the induction measure in [1]). The techniques to eliminate non-formula structural contraction rules are useful for simplifying the proof of cut admissibility in the sequent calculi of the existing or emerging non-classical logics.

This work has only a marginal technical dependency on earlier works: it suffices to have the knowledge of LBI [24]; and to understand [24], it suffices to have the basic knowledge of the structural proof theory [16, 25].

1.2 Structure of the remaining sections

In Section 2 we present technical preliminaries of BI proof theory. In Section 3 we introduce our BI calculus LBIZ with no structural rules. In Section 4 we show its main properties including admissibility of structural rules and its equivalence to LBI. We also show Cut admissibility in LBIZ. Section 5 concludes.

2 BI Proof Theory - Preliminaries

We assume availability of the following meta-logical notations. “If and only if” is abbreviated by “iff”.

▶ **Definition 1** (Meta-connectives). We denote logical conjunction (“and”) by $\wedge^\dagger$, logical disjunction (“or”) by $\vee^\dagger$, material implication (“implies”) by $\rightarrow^\dagger$, and equivalence by $\leftrightarrow^\dagger$. These follow the semantics of standard classical logic’s.

We denote the set of propositional variables by $\mathcal{P}$ and refer to an element of $\mathcal{P}$ by p or q with or without a subscript.

A BI formula $F(, G, H)$ with or without a subscript is constructed from the following grammar: $F := p \mid \top \mid \bot \mid {}^{*}\top \mid F \wedge F \mid F \vee F \mid F \supset F \mid F * F \mid F \mathrel{-\!\!*} F$. The set of BI formulas is denoted by $\mathfrak{F}$.

▶ **Definition 2** (BI structures). BI structure $\Gamma(, Re)$ with or without a subscript/superscript, commonly referred to as a bunch [22], is defined by: $\Gamma := F \mid \Gamma; \Gamma \mid \Gamma, \Gamma$. We denote by $\mathfrak{S}$ the set of BI structures.

We define the binding order to be $[\wedge, \vee, *] \gg [\supset, \mathrel{-\!\!*}] \gg [;\ ,] \gg [\wedge^\dagger, \vee^\dagger] \gg [\rightarrow^\dagger, \leftrightarrow^\dagger]$ in a strictly decreasing precedence. Connectives in the same group have the same binding precedence.

Both of the structural connectives “;” and “,” are defined to be associative and commutative. On the other hand, we do not assume distributivity of “;” over ‘,’ or vice versa. A context “ $\Gamma(-)$” (with a hole “ $-$") takes the form of a tree because of the nesting of additive/multiplicative structures.

▶ **Definition 3** (Context). A context $\Gamma(-)$ is finitely constructed from the following grammar:
$\Gamma(-) := - \mid \Gamma(-); \Gamma \mid \Gamma; \Gamma(-) \mid \Gamma(-), \Gamma \mid \Gamma, \Gamma(-)$.

$$\frac{}{F \vdash F}\ \text{id} \qquad \frac{\Gamma_1 \vdash G \qquad \Gamma(G) \vdash H}{\Gamma(\Gamma_1) \vdash H}\ \text{Cut} \qquad \frac{}{\Gamma(\bot) \vdash H}\ \bot L$$

$$\frac{}{\Gamma \vdash \top}\ \top R \qquad \frac{}{{}^{*}\top \vdash {}^{*}\top}\ {}^{*}\top R \qquad \frac{\Gamma(F;G) \vdash H}{\Gamma(F \wedge G) \vdash H}\ \wedge L$$

$$\frac{\Gamma(F) \vdash H \qquad \Gamma(G) \vdash H}{\Gamma(F \vee G) \vdash H}\ \vee L \qquad \frac{\Gamma_1 \vdash F \qquad \Gamma(\Gamma_1;G) \vdash H}{\Gamma(\Gamma_1;F \supset G) \vdash H}\ \supset L$$

$$\frac{\Gamma(F,G) \vdash H}{\Gamma(F * G) \vdash H}\ *L \qquad \frac{\Gamma_1 \vdash F \qquad \Gamma(G) \vdash H}{\Gamma(\Gamma_1, F -\!\!* G) \vdash H}\ -\!\!*L \qquad \frac{\Gamma \vdash F \qquad \Gamma \vdash G}{\Gamma \vdash F \wedge G}\ \wedge R$$

$$\frac{\Gamma \vdash F_i}{\Gamma \vdash F_1 \vee F_2}\ \vee R \qquad \frac{\Gamma;F \vdash G}{\Gamma \vdash F \supset G}\ \supset R \qquad \frac{\Gamma_1 \vdash F \qquad \Gamma_2 \vdash G}{\Gamma_1, \Gamma_2 \vdash F * G}\ *R$$

$$\frac{\Gamma, F \vdash G}{\Gamma \vdash F -\!\!* G}\ -\!\!*R \qquad \frac{\Gamma(\Gamma_1) \vdash H}{\Gamma(\Gamma_1;\Gamma_2) \vdash H}\ \text{Wk L} \qquad \frac{\Gamma(\Gamma_1;\Gamma_1) \vdash H}{\Gamma(\Gamma_1) \vdash H}\ \text{Ctr L}$$

$$\begin{array}{c} \Gamma(\Gamma_1;\top) \vdash H \\ ::::::::::::::: \\ \Gamma(\Gamma_1) \vdash H \end{array}\ EqAnt_1 \qquad \begin{array}{c} \Gamma(\Gamma_1, {}^{*}\top) \vdash H \\ ::::::::::::::: \\ \Gamma(\Gamma_1) \vdash H \end{array}\ EqAnt_2$$

Figure 1 LBI: a BI sequent calculus. Inference rules with a double-dotted line are bidirectional. $i \in \{1, 2\}$. Structural connectives are fully associative and commutative.

We assume that a BI structure Γ_2 replaces $-$ in a context $\Gamma_1(-)$ as $\Gamma_1(\Gamma_2)$ which, we again assume, is a BI structure.

▶ **Definition 4** (Sequents)**.** The set of BI sequents $\mathfrak{D}$ is defined by:
$\mathfrak{D} := \{\Gamma \vdash F \mid \Gamma \in \mathfrak{S} \wedge^{\dagger} F \in \mathfrak{F}\}$.
We call the left hand side of $\vdash$ antecedent, and the right hand side of $\vdash$ consequent.

A variant of the first BI sequent calculus LBI [24] is found in Figure 1. Notice that we do not use the nullary structural connectives used in the reference. All the additive inference rules, by which we mean all the inference rules that originate in IL, share contexts. Consider $\vee L$ for example. In the inference rule the same context in the conclusion propagates onto both premises. Multiplicative inference rules, by which we mean the inference rules that originate in MILL, are context-free [25] or resource sensitive. A good example to illustrate this is $*R$: both Γ_1 and Γ_2 in the conclusion sequent are viewed as resources for the inference rule, and are split into the premises of the rule. Note again our assumption of commutativity of "," here. Cut is admissible in [LBI- Cut].

▶ **Lemma 5** (Cut admissibility in LBI - Cut)**.** *There is a direct cut elimination procedure which proves admissibility of* Cut *in [*LBI- Cut*] (sketched in [24]; corrected in [2]).*

The following derivation highlights a simple additive/multiplicative interaction in BI.

$$\dfrac{\dfrac{\dfrac{}{F \vdash F}\ \text{id} \qquad \dfrac{}{F \vdash F}\ \text{id}}{F \vdash F \wedge F}\ \wedge R}{\vdash F -\!\!* F \wedge F}\ -\!\!*L$$

This shows that $F -\!\!* F \wedge F$ is provable in LBI. Further, given that semantics is given to LBI [24], it is a valid BI formula. Any others that are provable in LBI are valid. We assume that readers are familiar with provability or derivability (found in standard proof theory texts), and with validity or satisfiability (found in Wikipedia).

3 LBIZ: A Structural-Rule-Free BI Sequent Calculus

In this section we present a new BI sequent calculus LBIZ (Figure 2) in which no structural rules appear. We first introduce notations necessary for reading inference rules in the calculus. From this point on, whenever we write $\widetilde{\Gamma}$ for any BI structure, it shall be agreed that it may be empty. The emptiness is in the following sense: $\widetilde{\Gamma_1}; \Gamma_2 = \Gamma_2$ if Γ_1 is empty; and $\widetilde{\Gamma_1}, \Gamma_2 = \Gamma_2$ if Γ_1 is empty. Apart from this, we use two other notations.

3.1 Essence of antecedent structures

Co-existence of IL and MILL in BI calls for new contraction-absorption techniques. We need to consider possible interferences to one structural rule from the others. To illustrate the technical difficulty, $EqAnt2_{\texttt{LBI}}$ for instance interacts directly with $WkL_{\texttt{LBI}}$. When $WkL_{\texttt{LBI}}$ is absorbed into the rest, the effect propagates to one direction of $EqAnt2_{\texttt{LBI}}$, resulting in;

$$\frac{\Gamma(\Gamma_1) \vdash H}{\Gamma(\Gamma_1, ({}^{*}\top; \widetilde{\Gamma_2})) \vdash H}\ EA_2$$

Hence absorption of $WkL_{\texttt{LBI}}$ must involve analysis of $EqAnt2_{\texttt{LBI}}$ as well. To solve this particular problem we define a new notation: 'essence' of BI structures.

▶ **Definition 6** (Essence of BI structures). Let Γ_1 be a BI structure. Then we have a set of its essences as defined in the following inductive rules.

- Γ_2 is an essence of Γ_1 if $\Gamma_1 = \Gamma_2$.[4]
- $\Gamma(\Gamma', ({}^{*}\top; \widetilde{\Gamma_2}))$[5] is an essence of Γ_1 if $\Gamma(\Gamma')$ is an essence of Γ_1.

By $\mathbb{E}(\Gamma_1)$ we denote an essence of Γ_1.

The essence takes care of an arbitrary number of EA_2 applications, while nicely retaining a compact representation of a sequent (see the calculus). In each of $\supset L$ and $-\!\!*L$, the essence in the premise(s) and that in the conclusion are the same and identical BI structure. Specifically, the use of $\mathbb{E}(\Gamma)$ in multiple sequents in a derivation tree signifies the same BI structure.

▶ **Example 7.** A LBIZ-derivation:

$$\dfrac{\dfrac{}{F_1; (({}^{*}\top; \Gamma_1), F_1 \supset F_2) \vdash F_1}\ id \qquad \dfrac{}{F_2; F_1; (({}^{*}\top; \Gamma_1), F_1 \supset F_2) \vdash F_2}\ id}{F_1; (({}^{*}\top; \Gamma_1), F_1 \supset F_2) \vdash F_2}\ \supset L$$

can be alternatively written down as:

$$\dfrac{\dfrac{}{\mathbb{E}(F_1; F_1 \supset F_2) \vdash F_1}\ id \qquad \dfrac{}{F_2; \mathbb{E}(F_1; F_1 \supset F_2) \vdash F_2}\ id}{\mathbb{E}(F_1; F_1 \supset F_2) \vdash F_2}\ \supset L$$

if $\mathbb{E}(F_1; F_1 \supset F_2) = F_1; (({}^{*}\top; \Gamma_1), F_1 \supset F_2)$.

$\mathbb{E}'(\Gamma)$ (or $\mathbb{E}_1(\Gamma)$ or any essence that differs from $\mathbb{E}$ by the presence of a subscript, a superscript or both) in the same derivation tree does not have to be coincident with the BI structure that the $\mathbb{E}(\Gamma)$ denotes. However, we do - for prevention of inundation of many superscripts and subscripts - make an exception. In the cases where no ambiguity is likely to arise such as in the following:

[4] For some Γ_2. The equality is of course up to associativity and commutativity.

[5] For some $\widetilde{\Gamma_2}$; similarly in the rest.

$$\frac{}{\mathbb{E}(\widetilde{\Gamma};p)\vdash p}\,id \qquad \frac{}{\Gamma(\bot)\vdash F}\,\bot L \qquad \frac{}{\Gamma\vdash\top}\,\top R \qquad \frac{}{\mathbb{E}(\widetilde{\Gamma};{}^{*}\top)\vdash {}^{*}\top}\,{}^{*}\top R$$

$$\frac{\Gamma(F;G)\vdash H}{\Gamma(F\wedge G)\vdash H}\,\wedge L \qquad \frac{\Gamma\vdash F \qquad \Gamma\vdash G}{\Gamma\vdash F\wedge G}\,\wedge R$$

$$\frac{\Gamma(F)\vdash H \qquad \Gamma(G)\vdash H}{\Gamma(F\vee G)\vdash H}\,\vee L \qquad \frac{\Gamma\vdash F_i}{\Gamma\vdash F_1\vee F_2}\,\vee R$$

$$\frac{\mathbb{E}(\widetilde{\Gamma_1};F\supset G)\vdash F \qquad \Gamma(G;\mathbb{E}(\widetilde{\Gamma_1};F\supset G))\vdash H}{\Gamma(\mathbb{E}(\widetilde{\Gamma_1};F\supset G))\vdash H}\,\supset L \qquad \frac{\Gamma;F\vdash G}{\Gamma\vdash F\supset G}\,\supset R$$

$$\frac{\Gamma(F,G)\vdash H}{\Gamma(F*G)\vdash H}\,*L \qquad \frac{Re_i\vdash F_1 \qquad Re_j\vdash F_2}{\Gamma'\vdash F_1*F_2}\,*R$$

$$\frac{Re_i\vdash F \qquad \Gamma((\widetilde{Re_j},G);(\widetilde{\Gamma'},\mathbb{E}(\widetilde{\Gamma_1};F\mathrel{-\!\!*}G)))\vdash H}{\Gamma(\widetilde{\Gamma'},\mathbb{E}(\widetilde{\Gamma_1};F\mathrel{-\!\!*}G))\vdash H}\,\mathrel{-\!\!*}L \qquad \frac{\Gamma,F\vdash G}{\Gamma\vdash F\mathrel{-\!\!*}G}\,\mathrel{-\!\!*}R$$

Figure 2 LBIZ: a BI sequent calculus with zero occurrence of explicit structural rules. $i,j\in\{1,2\}$. $i\neq j$. Structural connectives are fully associative and commutative. In $*R$ and $\mathrel{-\!\!*}L$, if Γ' is not empty, $(Re_1,Re_2)\in$ Candidate(Γ'); otherwise, $Re_i={}^{*}\top$ and Re_j is empty. Both $\mathbb{E}$ and Candidate are as defined in the main text.

$$\frac{\Gamma(\mathbb{E}(\Gamma_1;F;G))\vdash H}{\Gamma(\mathbb{E}(\Gamma_1;F\wedge G))\vdash H}\,\wedge L$$

we assume that the essence in the conclusion is the same antecedent structure as the essence in the premise(s) except what the inference rule modifies.

3.2 Correspondence between Re_i/Re_j and Γ'

▶ **Definition 8** (Relation $\preceq$)**.** We define a binary relation $\preceq:\mathfrak{S}\times\mathfrak{S}$ as follows.

- $\Gamma_1\preceq\Gamma_2$ if $\Gamma_1=\Gamma_2$.
- $\Gamma(\Gamma_1)\preceq\Gamma(\Gamma_1;\Gamma')$.
- $[\Gamma_1\preceq\Gamma_2]\wedge^{\dagger}[\Gamma_2\preceq\Gamma_3]\rightarrow^{\dagger}[\Gamma_1\preceq\Gamma_3]$.

Intuitively if $\Gamma_1\preceq\Gamma_2$, then there exists a LBI-derivation:

$$\begin{array}{c}\Gamma(\Gamma_1)\vdash H\\ \hline\hline \Gamma(\Gamma_2)\vdash H\end{array}\,WkL$$

for any $\Gamma(\Gamma_1)$ and any H. Here and elsewhere a double line indicates zero or more derivation steps.

▶ **Definition 9** (Candidates)**.** Let Γ be a BI structure, then any of the following pairs is a candidate of Γ.

- $(\Gamma_x,{}^{*}\top)$ if $\Gamma_x\preceq\Gamma$.
- (Γ_x,Γ_y) if $\Gamma_x,\Gamma_y\preceq\Gamma$.

We denote the set of candidates of Γ by Candidate(Γ).

Now we see the connection between Re_i/Re_j and Γ' in the two rules $*R/\mathrel{-\!\!*}L$.

▶ **Definition 10** (Re_i/Re_j in $*R/\mathrel{-\!\!*}L$)**.** In $*R$ and $\mathrel{-\!\!*}L$, if Γ' is empty (this case applies to $\mathrel{-\!\!*}L$ only), $Re_i={}^{*}\top$ and Re_j is empty. If it is not empty, then $(Re_1,Re_2)\in$ Candidate(Γ').

`Candidate` allows for absorption of an arbitrary number of Wk L applications in the two inference rules. The sequent: $D : p_1;((p_2;p_3),(p_4;p_5)) \vdash p_2 * p_5$, illustrates why it is used. It is clearly `LBI`-derivable:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p_2 \vdash p_2}\,\text{id} \qquad \dfrac{}{p_5 \vdash p_5}\,\text{id}}{p_2, p_5 \vdash p_2 * p_5}\,{*R}}{p_2,(p_4;p_5) \vdash p_2 * p_5}\,\text{Wk L}}{(p_2;p_3),(p_4;p_5) \vdash p_2 * p_5}\,\text{Wk L}}{D : p_1;((p_2;p_3),(p_4;p_5)) \vdash p_2 * p_5}\,\text{Wk L}$$

However, $*R$ in `LBI` does not apply immediately to D. Hence $*R$ in `LBIZ` must absorb Wk L.

With the two notations we have introduced, what the inference rules in `LBIZ` are doing should be clear. There are no structural rules. Implicit contraction occurs only in $\supset L$ and $-\!*L$.[6] In both of the inference rules, a structure rather than a formula duplicates upwards. This is necessary, for we have the following observation.

▶ **Observation 11** (Non-formula contractions are not admissible). *There exist sequents* $\Gamma \vdash F$ *which are derivable in* `LBI - Cut` *but not derivable in* `LBI - Cut` *without structural contraction.*

Proof. For $-\!*L$ use a sequent $\top -\!* p_1, \top -\!*(p_1 \supset p_2) \vdash p_2$ and assume that every propositional variable is distinct. Then without contraction, there are several derivations. Two sensible ones are shown below (the rest similar). Here and elsewhere we may label a sequent by D with or without a subscript/superscript just so that we may refer to it by the name.

1. $$\dfrac{\dfrac{}{\top -\!*(p_1 \supset p_2) \vdash \top}\,\top R \qquad p_1 \vdash p_2}{D : \top -\!* p_1, \top -\!*(p_1 \supset p_2) \vdash p_2}\,-\!*L$$

2. $$\dfrac{\dfrac{}{\top -\!* p_1 \vdash \top}\,\top R \qquad \dfrac{\dfrac{\top \vdash p_1 \qquad \dfrac{}{p_2 \vdash p_2}\,id}{\top ; p_1 \supset p_2 \vdash p_2}\,\supset L}{p_1 \supset p_2 \vdash p_2}\,EqAnt_1 L}{D : \top -\!* p_1, \top -\!*(p_1 \supset p_2) \vdash p_2}\,-\!*L$$

In both of the derivation trees above, one branch is open. Moreover, such holds true when only formula-level contraction is permitted in `LBI`. The sequent D cannot be derived under the given restriction. If non-formula contractions are available, there is another construction leading to a closed derivation tree:

$$\dfrac{\dfrac{\Pi(D_1) \qquad \Pi(D_2)}{(\top -\!* p_1, \top -\!*(p_1 \supset p_2)); (\top -\!* p_1, \top -\!*(p_1 \supset p_2)) \vdash p_2}\,-\!*L}{D : \top -\!* p_1, \top -\!*(p_1 \supset p_2) \vdash p_2}\,CtrL$$

where $\Pi(D_1)$ and $\Pi(D_2)$ are:

$\boldsymbol{\Pi(D_1)}$:

$$\dfrac{}{\top -\!*(p_1 \supset p_2) \vdash \top}\,\top R$$

$\boldsymbol{\Pi(D_2)}$:

[6] Implicit weakening and others occur also in other inference rules; but they are not very relevant in backward theorem proving.

$$\dfrac{\dfrac{}{\top\mathrel{-\!\!*}p_1\vdash\top}\,\top R \qquad \dfrac{\dfrac{}{p_1\vdash p_1}\,id \qquad \dfrac{\dfrac{}{p_2\vdash p_2}\,id}{p_1;p_2\vdash p_2}\,WkL}{p_1;p_1\supset p_2\vdash p_2}\,\supset L}{p_1;(\top\mathrel{-\!\!*}p_1,\top\mathrel{-\!\!*}(p_1\supset p_2))\vdash p_2}\,\mathrel{-\!\!*}L$$

All the derivation tree branches are closed.

For $\supset L$, use $({}^{*}\!\top;p_1),({}^{*}\!\top;p_1{\supset}p_2)\vdash p_2$. Without non-formula contractions we have (only two sensible ones are shown; the rest similar):

1.

$$\dfrac{{}^{*}\!\top\vdash p_1 \qquad \dfrac{\dfrac{\dfrac{}{p_2\vdash p_2}\,id}{{}^{*}\!\top;p_2\vdash p_2}\,WkL}{({}^{*}\!\top;p_1),({}^{*}\!\top;p_2)\vdash p_2}\,EA_2}{D:({}^{*}\!\top;p_1),({}^{*}\!\top;p_1{\supset}p_2)\vdash p_2}\,\supset L$$

2.

$$\dfrac{\dfrac{p_1\vdash p_2}{{}^{*}\!\top;p_1\vdash p_2}\,WkL}{D:({}^{*}\!\top;p_1),({}^{*}\!\top;p_1{\supset}p_2)\vdash p_2}\,EA_2$$

In the presence of structural contraction, there is a closed derivation.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{p_1\vdash p_1}\,id}{{}^{*}\!\top;p_1;{}^{*}\!\top\vdash p_1}\,WkL \qquad \dfrac{\dfrac{}{p_2\vdash p_2}\,id}{{}^{*}\!\top;p_1;{}^{*}\!\top;p_2\vdash p_2}\,WkL}{{}^{*}\!\top;p_1;{}^{*}\!\top;p_1{\supset}p_2\vdash p_2}\,\supset L}{(({}^{*}\!\top;p_1),({}^{*}\!\top;p_1{\supset}p_2));(({}^{*}\!\top;p_1),({}^{*}\!\top;p_1{\supset}p_2))\vdash p_2}\,EA_2}{D:({}^{*}\!\top;p_1),({}^{*}\!\top;p_1{\supset}p_2)\vdash p_2}\,CtrL$$

◀

We list LBIZ derivations of the two examples in the observation for easy comparisons. We assume that $\Gamma=(\top\mathrel{-\!\!*}p_1,\top\mathrel{-\!\!*}(p_1\supset p_2))$. Also, by $\Pi(D)$ we denote a derivation tree of a sequent D. We assume that $\Pi(D)$ is always closed: every derivation branch of the tree has an empty sequent as the leaf node (the premise of an axiom).

$$\dfrac{\dfrac{}{\top\mathrel{-\!\!*}p_1\vdash\top}\,\top R \qquad \dfrac{\dfrac{}{\top\mathrel{-\!\!*}(p_1\supset p_2)\vdash\top}\,\top R \qquad \Pi(({}^{*}\!\top,p_1);({}^{*}\!\top,p_1\supset p_2);\Gamma\vdash p_2)}{({}^{*}\!\top,p_1);\Gamma\vdash p_2}\,\mathrel{-\!\!*}L}{\Gamma\vdash p_2}\,\mathrel{-\!\!*}L$$

$\Pi(({}^{*}\!\top,p_1);({}^{*}\!\top,p_1\supset p_2);\Gamma\vdash p_2)$ is as follows.

$$\dfrac{\dfrac{}{({}^{*}\!\top,p_1);({}^{*}\!\top,p_1\supset p_2);\Gamma\vdash p_1}\,\mathrm{id} \qquad \dfrac{}{p_2;({}^{*}\!\top,p_1);({}^{*}\!\top,p_1\supset p_2);\Gamma\vdash p_2}\,\mathrm{id}}{({}^{*}\!\top,p_1);({}^{*}\!\top,p_1\supset p_2);\Gamma\vdash p_2}\,\supset L$$

For the other sequent, we have:

$$\dfrac{\dfrac{}{({}^{*}\!\top;p_1),({}^{*}\!\top;p_1\supset p_2)\vdash p_1}\,\mathrm{id} \qquad \dfrac{}{p_2;({}^{*}\!\top;p_1),({}^{*}\!\top;p_1\supset p_2)\vdash p_2}\,\mathrm{id}}{({}^{*}\!\top;p_1),({}^{*}\!\top;p_1\supset p_2)\vdash p_2}\,\supset L$$

4 Main Properties of LBIZ

In this section we show the main properties of LBIZ such as admissibility of weakening, that of EA_2, that of both $EqAnt_{1\,\mathtt{LBI}}$ and $EqAnt_{2\,\mathtt{LBI}}$, that of contraction, and its equivalence to LBI. Cut is also admissible. We will refer to the notion of derivation depth very often.

▶ **Definition 12** (Derivation depth). Let $\Pi(D)$ be a derivation tree. Then the derivation depth of D', a node in $\Pi(D)$, is:

- 1 if D' is the conclusion node of an axiom inference rule.
- 1 + (derivation depth of D_1) if $\Pi(D')$ looks like:
$$\frac{\Pi(D_1)}{D'}$$
- 1 + (the larger of the derivation depths of D_1 and D_2) if $\Pi(D')$ looks like:
$$\frac{\Pi(D_1) \qquad \Pi(D_2)}{D'}$$

4.1 Admissibility of weakening and EA_2

Admissibilities of both weakening and EA_2 are proved depth-preserving. This means in case of weakening that if a sequent $\Gamma(\Gamma_1) \vdash H$ is derivable with derivation depth of k, then $\Gamma(\Gamma_1; \Gamma_2) \vdash H$ is derivable with derivation depth of l such that $l \leq k$.

▶ **Proposition 13** (LBIZ weakening admissibility)**.** *If a sequent $D : \Gamma(\Gamma_1) \vdash F$ is* LBIZ*-derivable, then so is $D' : \Gamma(\Gamma_1; \Gamma_2) \vdash F$ depth-preserving.*

Proof. By induction on derivation depth of D. ◀

▶ **Proposition 14** (Admissibility of EA_2)**.** *If a sequent $D : \Gamma(\Gamma_1) \vdash F$ is* LBIZ*-derivable, then so is $D' : \Gamma(\mathbb{E}(\Gamma_1)) \vdash F$ depth-preserving.*

Proof. By induction on derivation depth of D. ◀

4.2 Inversion lemma

The inversion lemma below is important in simplification of the subsequent discussion.

▶ **Lemma 15** (Inversion lemma for LBIZ)**.** *For the following sequent pairs, if the sequent on the left is* LBIZ*-derivable at most with the derivation depth of k, then so is (are) the sequent(s) on the right.*

$$\begin{array}{rl}
\Gamma(F \wedge G) \vdash H, & \Gamma(F; G) \vdash H \\
\Gamma(F_1 \vee F_2) \vdash H, & \textit{both } \Gamma(F_1) \vdash H \textit{ and } \Gamma(F_2) \vdash H \\
\Gamma(F * G) \vdash H, & \Gamma(F, G) \vdash H \\
\Gamma(\Gamma_1; \top) \vdash H, & \Gamma(\Gamma_1) \vdash H \\
\Gamma(\Gamma_1, {}^{*}\!\top) \vdash H, & \Gamma(\Gamma_1) \vdash H \\
\Gamma \vdash F \wedge G, & \textit{both } \Gamma \vdash F \textit{ and } \Gamma \vdash G \\
\Gamma \vdash F \supset G, & \Gamma; F \vdash G \\
\Gamma \vdash F \mathrel{-\!\!*} G, & \Gamma, F \vdash G
\end{array}$$

Proof. By induction on derivation depth. ◀

4.3 Admissibility of $EqAnt_{1,2}$

▶ **Proposition 16** (Admissibility of $EqAnt_{1,2}$)**.** *$EqAnt_1$* $_{\texttt{LBI}}$ *and $EqAnt_2$* $_{\texttt{LBI}}$ *are depth-preserving admissible in* LBIZ*.*

Proof. Follows from inversion lemma,[7] Proposition 13 and Proposition 14. ◀

[7] Inversion lemma proves one direction.

4.4 Preparation for contraction admissibility in $*R/-\!\!*L$ cases

We dedicate one subsection here to prepare for the main proof of contraction admissibility. Based on Proposition 13, we make an observation about the set of candidates. The discovery, which is to be stated in Proposition 18, led to the solution to the problem of the elimination of `LBI` structural contraction.

▶ **Definition 17** (Representing candidates). Let $\hat{\preceq} : \mathfrak{S} \times \mathfrak{S}$ be a binary relation satisfying:

- $\Gamma_1 \hat{\preceq} \Gamma_2$ if $\Gamma_1 = \Gamma_2$.
- $\Gamma_1 \hat{\preceq} \Gamma_1; \Gamma_3$.
- $[\Gamma_1 \hat{\preceq} \Gamma_2] \wedge^{\dagger} [\Gamma_2 \hat{\preceq} \Gamma_3] \rightarrow^{\dagger} [\Gamma_1 \hat{\preceq} \Gamma_3]$.
- $\Gamma_1, \Gamma_2 \hat{\preceq} \Gamma_1, (\Gamma_2; \Gamma_3)$.

Now let Γ be a `BI` structure. Then any of the following pairs is a representing candidate of Γ.

- $(\Gamma_x, {}^{*}\top)$ if $\Gamma_x \hat{\preceq} \Gamma$.
- (Γ_x, Γ_y) if $\Gamma_x, \Gamma_y \hat{\preceq} \Gamma$.

We denote the set of representing candidates of Γ by `RepCandidate`(Γ).

We trivially have that `RepCandidate`$(\Gamma) \subseteq$ `Candidate`(Γ) for any Γ. More can be said.

▶ **Proposition 18** (Sufficiency of `RepCandidate`). *`LBIZ` with `RepCandidate` instead of `Candidate` for (Re_1, Re_2) is as expressive as `LBIZ` (with `Candidate`).*

Proof. The only inference rules in `LBIZ` that use `Candidate` are $*R$ and $-\!\!*L$. So it suffices to consider only those.

For $*R$, suppose by way of showing contradiction that `LBIZ` with `RepCandidate` is not as expressive as `LBIZ`, then there exists some `LBIZ` derivation tree $\Pi(D)$:

$$\dfrac{\begin{matrix}\vdots \\ D_1 : Re_i \vdash F_1\end{matrix} \qquad \begin{matrix}\vdots \\ D_2 : Re_j \vdash F_2\end{matrix}}{D : \Gamma' \vdash F_1 * F_2}\ *R$$

such that (Re_1, Re_2) must be in `Candidate`$(\Gamma')\backslash$`RepCandidate`(Γ'). Now, without loss of generality assume $(i, j) = (1, 2)$. Then $D_1' : Re_i' \vdash F_1$ and $D_2' : Re_j' \vdash F_2$ for $(Re_i', Re_j') \in$ `RepCandidate`(Γ') are also `LBIZ` derivable (by Proposition 13). But this means that we can choose the (Re_i', Re_j') for (Re_1, Re_2), a direct contradiction to the supposition. Similarly for $-\!\!*L$. ◀

▶ **Theorem 19** (Contraction admissibility in `LBIZ`). *If $D : \Gamma(\Gamma_a; \Gamma_a) \vdash F$ is `LBIZ`-derivable, then so is $D' : \Gamma(\Gamma_a) \vdash F$. The derivation depth is preserved.*

Proof. By induction on derivation depth. The base cases are when it is 1, *i.e.* when D is the conclusion sequent of an axiom. Consider which axiom has applied. If it is $\top R$, then it is trivial to show that if $\Gamma(\Gamma_a; \Gamma_a) \vdash \top$, then so is $\Gamma(\Gamma_a) \vdash \top$. Also for $\bot L$, a single occurrence of $\bot$ on the antecedent part of D suffices for the $\bot L$ application, and the current theorem is trivially provable in this case, too. For both *id* and ${}^{*}\top R$, $\Pi(D)$ looks like:

$$\dfrac{}{\mathbb{E}(\widetilde{\Gamma_1}; \alpha) \vdash \alpha}$$

where α is $p \in \mathcal{P}$ for *id*, ${}^{*}\top$ for ${}^{*}\top R$ and $\Gamma(\Gamma_a; \Gamma_a) = \mathbb{E}(\widetilde{\Gamma_1}; \alpha)$. If α is not a sub-structure of either of the occurrences of Γ_a, then D' is trivially derivable. Otherwise, assume that the focused α in $\mathbb{E}(\widetilde{\Gamma_1}; \alpha)$ is a sub-structure of one of the occurrences of Γ_a in $\Gamma(\Gamma_a; \Gamma_a)$.

Then there exists some Γ_2 and $\widetilde{\Gamma_3}$ such that $\mathbb{E}(\widetilde{\Gamma_1}; \alpha) = \mathbb{E}(\Gamma_2; \widetilde{\Gamma_3}; \alpha) = \mathbb{E}_1(\Gamma_2); \mathbb{E}_2(\widetilde{\Gamma_3}; \alpha)$ and that Γ_a is an essence of $\widetilde{\Gamma_3}; \alpha$. But then $D' : \Gamma(\Gamma_a)$ is still an axiom.

For inductive cases, suppose that the current theorem holds true for any derivation depth of up to k. We must demonstrate that it still holds for the derivation depth of $k+1$. Consider what the `LBIZ` inference rule applied last is, and, in case of a left inference rule, consider where the active structure Γ_b of the inference rule is in $\Gamma(\Gamma_a; \Gamma_a)$.

1. $\wedge L$, and Γ_b is $F_1 \wedge F_2$: if Γ_b does not appear in Γ_a, induction hypothesis on the premise sequent concludes. Otherwise, $\Pi(D)$ looks like:
$$\dfrac{\begin{array}{c}\vdots\\ D_1 : \Gamma(\Gamma'_a(F_1; F_2); \Gamma'_a(F_1 \wedge F_2)) \vdash H\end{array}}{D : \Gamma(\Gamma'_a(F_1 \wedge F_2); \Gamma'_a(F_1 \wedge F_2)) \vdash H}\wedge L$$
$D'_1 : \Gamma(\Gamma'_a(F_1; F_2); \Gamma'_a(F_1; F_2)) \vdash H$ is `LBIZ`-derivable (inversion lemma);
$D''_1 : \Gamma(\Gamma'_a(F_1; F_2)) \vdash H$ is also `LBIZ`-derivable (induction hypothesis); then $\wedge L$ on D''_1 concludes.
2. $\supset L$, and Γ_b is $\mathbb{E}(\widetilde{\Gamma'}; F \supset G)$: if Γ_b does not appear in Γ_a, then the induction hypothesis on both of the premises concludes. If it is entirely in Γ_a, then $\Pi(D)$ looks either like:
$$\dfrac{\begin{array}{c}\vdots\\ D_1 : \mathbb{E}(\widetilde{\Gamma'}; F \supset G) \vdash F\end{array} \qquad \begin{array}{c}\vdots\\ D_2\end{array}}{D : \Gamma(\Gamma'_a(\mathbb{E}(\widetilde{\Gamma'}; F \supset G)); \widetilde{\Gamma'_a}(\mathbb{E}(\widetilde{\Gamma'}; F \supset G))) \vdash H}\supset L$$
where $D_2 : \Gamma(\Gamma'_a(G; \mathbb{E}(\widetilde{\Gamma'}; F{\supset}G)); \Gamma'_a(\mathbb{E}(\widetilde{\Gamma'}; F{\supset}G))) \vdash H$, or, in case Γ_a is $\Gamma'_a; F{\supset}G$, like:
$$\dfrac{\begin{array}{c}\vdots\\ D_1 : \Gamma'_a; F{\supset}G; \Gamma'_a; F{\supset}G \vdash F\end{array} \qquad \begin{array}{c}\vdots\\ D_2\end{array}}{D : \Gamma(\Gamma'_a; F{\supset}G; \Gamma'_a; F{\supset}G) \vdash H}\supset L$$
where $D_2 : \Gamma(G; \Gamma'_a; F{\supset}G; \Gamma'_a; F{\supset}G) \vdash H$.
In the former case,
$D'_2 : \Gamma(\Gamma'_a(G; \mathbb{E}(\widetilde{\Gamma'}; F{\supset}G)); \Gamma'_a(G; \mathbb{E}(\widetilde{\Gamma'}; F{\supset}G))) \vdash H$ (weakening admissibility);
$D''_2 : \Gamma(\Gamma'_a(G; \mathbb{E}(\widetilde{\Gamma'}; F{\supset}G))) \vdash H$ (induction hypothesis);
then $\supset L$ on D_1 and D''_2 concludes. In the latter, induction hypothesis on D_1 and on D_2; then via $\supset L$ for a conclusion. Finally, if only a substructure of Γ_b is in Γ_a with the rest spilling out of Γ_a, then if the principal formula $F \supset G$ does not occur in Γ_a, then straightforward; otherwise similar to the latter case.
3. $*R$: $\Pi(D)$ looks like:
$$\dfrac{\begin{array}{c}\vdots\\ D_1 : Re_i \vdash F_1\end{array} \qquad \begin{array}{c}\vdots\\ D_2 : Re_j \vdash F_2\end{array}}{D : \Gamma(\Gamma_a; \Gamma_a) \vdash F_1 * F_2}*R$$
By Proposition 18, assume that $(Re_1, Re_2) \in \texttt{RepCandidate}(\Gamma(\Gamma_a; \Gamma_a))$ without loss of generality. Then by the definition of $\hat{\preceq}$ it must be that either (1) $\Gamma_a; \Gamma_a$ preserves completely in Re_1 or Re_2, or (2) it remains neither in Re_1 nor in Re_2. If $\Gamma_a; \Gamma_a$ is preserved in Re_1 (or Re_2), then induction hypothesis on the premise that has Re_1 (or Re_2) and then $*R$ conclude; otherwise, it is trivial to see that only a single Γ_a needs to be present in D.
4. $-\!\!*L$, and Γ_b is $\widetilde{\Gamma'}, \mathbb{E}(\widetilde{\Gamma_1}; F{-\!\!*}G)$: if Γ_b is not in Γ_a, then induction hypothesis on the right premise sequent concludes. If it is in Γ_a, $\Pi(D)$ looks like:
$$\dfrac{\begin{array}{c}\vdots\\ D_1 : Re_i \vdash F\end{array} \qquad \begin{array}{c}\vdots\\ D_2\end{array}}{D : \Gamma(\Gamma'_a(\widetilde{\Gamma'}, \mathbb{E}(\widetilde{\Gamma_1}; F{-\!\!*}G)); \Gamma'_a(\widetilde{\Gamma'}, \mathbb{E}(\widetilde{\Gamma_1}; F{-\!\!*}G))) \vdash H}{-\!\!*}L_1$$

where D_2 is:

$$\Gamma(\Gamma'_a((\widetilde{Re_j}, G); (\widetilde{\Gamma'}, \mathbb{E}(\widetilde{\Gamma_1}; F{-\!*}G))); \Gamma'_a(\widetilde{\Gamma'}, \mathbb{E}(\widetilde{\Gamma_1}; F{-\!*}G))) \vdash H$$

$D'_2 : \Gamma(\Gamma'_a((\widetilde{Re_j}, G); (\widetilde{\Gamma'}, \mathbb{E}(\widetilde{\Gamma_1}; F{-\!*}G))); \Gamma'_a((\widetilde{Re_j}, G); (\widetilde{\Gamma'}, \mathbb{E}(\widetilde{\Gamma_1}; F{-\!*}G)))) \vdash H$ via Proposition 13 is also LBIZ-derivable. $D''_2 : \Gamma(\Gamma'_a((\widetilde{Re_j}, G); (\widetilde{\Gamma'}, \mathbb{E}(\widetilde{\Gamma_1}; F{-\!*}G)))) \vdash H$ via induction hypothesis. Then ${-\!*}L$ on D_1 and D''_2 concludes. If, on the other hand, Γ_a is in Γ_b, then it is either in Γ_1 or in Γ'. But if it is in Γ_1, then it must be weakened away, and if it is in Γ', similar to the $*R$ case.

5. Other cases are similar to one of the cases already examined. ◀

4.5 Equivalence of LBIZ to LBI

▶ **Theorem 20** (Equivalence between LBIZ and LBI). *$D : \Gamma \vdash F$ is* LBIZ*-derivable if and only if it is* LBI*-derivable.*

Proof. Into the *only if* direction, assume that D is LBIZ-derivable, and then show that there is a LBI-derivation for each LBIZ derivation. But this is obvious because each LBIZ inference rule is derivable in LBI.[8]

Into the *if* direction, assume that D is LBI-derivable, and then show that there is a corresponding LBIZ-derivation to each LBI derivation by induction on the derivation depth of D.

If it is 1, *i.e.* if D is the conclusion sequent of an axiom, we note that $\bot L_{\texttt{LBI}}$ is identical to $\bot L_{\texttt{LBIZ}}$; $id_{\texttt{LBI}}$ and ${}^*\!\top R_{\texttt{LBI}}$ via $id_{\texttt{LBIZ}}$ and resp. ${}^*\!\top R_{\texttt{LBIZ}}$ with Proposition 13 and Proposition 14; and $\top R_{\texttt{LBI}}$ is identical to $\top R_{\texttt{LBIZ}}$. For inductive cases, assume that the *if* direction holds true up to the LBI-derivation depth of k, then it must be demonstrated that it still holds true for the LBI-derivation depth of $k+1$. Consider what the LBI rule applied last is:

1. $\supset L_{\texttt{LBI}}$: $\Pi_{\texttt{LBI}}(D)$ looks like:

$$\dfrac{\begin{array}{c}\vdots\\ D_1 : \Gamma_1 \vdash F\end{array} \qquad \begin{array}{c}\vdots\\ D_2 : \Gamma(\Gamma_1; G) \vdash H\end{array}}{D : \Gamma(\Gamma_1; F \supset G) \vdash H} \supset L_{\texttt{LBI}}$$

By induction hypothesis, both D_1 and D_2 are also LBIZ-derivable. Proposition 13 on D_1 in LBIZ-space results in $D'_1 : \Gamma_1; F \supset G \vdash F$, and on D_2 results in $D'_2 : \Gamma(\Gamma_1; G; F \supset G) \vdash H$. Then an application of $\supset L_{\texttt{LBIZ}}$ on D'_1 and D_2 concludes in LBIZ-space.

2. ${-\!*}L_{\texttt{LBI}}$: $\Pi_{\texttt{LBI}}(D)$ looks like:

$$\dfrac{\begin{array}{c}\vdots\\ D_1 : \Gamma_1 \vdash F\end{array} \qquad \begin{array}{c}\vdots\\ D_2 : \Gamma(G) \vdash H\end{array}}{D : \Gamma(\Gamma_1, F{-\!*}G) \vdash H} {-\!*}L_{\texttt{LBI}}$$

By induction hypothesis, D_1 and D_2 are also LBIZ-derivable.

   a. If $\Gamma(G)$ is G, *i.e.* if the antecedent part of D_2 is a formula (G), then Proposition 13 on D_2 results in $D'_2 : G; (\Gamma_1, F{-\!*}G) \vdash H$ in LBIZ-space. Then ${-\!*}L_{\texttt{LBIZ}}$ on D_1 and D'_2 leads to $D' : \Gamma_1, F{-\!*}G \vdash H$ as required.

   b. If $\Gamma(G)$ is $\Gamma'(\Gamma'', G)$, then Proposition 13 on D_2 leads to $D'_2 : \Gamma'((\Gamma'', G); (\Gamma'', \Gamma_1, F{-\!*}G)) \vdash H$. Then ${-\!*}L_{\texttt{LBIZ}}$ on D_1 and D'_2 leads to $D' : \Gamma'(\Gamma'', \Gamma_1, F{-\!*}G) \vdash H$ as required.

[8] Note that EA_2 is LBI-derivable with $WkL_{\texttt{LBI}}$ and $EqAnt_{2\,\texttt{LBI}}$.

   c. Finally, if $\Gamma(G)$ is $\Gamma'(\Gamma''; G) \vdash H$, then Proposition 13 on D_2 gives $D'_2 : \Gamma'(\Gamma''; G; (\Gamma_1, F \mathrel{-\!\!*} G)) \vdash H$. Then $\mathrel{-\!\!*}L_{\texttt{LBIZ}}$ on D_1 and D'_2 leads to $D' : \Gamma'(\Gamma''; (\Gamma_1, F \mathrel{-\!\!*} G)) \vdash H$ as required.

3. $WkL_{\texttt{LBI}}$: Proposition 13.
4. $CtrL_{\texttt{LBI}}$: Theorem 19.
5. $EqAnt_{1\,\texttt{LBI}}$: Proposition 16.
6. $EqAnt_{2\,\texttt{LBI}}$: Proposition 16.
7. The rest: straightforward. ◀

4.6 LBIZ Cut Elimination

Cut is admissible in LBIZ. As a reminder (although already stated under Figure 1) Cut is the following rule:

$$\frac{\Gamma_1 \vdash F \qquad \Gamma_2(F) \vdash G}{\Gamma_2(\Gamma_1) \vdash G}\,\texttt{Cut}$$

Just as in the case of intuitionistic logic, cut admissibility proof for a contraction-free BI sequent calculus is simpler than that for LBI [2]. Since we have already proved depth-preserving weakening admissibility, the following context sharing cut, $\texttt{Cut}_{CS}$, is easily verified derivable in LBIZ + Cut:

$$\frac{\widetilde{\Gamma_3}; \Gamma_1 \vdash F \qquad \Gamma_2(F; \Gamma_1) \vdash H}{\Gamma_2(\widetilde{\Gamma_3}; \Gamma_1) \vdash H}\,\texttt{Cut}_{CS}$$

where Γ_1 appears on both of the premises. F in the above cut rule appearing on both premises is called the cut formula. The use of $\texttt{Cut}_{CS}$ simplifies the cut elimination proof a little.

We recall the standard notations of the cut rank and the cut level.

▶ **Definition 21** (Cut level/rank). Given a cut instance in a closed derivation:

$$\frac{D_1 : \Gamma_1 \vdash F \qquad D_2 : \Gamma_2(F) \vdash H}{D_3 : \Gamma_2(\Gamma_1) \vdash H}\,\texttt{Cut}$$

The level of the cut instance is: $\texttt{der\_depth}(D_1) + \texttt{der\_depth}(D_2)$, where $\texttt{der\_depth}(D)$ denotes derivation depth of D. The rank of the cut instance is the size of the cut formula F, $\texttt{f\_size}(F)$, which is defined as follows:

- it is 1 if F is a nullary logical connective or a propositional variable.
- it is $\texttt{f\_size}(F_1) + \texttt{f\_size}(F_2) + 1$ if F is in the form: $F_1 \bullet F_2$ for $\bullet \in \{\wedge, \vee, \supset, *, \mathrel{-\!\!*}\}$.

▶ **Theorem 22** (Cut admissibility in LBIZ). Cut *is admissible in* LBIZ.

Proof. By induction on the cut rank and a sub-induction on the cut level. We make use of $\texttt{Cut}_{CS}$. In this proof (X, Y) for some LBIZ inference rules X and Y means that one of the premises has been just derived with X and the other with Y. $\Gamma(\Gamma_1)(\Gamma_2)$ abbreviates $(\Gamma(\Gamma_1))(\Gamma_2)$. In pairs of derivations below, the first is the derivation tree to be permuted and the second is the permuted derivation tree.

***(id, id)*:**

1.

$$\frac{\dfrac{}{\mathbb{E}(\widetilde{\Gamma_1}; p) \vdash p}\,id \qquad \dfrac{}{\mathbb{E}'(\widetilde{\Gamma_2}; p) \vdash p}\,id}{\mathbb{E}'(\widetilde{\Gamma_2}; \mathbb{E}(\widetilde{\Gamma_1}; p)) \vdash p}\,\texttt{Cut}$$

⇒

$$\dfrac{}{\mathbb{E}'(\widetilde{\Gamma_2};\mathbb{E}(\widetilde{\Gamma_1};p)) \vdash p}\,\mathit{id}$$

Of course, for the above permutation to be correct, we must be able to demonstrate the fact that the antecedent structure $\mathbb{E}''(\widetilde{\Gamma_2};\widetilde{\Gamma_1};p)$ is such that $[\mathbb{E}''(\widetilde{\Gamma_2};\widetilde{\Gamma_1};p)] = [\mathbb{E}'(\widetilde{\Gamma_2};\mathbb{E}(\widetilde{\Gamma_1};p))]$. But note that it only takes a finite number of (backward) EA_2 applications (*Cf.* Proposition 14) on $\widetilde{\Gamma_2};\mathbb{E}(\widetilde{\Gamma_1};p) \vdash p$ to upward derive $\widetilde{\Gamma_2};\widetilde{\Gamma_1};p \vdash p$. The implication is that, since $\widetilde{\Gamma_2};\mathbb{E}(\widetilde{\Gamma_1};p) \vdash p$ results upward from $\mathbb{E}'(\widetilde{\Gamma_2};\mathbb{E}(\widetilde{\Gamma_1};p)) \vdash p$ also in a finite number of backward EA_2 applications, the antecedent structure must be in the form: $\mathbb{E}''(\widetilde{\Gamma_2};\widetilde{\Gamma_1};p)$.

2.

$$\dfrac{\dfrac{}{\mathbb{E}(\widetilde{\Gamma_1};p) \vdash p}\,\mathit{id} \qquad \dfrac{}{\mathbb{E}'(\Gamma_2(p);q) \vdash q}\,\mathit{id}}{\mathbb{E}'(\Gamma_2(\mathbb{E}(\widetilde{\Gamma_1};p));q) \vdash q}\,\texttt{Cut}$$

This and the other patterns for which one of the premises is an axiom sequent are straightforward.

For the remaining cases, if the cut formula is principal only for one of the premise sequents, then we follow the routine [25] to permute up the other premise sequent for which it is the principal. For example, in case we have the derivation pattern below:

$$\dfrac{\dfrac{D_1 \qquad D_2}{D_5 : \Gamma_1(H_1 \vee H_2) \vdash F_1 \supset F_2}\,\vee L \qquad \dfrac{D_3 : \mathbb{E}(\widetilde{\Gamma_3};F_1 \supset F_2) \vdash F_1 \qquad D_4 : \Gamma_2(F_2;\mathbb{E}(\widetilde{\Gamma_3};F_1 \supset F_2)) \vdash H}{D_6 : \Gamma_2(\mathbb{E}(\widetilde{\Gamma_3};F_1 \supset F_2)) \vdash H}\,\supset L}{\Gamma_2(\mathbb{E}(\widetilde{\Gamma_3};\Gamma_1(H_1 \vee H_2))) \vdash H}\,\texttt{Cut}$$

for $D_1 : \Gamma_1(H_1) \vdash F_1 \supset F_2$ and $D_2 : \Gamma_1(H_2) \vdash F_1 \supset F_2$, the cut formula $F_1 \supset F_2$ is not the principal on the left premise. In this case, we simply apply `Cut` on the pairs: (D_1, D_6) and (D_2, D_6), to conclude:

$$\dfrac{\dfrac{D_1 \qquad D_6}{\Gamma_2(\mathbb{E}(\widetilde{\Gamma_3};\Gamma_1(H_1))) \vdash H}\,\texttt{Cut} \qquad \dfrac{D_2 \qquad D_6}{\Gamma_2(\mathbb{E}(\widetilde{\Gamma_3};\Gamma_1(H_2))) \vdash H}\,\texttt{Cut}}{\Gamma_2(\mathbb{E}(\widetilde{\Gamma_3};\Gamma_1(H_1 \vee H_2))) \vdash H}\,\vee L$$

Of course, for this particular permutation to be correct, we must be able to demonstrate, in the permuted derivation tree, that $\mathbb{E}(\widetilde{\Gamma_3};\Gamma_1(H_1 \vee H_2)) = \mathbb{E}'(\widetilde{\Gamma_3}) \star \Gamma_1(H_1 \vee H_2)$ with $\star$ either a semi-colon or a comma, that $\mathbb{E}(\widetilde{\Gamma_3};\Gamma_1(H_1)) = \mathbb{E}'(\widetilde{\Gamma_3}) \star \Gamma_1(H_1)$, and that $\mathbb{E}(\widetilde{\Gamma_3};\Gamma_1(H_2)) = \mathbb{E}'(\widetilde{\Gamma_3}) \star \Gamma_1(H_2)$. But this is vacuous since the cut formula which is replaced by the structure $\Gamma_1(H_1)$ or $\Gamma_1(H_2)$ is a formula.

The cases that remain are those for which both premises of the cut instance have the cut formula as the principal. We go through each of them to conclude the proof.

$(\wedge L, \wedge R)$:

$$\dfrac{\dfrac{D_1 : \Gamma_1 \vdash F_1 \qquad D_2 : \Gamma_1 \vdash F_2}{\Gamma_1 \vdash F_1 \wedge F_2}\,\wedge R \qquad \dfrac{D_3 : \Gamma_2(F_1;F_2) \vdash H}{\Gamma_2(F_1 \wedge F_2) \vdash H}\,\wedge L}{\Gamma_2(\Gamma_1) \vdash H}\,\texttt{Cut}$$

⇒

$$\dfrac{D_2 \qquad \dfrac{D_1 \qquad D_3}{\Gamma_2(\Gamma_1; F_2) \vdash H}\,\mathtt{Cut}}{\Gamma_2(\Gamma_1) \vdash H}\,\mathtt{Cut}_{CS}$$

($\vee L, \vee R$):

$$\dfrac{\dfrac{D_1 : \Gamma_1 \vdash F_i \quad (i \in \{1,2\})}{\Gamma_1 \vdash F_1 \vee F_2}\,\vee R \qquad \dfrac{D_2 : \Gamma_2(F_1) \vdash H \qquad D_3 : \Gamma_2(F_2) \vdash H}{\Gamma_2(F_1 \vee F_2) \vdash H}\,\vee L}{\Gamma_2(\Gamma_1) \vdash H}\,\mathtt{Cut}$$

$\Rightarrow$

$$\dfrac{D_1 \qquad D_{(2\ or\ 3)}}{\Gamma_2(\Gamma_1) \vdash H}\,\mathtt{Cut}$$

The value of i decides which of D_2 or D_3 is the right premise sequent.

($\supset L, \supset R$):

$$\dfrac{\dfrac{D_1 : \Gamma_3; F_1 \vdash F_2}{D_4 : \Gamma_3 \vdash F_1 \supset F_2}\supset R \qquad \dfrac{D_2 : \mathbb{E}(\widetilde{\Gamma_1}; F_1 \supset F_2) \vdash F_1 \qquad D_3 : \Gamma_2(F_2; \mathbb{E}(\widetilde{\Gamma_1}; F_1 \supset F_2)) \vdash H}{\Gamma_2(\mathbb{E}(\widetilde{\Gamma_1}; F_1 \supset F_2)) \vdash H}\supset L}{\Gamma_2(\mathbb{E}(\widetilde{\Gamma_1}; \Gamma_3)) \vdash H}\,\mathtt{Cut}$$

$\Rightarrow$

$$\begin{array}{c}
\dfrac{\dfrac{\dfrac{D_4 \qquad D_2}{\mathbb{E}(\widetilde{\Gamma_1}; \Gamma_3) \vdash F_1}\,\mathtt{Cut} \qquad D_1}{\Gamma_3; \mathbb{E}(\widetilde{\Gamma_1}; \Gamma_3) \vdash F_2}\,\mathtt{Cut} \qquad \dfrac{D_4 \qquad D_3}{\Gamma_2(F_2; \mathbb{E}(\widetilde{\Gamma_1}; \Gamma_3)) \vdash H}\,\mathtt{Cut}}{\Gamma_2(\Gamma_3; \mathbb{E}(\widetilde{\Gamma_1}; \Gamma_3)) \vdash H}\,\mathtt{Cut}_{CS} \\
\cdots\cdots\cdots\cdots\cdots\cdots \ \text{Proposition 13} \\
\Gamma_2(\widetilde{\Gamma_1}; \Gamma_3; \mathbb{E}(\widetilde{\Gamma_1}; \Gamma_3)) \vdash H \\
\cdots\cdots\cdots\cdots\cdots\cdots \ \text{Proposition 14} \\
\Gamma_2(\mathbb{E}(\widetilde{\Gamma_1}; \Gamma_3); \mathbb{E}(\widetilde{\Gamma_1}; \Gamma_3)) \vdash H \\
\cdots\cdots\cdots\cdots\cdots\cdots \ \text{Theorem 19} \\
\Gamma_2(\mathbb{E}(\widetilde{\Gamma_1}; \Gamma_3)) \vdash H
\end{array}$$

The derivation steps with a dotted line are depth-preserving.

($*L, *R$):

$$\dfrac{\dfrac{D_1 : Re_i \vdash F_1 \qquad D_2 : Re_j \vdash F_2}{\Gamma_1 \vdash F_1 * F_2}\,*R \qquad \dfrac{D_3 : \Gamma_2(F_1, F_2) \vdash H}{\Gamma_2(F_1 * F_2) \vdash H}\,*L}{\Gamma_2(\Gamma_1) \vdash H}\,\mathtt{Cut}$$

$\Rightarrow$

$$\begin{array}{c}
\dfrac{D_2 \qquad \dfrac{D_1 \qquad D_3}{\Gamma_2(Re_i, F_2) \vdash H}\,\mathtt{Cut}}{\Gamma_2(Re_i, Re_j) \vdash H}\,\mathtt{Cut} \\
\cdots\cdots\cdots\cdots\cdots \ \text{Proposition 13} \\
\Gamma_2(\Gamma_1) \vdash H
\end{array}$$

($-\!\!*L, -\!\!*R$):

$$\dfrac{\dfrac{D_1 : \Gamma_1, F_1 \vdash F_2}{D_4 : \Gamma_1 \vdash F_1 -\!\!* F_2}\,-\!\!*R \qquad \dfrac{D_2 : Re_i \vdash F_1 \qquad D_3 : \Gamma_2((\widetilde{Re_j}, F_2); (\widetilde{\Gamma}', \mathbb{E}(\widetilde{\Gamma_3}; F_1 -\!\!* F_2))) \vdash H}{\Gamma_2(\widetilde{\Gamma}', \mathbb{E}(\widetilde{\Gamma_3}; F_1 -\!\!* F_2)) \vdash H}\,-\!\!*L_1}{\Gamma_2(\widetilde{\Gamma}', \mathbb{E}(\widetilde{\Gamma_3}; \Gamma_1)) \vdash H}\,\mathtt{Cut}$$

$\Rightarrow$

$$\begin{array}{ll}
\dfrac{D_2 \qquad \dfrac{D_1 \qquad \dfrac{D_4 \qquad D_3}{\Gamma_2((\widetilde{Re_j}, F_2); (\widetilde{\Gamma}', \mathbb{E}(\widetilde{\Gamma_3}; \Gamma_1))) \vdash H}\,\texttt{Cut}}{\Gamma_2((\widetilde{Re_j}, \Gamma_1, F_1); (\Gamma', \mathbb{E}(\widetilde{\Gamma_3}; \Gamma_1))) \vdash H}\,\texttt{Cut}}{\Gamma_2((\widetilde{Re_j}, \Gamma_1, Re_i); (\widetilde{\Gamma}', \mathbb{E}(\widetilde{\Gamma_3}; \Gamma_1))) \vdash H}\,\texttt{Cut} & \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \text{Proposition 13} \\
\Gamma_2((\widetilde{\Gamma}', (\widetilde{\Gamma_3}; \Gamma_1)); (\widetilde{\Gamma}', \mathbb{E}(\widetilde{\Gamma_3}; \Gamma_1))) \vdash H & \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \text{Proposition 14} \\
\Gamma_2((\widetilde{\Gamma}', \mathbb{E}(\widetilde{\Gamma_3}; \Gamma_1)); (\widetilde{\Gamma}', \mathbb{E}(\widetilde{\Gamma_3}; \Gamma_1))) \vdash H & \\
\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \text{Theorem 19} \\
\Gamma_2(\widetilde{\Gamma}', \mathbb{E}(\widetilde{\Gamma_3}; \Gamma_1)) \vdash H &
\end{array}$$

◀

5 Conclusion

We addressed the problem of structural rule absorption in BI sequent calculus. This problem was around for a while. As far back as we can see, the first proximate attempt was made in [21]. References to the problem were subsequently made [9, 21, 4] in a discussion. The work that came closest to ours is one by Donnelly *et al.* [7]. They consider weakening absorption in the context of forward theorem proving (where weakening rather than contraction is a source of non-termination). One inconvenience in their approach, however, is that the effect of weakening is not totally isolated from that of contraction: it is absorbed into contraction as well as into logical rules. But then structural weakening is still possible through the new structural contraction. Also, the coupling of the two structural rules amplifies the difficulty of analysis on the behaviour of contraction. Further, their work is on a subset of BI without units. In comparison, our solution covers the whole BI. Techniques we used in this work should be useful in the derivation of contraction-free sequent calculi of other non-classical logics that come with a non-formula structural contraction rule. For instance, nested sequent calculi [15, 12, 17] of some constructive modal logics (those only with k1 and k2 axioms) [23], when they are extended with additional modal axioms including 5 axiom, are known to truly require non-formula contractions, in the presence of which cut-elimination proof becomes demanding. As is always the case, there are fewer cases to cover in cut-elimination proof when there are no structural contraction. There are also more recent BI extensions in sequent calculus such as [14], to which this work has relevance. Seeing the complexity of LBIZ, one may also consider development of another formalism that may represent BI and other similar non-classical logics more informatively.

Acknowledgements. I thank two reviewers for this paper for kindly pointing out typographical errors. This contribution is also in Section 3 of my PhD Thesis written a while back, and my PhD study was supported in part by an EPSRC project EP/G042322.

References

1 Ryuta Arisaka, Anupam Das, and Lutz Straßburger. On Nested Sequents for Constructive Modal Logics. *Logical Methods in Computer Science*, 11(3), 2015.

2 Ryuta Arisaka and Shengchao Qin. LBI cut elimination proof with BI-Multi-Cut. In *TASE*, pages 235–238, 2012.

3 Nuel Belnap. Display logic. *Journal of Philosophical Logic*, 11(4):375–417, 1982.

4 James Brotherston and Cristiano Calcagno. Classical BI: a logic for reasoning about dualising resources. In *POPL*, pages 328–339, 2009.

5 Carlos Caleiro and Jaime Ramos. Combining classical and intuitionistic implications. In *Frontiers of Combining Systms*, pages 118–132, 2007.

6 Luis Fariñas del Cerro and Andreas Herzig. Combining classical and intuitionistic logic, or: Intuitionistic implication as a conditional. In *Frontiers of Combining Systms*, pages 93–102, 1996.

7 Kevin Donnelly, Tyler Gibson, Neel Krishnaswami, Stephen Magill, and Sungwoo Park. The inverse method for the logic of bunched implications. In *LPAR*, pages 466–480, 2004.

8 Roy Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *J. Symb. Log.*, 57(3):795–807, 1992.

9 Didier Galmiche, Daniel Méry, and David J. Pym. The semantics of BI and resource tableaux. *Mathematical Structures in Computer Science*, 15(6):1033–1088, 2005. `doi:10.1017/S0960129505004858`.

10 Steve Giambrone. TW_+ AND RW_+ ARE DECIDABLE. *Journal of Philosophical Logic*, 14(3):235–254, 1985.

11 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

12 Rajeev Goré, Linda Postniece, and Alwen Tiu. Cut-elimination and proof search for bi-intuitionistic tense logic. *CoRR*, abs/1006.4793, 2010.

13 Haskell B. Curry. *A Theory of Formal Deductibility*. University of Notre Dame, 1950.

14 Norihiro Kamide. Temporal BI: Proof system, semantics and translations. *Theor. Comput. Sci.*, 492:40–69, 2013.

15 Ryo Kashima. Cut-free sequent calculi for some tense logics. *Studia Logica*, 53(1):119–135, 1994.

16 Stephen C. Kleene. *Introduction to META-MATHEMATICS*. North-Holland Publishing Co., 1952.

17 Sonia Marin and Lutz Straßburger. Label-free Modular Systems for Classical and Intuitionistic Modal Logics. In *Advances in Modal Logic*, pages 387–406, 2014.

18 Robert K. Meyer and Michael A. McRobbie. Multisets and relevant implication I. *Australasian Journal of Philosophy*, 60(3):107–139, 1982.

19 Robert K. Meyer and Michael A. McRobbie. Multisets and relevant implication II. *Australasian Journal of Philosophy*, 60(3):265–281, 1982.

20 Michael Moortgat. Categorial type logics. *HANDBOOK OF LOGIC AND LANGUAGE*, pages 93–177, 1997.

21 Peter W. O'Hearn. On bunched typing. *Journal of Functional Programming*, 13(4):747–796, 2003.

22 Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

23 Dag Prawitz. *Natural Deduction – A Proof-Theoretical Study*. Almquist and Wiksell, 1965.

24 David J. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*. Kluwer Academic Publishers, 2002.

25 Anne S. Troelstra and Helmut Schwichtenberg. *Basic proof theory (2nd ed.)*. Cambridge University Press, 2000.

Environmental Bisimulations for Delimited-Control Operators with Dynamic Prompt Generation[*]

Andrés Aristizábal[1], Dariusz Biernacki[2], Sergueï Lenglet[3], and Piotr Polesiuk[4]

1 **Pontificia Universidad Javeriana Cali, Cali, Columbia**
aaaristizabal@javerianacali.edu.co
2 **University of Wrocław, Wrocław, Poland**
dabi@cs.uni.wroc.pl
3 **Université de Lorraine, Nancy, France**
serguei.lenglet@univ-lorraine.fr
4 **University of Wrocław, Wrocław, Poland**
ppolesiuk@cs.uni.wroc.pl

Abstract

We present sound and complete environmental bisimilarities for a variant of Dybvig et al.'s calculus of multi-prompted delimited-control operators with dynamic prompt generation. The reasoning principles that we obtain generalize and advance the existing techniques for establishing program equivalence in calculi with single-prompted delimited control.

The basic theory that we develop is presented using Madiot et al.'s framework that allows for smooth integration and composition of up-to techniques facilitating bisimulation proofs. We also generalize the framework in order to express environmental bisimulations that support equivalence proofs of evaluation contexts representing continuations. This change leads to a novel and powerful up-to technique enhancing bisimulation proofs in the presence of control operators.

1998 ACM Subject Classification D.3.3 Language Constructs and Features, F.3.2 Semantics of Programming Languages

Keywords and phrases delimited continuation, dynamic prompt generation, contextual equivalence, environmental bisimulation, up-to technique

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.9

1 Introduction

Control operators for delimited continuations, introduced independently by Felleisen [12] and by Danvy and Filinski [9], allow the programmer to delimit the current context of computation and to abstract such a delimited context as a first-class value. It has been shown that all computational effects are expressible in terms of delimited continuations [13], and so there exists a large body of work devoted to this canonical control structure, including our work on a theory of program equivalence for the operators shift and reset [5, 6, 7].

In their paper on type-directed partial evaluation for typed λ-calculus with sums, Balat et al. [2] have demonstrated that Gunter et al.'s delimited-control operators set and cupto [15], that support multiple prompts along with dynamic prompt generation, can have a practical advantage over single-prompted operators such as shift and reset. Delimited-control operators

[*] This work was partially supported by PHC Polonium and by National Science Centre, Poland, grant no. 2014/15/B/ST6/00619.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 9; pp. 9:1–9:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

with dynamically-generated prompts are now available in several production programming languages such as OCaml [19] and Racket [14], and they have been given formal semantic treatment in the literature. In particular, Dybvig et al. [11] have proposed a calculus that extends the call-by-value λ-calculus with several primitives that allow for: fresh-prompt generation, delimiting computations with a prompt, abstracting control up to the corresponding prompt, and composing captured continuations. Dybvig et al.'s building blocks were shown to be able to naturally express most of other existing control operators and as such they form a general framework for studying delimited continuations. Reasoning about program equivalence in Dybvig et al.'s calculus is considerably more challenging than in single-prompted calculi: one needs to reconcile control effects with the intricacies introduced by fresh-prompt generation and local visibility of prompts.

In this article we investigate the behavioral theory of a slightly modified version of Dybvig et al.'s calculus that we call the $\lambda_{\mathcal{G}\#}$-calculus. One of the most natural notions of program equivalence in languages based on the λ-calculus is *contextual equivalence*: two terms are contextually equivalent if we cannot distinguish them when evaluated within any context. The quantification over contexts makes this relation hard to use in practice, so it is common to characterize it using simpler relations, like coinductively defined *bisimilarities.* As pointed out in [21], among the existing notions of bisimilarities, *environmental bisimilarity* [29] is the most appropriate candidate to characterize contextual equivalence in a calculus with generated resources, such as prompts in $\lambda_{\mathcal{G}\#}$. Indeed, this bisimilarity features an environment which accumulates knowledge about the terms we compare. This is crucial in our case to remember the relationships between the prompts generated by the compared programs. We therefore define environmental bisimilarities for $\lambda_{\mathcal{G}\#}$, as well as *up-to techniques*, which are used to simplify the equivalence proof of two given programs. We do so using the recently developed framework of Madiot et al. [25, 24], where it is simpler to prove that a bisimilarity and its up-to techniques are *sound* (i.e., imply contextual equivalence).

After presenting the syntax, semantics, and contextual equivalence of the calculus in Section 2, in Section 3 we define a sound and complete environmental bisimilarity and its corresponding up-to techniques. In particular, we define a bisimulation *up to context*, which allows to forget about a common context when comparing two terms in a bisimulation proof. The bisimilarity we define is useful enough to prove, e.g., the folklore theorem about delimited control [4] expressing that the static delimited-control operators shift and reset [9] can be simulated by the dynamic control operators control and prompt [12]. The technique, however, in general requires a cumbersome analysis of terms of the form $E[e]$, where E is a captured evaluation context and e is any expression (not necessarily a value). We therefore define in Section 4 a refined bisimilarity, called $\star$-bisimilarity, and a more expressive bisimulation up to context, which allows to factor out a context built with captured continuations. Proving the soundness of these two relations requires us to extend Madiot et al.'s framework. These results non-trivially generalize and considerably improve the existing techniques [7]. Finally, we discuss related work and conclude in Section 5. An accompanying research report [1] contains the proofs.

2 The Calculus $\lambda_{\mathcal{G}\#}$

The calculus we consider, called $\lambda_{\mathcal{G}\#}$, extends the call-by-value λ-calculus with four building blocks for constructing delimited-control operators as first proposed by Dybvig et al. [11]. [1]

[1] Dybvig et al.'s control operators slightly differ from their counterparts considered in this work, but they can be straightforwardly macro-expressed in the $\lambda_{\mathcal{G}\#}$-calculus.

Syntax. We assume we have a countably infinite set of term variables, ranged over by x, y, z, and k, as well as a countably infinite set of prompts, ranged over by p, q. Given an entity denoted by a meta-variable m, we write $\overrightarrow{m}$ for a (possibly empty) sequence of such entities. Expressions (e), values (v), and evaluation contexts (E) are defined as follows:

$$\begin{array}{rcll} e & ::= & v \mid e\,e \mid \mathcal{P}x.e \mid \#_v e \mid \mathcal{G}_v x.e \mid v \triangleleft e & \text{(expressions)} \\ v & ::= & x \mid \lambda x.e \mid p \mid \ulcorner E \urcorner & \text{(values)} \\ E & ::= & \Box \mid E\,e \mid v\,E \mid \#_p E & \text{(evaluation contexts)} \end{array}$$

Values include captured evaluation contexts $\ulcorner E \urcorner$, representing delimited continuations, as well as generated prompts p. Expressions include the four building blocks for delimited control: $\mathcal{P}x.e$ is a prompt-generating construct, where x represents a fresh prompt locally visible in e, $\#_v e$ is a control delimiter for e, $\mathcal{G}_v x.e$ is a continuation grabbing or capturing construct, and $v \triangleleft e$ is a throw construct.

Evaluation contexts, in addition to the standard call-by-value contexts, include delimited contexts of the form $\#_p E$, and they are interpreted outside-in. We use the standard notation $E[e]$ ($E[E']$) for plugging a context E with an expression e (with a context E'). Evaluation contexts are a special case of (general) contexts, understood as a term with a hole and ranged over by C.

The expressions $\lambda x.e$, $\mathcal{P}x.e$, and $\mathcal{G}_v x.e$ bind x; we adopt the standard conventions concerning α-equivalence. If x does not occur in e, we write $\lambda\_.e$, $\mathcal{P}\_.e$, and $\mathcal{G}_v\_.e$. The set of free variables of e is written $\mathsf{fv}(e)$; a term e is called closed if $\mathsf{fv}(e) = \emptyset$. We extend these notions to evaluation contexts. We write $\#(e)$ (or $\#(E)$) for the set of all prompts that occur in e (or E respectively). The set $\mathsf{sp}(E)$ of surrounding prompts in E is the set of all prompts guarding the hole in E, defined as $\{p \mid \exists E_1, E_2, E = E_1[\#_p E_2]\}$.

Reduction semantics. The reduction semantics of $\lambda_{\mathcal{G}\#}$ is given by the following rules:

$$\begin{array}{rcll} (\lambda x.e)\,v & \to & e\{v/x\} & \\ \#_p v & \to & v & \\ \#_p E[\mathcal{G}_p x.e] & \to & e\{\ulcorner E \urcorner/x\} & p \notin \mathsf{sp}(E) \\ \ulcorner E \urcorner \triangleleft e & \to & E[e] & \\ \mathcal{P}x.e & \to & e\{p/x\} & p \notin \#(e) \end{array}$$

$$\textsc{Compatibility}\quad \frac{e_1 \to e_2 \qquad \mathsf{fresh}(e_2, e_1, E)}{E[e_1] \to E[e_2]}$$

The first rule is the standard β_v-reduction. The second rule signals that a computation has been completed for a given prompt. The third rule abstracts the evaluation context up to the dynamically nearest control delimiter matching the prompt of the grab operator. In the fourth rule, an expression is thrown (plugged, really) to the captured context. Note that, like in Dybvig et al.'s calculus, the expression e is not evaluated before the throw operation takes place. In the last rule, a prompt p is generated under the condition that it is fresh for e.

The compatibility rule needs a side condition, simply because a prompt that is fresh for e may not be fresh for a surrounding evaluation context. Given three entities m_1, m_2, m_3 for which $\#$ is defined, we write $\mathsf{fresh}(m_1, m_2, m_3)$ for the condition $(\#(m_1)\backslash\#(m_2))\cap\#(m_3) = \varnothing$, so the side condition states that E must not mention prompts generated in the reduction step $e_1 \to e_2$. This approach differs from the previous work on bisimulations for resource-generating constructs [23, 22, 30, 31, 32, 3, 26], where configurations of the operational semantics contain explicit information about the resources, typically represented by a set. We find our way of proceeding less invasive to the semantics of the calculus.

When reasoning about reductions in the $\lambda_{\mathcal{G}\#}$-calculus, we rely on the notion of *permutation* (a bijection on prompts), ranged over by σ, which allows to reshuffle the prompts of an expression to avoid potential collisions (e with prompts permuted by σ is written $e\sigma$). E.g., we can use the first item of the following lemma before applying the compatibility rule, to be sure that any prompt generated by $e_1 \to e_2$ is not in $\#(E)$.

▶ **Lemma 1.** *Let σ be a permutation.*

- *If $e_1 \to e_2$ then $e_1\sigma \to e_2\sigma$.*
- *For any entities m_1, m_2, m_3, we have $\mathsf{fresh}(m_1, m_2, m_3)$ iff $\mathsf{fresh}(m_1\sigma, m_2\sigma, m_3\sigma)$.*

A closed term e either uniquely, up to permutation of prompts, reduces to a term e', or it is a normal form (i.e., there is no e'' such that $e \to e''$). In the latter case, we distinguish values, control-stuck terms $E[\mathcal{G}_p k.e]$ where $p \notin \mathsf{sp}(E)$, and the remaining expressions that we call errors (e.g., $E[p\, v]$ or $E[\mathcal{G}_{\lambda x.e} k.e']$). We write $e_1 \to^* e_2$ if e_1 reduces to e_2 in many (possibly 0) steps, and we write $e \lightning$ when a term e diverges (i.e., there exists an infinite sequence of reductions starting with e) or when it reduces (in many steps) to an error.

When writing examples, we use the fixed-point operator fix, let-construct, conditional if along with boolean values true and false, and sequencing ";", all defined as in the call-by-value λ-calculus. We also use the diverging term $\Omega \overset{\mathsf{def}}{=} (\lambda x.x\, x)\, (\lambda x.x\, x)$, and we define an operator $\overset{?}{=}$ to test the equality between prompts, as follows:

$$e_1 \overset{?}{=} e_2 \quad \overset{\mathsf{def}}{=} \quad \mathsf{let}\ x = e_1\ \mathsf{in}\ \mathsf{let}\ y = e_2\ \mathsf{in}\ \#_x((\#_y \mathcal{G}_x \_.\mathsf{false});\mathsf{true})$$

If e_1 and e_2 evaluate to different prompts, then the grab operator captures the context up to the outermost prompt to throw it away, and false is returned; otherwise, true is returned.

Contextual equivalence. We now define formally what it takes for two terms to be considered equivalent in the $\lambda_{\mathcal{G}\#}$-calculus. First, we characterize when two closed expressions have equivalent observable actions in the calculus, by defining the following relation $\sim$.

▶ **Definition 2.** We say that e_1 and e_2 have equivalent observable actions, noted $e_1 \sim e_2$, if

1. $e_1 \to^* v_1$ iff $e_2 \to^* v_2$,
2. $e_1 \to^* E_1[\mathcal{G}_{p_1} x.e_1']$ iff $e_2 \to^* E_2[\mathcal{G}_{p_2} x.e_2']$, where $p_1 \notin \mathsf{sp}(E_1)$ and $p_2 \notin \mathsf{sp}(E_2)$,
3. $e_1 \lightning$ iff $e_2 \lightning$.

We can see that errors and divergence are treated as equivalent, which is standard.

Based on $\sim$, we define *contextual equivalence* as follows.

▶ **Definition 3** (Contextual equivalence)**.** Two closed expressions e_1 and e_2 are contextually equivalent, written, $e_1 \equiv_E e_2$, if for all E such that $\#(E) = \varnothing$, we have $E[e_1] \sim E[e_2]$.

Contextual equivalence can be extended to open terms in a standard way: if $\mathsf{fv}(e_1) \cup \mathsf{fv}(e_2) \subseteq \overrightarrow{x}$, then $e_1 \equiv_E e_2$ if $\lambda\overrightarrow{x}.e_1 \equiv_E \lambda\overrightarrow{x}.e_2$. We test terms using only promptless contexts, because the testing context should not use prompts that are private for the tested expressions. For example, the expressions $\lambda f.f\ p\ q$ and $\lambda f.f\ q\ p$ should be considered equivalent if nothing is known from the outside about p and q. As common in calculi with resource generation [31, 30, 29], testing with evaluation contexts (as in $\equiv_E$) is not the same as testing with all contexts: we have $\mathcal{P}x.x \equiv_E p$, but these terms can be distinguished by

$$\mathsf{let}\ f = \lambda x.\Box\ \mathsf{in}\ \mathsf{if}\ f\ \lambda x.x \overset{?}{=} f\ \lambda x.x\ \mathsf{then}\ \Omega\ \mathsf{else}\ \lambda x.x,$$

In the rest of the article, we show how to characterize $\equiv_E$ with environmental bisimilarities.[2]

[2] If $\equiv_C$ is the contextual equivalence testing with all contexts, then we can prove that $e_1 \equiv_C e_2$ iff $\lambda x.e_1 \equiv_E \lambda x.e_2$, where x is any variable. We therefore obtain a proof method for $\equiv_C$ as well.

3 Environmental Bisimilarity

In this section, we propose a first characterization of $\equiv_E$ using an environmental bisimilarity. We express the bisimilarity in the style of [25], using a so called first-order labeled transition system (LTS), to factorize the soundness proofs of the bisimilarity and its up-to techniques. We start by defining the LTS and its corresponding bisimilarity.

3.1 Labeled Transition System and Bisimilarity

In the original formulation of environmental bisimulation [29], two expressions e_1 and e_2 are compared under some environment $\mathcal{E}$, which represents the knowledge of an external observer about e_1 and e_2. The definition of the bisimulation enforces some conditions on e_1 and e_2 as well as on $\mathcal{E}$. In Madiot et al.'s framework [25, 24], the conditions on e_1, e_2, and $\mathcal{E}$ are expressed using a LTS between *states* of the form (Γ, e_1) (and (Δ, e_2)), where Γ (and Δ) is a finite sequence of values corresponding to the first (and second) projection of the environment $\mathcal{E}$. Note that in (Γ, e_1), e_1 may be a value, and therefore a state can be simply of the form Γ. Transitions from states of the form (Γ, e_1) (where e_1 is not a value) express conditions on e_1, while transitions from states of the form Γ explain how we compare environments. In the rest of the paper we use Γ, Δ to range over finite sequences of values, and we write Γ_i, Δ_i for the i th element of the sequence. We use Σ, Θ to range over states.

Figure 1 presents the LTS $\xrightarrow{\alpha}$, where α ranges over all the labels. We define $\#(\Gamma)$ as $\bigcup_i \#(\Gamma_i)$. The transition $\xrightarrow{\mathbb{E}}$ uses a relation $e \overset{=}{\rightarrow} e'$, defined as follows: if $e \rightarrow e'$, then $e \overset{=}{\rightarrow} e'$, and if e is a normal form, then $e \overset{=}{\rightarrow} e$.[3] To build expressions out of sequences of values, we use different kinds of *multi-hole contexts* defined as follows.

$$\begin{array}{rcll}
\mathbb{C} & ::= & \mathbb{C}_{\mathsf{v}} \mid \mathbb{C}\,\mathbb{C} \mid \mathcal{P}x.\mathbb{C} \mid \#_{\mathbb{C}_{\mathsf{v}}}\mathbb{C} \mid \mathcal{G}_{\mathbb{C}_{\mathsf{v}}}x.\mathbb{C} \mid \mathbb{C}_{\mathsf{v}} \lhd \mathbb{C} & \text{(contexts)} \\
\mathbb{C}_{\mathsf{v}} & ::= & x \mid \lambda x.\mathbb{C} \mid \ulcorner \mathbb{E} \urcorner \mid \Box_i & \text{(value contexts)} \\
\mathbb{E} & ::= & \Box \mid \mathbb{E}\,\mathbb{C} \mid \mathbb{C}_{\mathsf{v}}\,\mathbb{E} \mid \#_{\Box_i}\mathbb{E} & \text{(evaluation contexts)}
\end{array}$$

The holes of a multi-hole context are indexed, except for the special hole $\Box$ of an evaluation context $\mathbb{E}$, which is in evaluation position (that is, filling the other holes of $\mathbb{E}$ with values gives a regular evaluation context E). We write $\mathbb{C}[\Gamma]$ (respectively $\mathbb{C}_{\mathsf{v}}[\Gamma]$ and $\mathbb{E}[\Gamma]$) for the application of a context $\mathbb{C}$ (respectively $\mathbb{C}_{\mathsf{v}}$ and $\mathbb{E}$) to a sequence Γ of values, which consists in replacing $\Box_i$ with Γ_i; we assume that this application produce an expression (or an evaluation context in the case of $\mathbb{E}$), i.e., each hole index in the context is smaller or equal than the size of Γ, and for each $\#_{\Box_i}\mathbb{E}$ construct, Γ_i is a prompt. We write $\mathbb{E}[e, \Gamma]$ for the same operation with evaluation contexts, where we assume that e is put in $\Box$ (note that e may also be a value). Notice that prompts are not part of the syntax of $\mathbb{C}_{\mathsf{v}}$, therefore a multi-hole context does not contain any prompt: if $\mathbb{C}[\Gamma]$, $\mathbb{C}_{\mathsf{v}}[\Gamma]$, or $\mathbb{E}[e, \Gamma]$ contains a prompt, then it comes from Γ or e. Our multi-hole contexts are promptless because $\equiv_E$ also tests with promptless contexts.

We now detail the rules of Figure 1, starting with the transitions that one can find in any call-by-value λ-calculus [25]. An internal action $(\Gamma, e_1) \xrightarrow{\tau} \Sigma$ corresponds to a reduction step, except we ensure that any generated prompt is fresh w.r.t. Γ. The transition $\Gamma \xrightarrow{\lambda, i, \mathbb{C}_{\mathsf{v}}} \Sigma$ signals that Γ_i is a λ-abstraction, which can be tested by passing it an argument built from Γ with the context $\mathbb{C}_{\mathsf{v}}$. The transition $\xrightarrow{\ulcorner . \urcorner, i, \mathbb{C}}$ for testing continuations is built the same way,

[3] The relation $\overset{=}{\rightarrow}$ is not exactly the reflexive closure of $\rightarrow$, since an expression which is not a normal form *has* to reduce.

$$\dfrac{e_1 \to e_2 \qquad \mathsf{fresh}(e_2, e_1, \Gamma)}{(\Gamma, e_1) \xrightarrow{\tau} (\Gamma, e_2)} \qquad \dfrac{\Gamma_i = \lambda x.e}{\Gamma \xrightarrow{\lambda, i, \mathbb{C}_{\mathsf{v}}} (\Gamma, e\{\mathbb{C}_{\mathsf{v}}[\Gamma]/x\})} \qquad \dfrac{}{\Gamma \xrightarrow{\mathsf{v}} \Gamma}$$

$$\dfrac{\Gamma_i = \ulcorner E \urcorner}{\Gamma \xrightarrow{\ulcorner . \urcorner, i, \mathbb{C}} (\Gamma, E[\mathbb{C}[\Gamma]])} \qquad \dfrac{\Gamma_i = p \qquad \Gamma_j = p}{\Gamma \xrightarrow{\#, i, j} \Gamma} \qquad \dfrac{p \notin \#(\Gamma)}{\Gamma \xrightarrow{\#} (\Gamma, p)}$$

$$\dfrac{p \notin \mathsf{sp}(E) \qquad \mathbb{E}[E[\mathcal{G}_p x.e], \Gamma] \xrightarrow{=} e'}{(\Gamma, E[\mathcal{G}_p x.e]) \xrightarrow{\mathbb{E}} (\Gamma, e')}$$

Figure 1 Labeled Transition System for $\lambda_{\mathcal{G}\#}$.

except we use a context $\mathbb{C}$, because any expression can be thrown to a captured context. Finally, the transition $\Gamma \xrightarrow{\mathsf{v}} \Gamma$ means that the state Γ is composed only of values; it does not test anything on Γ, but this transition is useful for the soundness proofs of Section 3.2. When we have $\Gamma \mathrel{\mathcal{R}} (\Delta, e)$ (where $\mathcal{R}$ is, e.g., a bisimulation), then (Δ, e) has to match with $(\Delta, e) \xrightarrow{\tau}^* \xrightarrow{\mathsf{v}} (\Delta, v)$ so that (Δ, v) is related to Γ. We can then continue the proofs with two related sequences of values. Such a transition has been suggested in [24, Remark 5.3.6] to simplify the proofs for a non-deterministic language, like $\lambda_{\mathcal{G}\#}$.

We now explain the rules involving prompts. When comparing two terms generating prompts, one can produce p and the other a different q, so we remember in Γ, Δ that p corresponds to q. But an observer can compare prompts using $\overset{?}{=}$, so p has to be related *only* to q. We check it with $\xrightarrow{\#, i, j}$: if $\Gamma \xrightarrow{\#, i, j} \Gamma$, then Δ has to match, meaning that $\Delta_i = \Delta_j$, and doing so for all j such that $\Gamma_i = \Gamma_j$ ensures that all copies of Γ_i are related only to Δ_i. The transition $\xrightarrow{\#, i, i}$ also signals that Γ_i is a prompt and should be related to a prompt. The other transition involving prompts is $\Gamma \xrightarrow{\#} (\Gamma, p)$, which encodes the possibility for an outside observer to generate fresh prompts to compare terms. If Γ is related to Δ, then Δ has to match by generating a prompt q, and we remember that p is related to q. For this rule to be automatically verified, we define the *prompt checking* rule for a relation $\mathcal{R}$ as follows:

$$\dfrac{\Gamma \mathrel{\mathcal{R}} \Delta \qquad p \notin \#(\Gamma) \qquad q \notin \#(\Delta)}{(\Gamma, p) \mathrel{\mathcal{R}} (\Delta, q)} \ (\#\text{-}\mathsf{check})$$

Henceforth, when we construct a bisimulation $\mathcal{R}$ by giving a set of rules, we always include the ($\#$-check) rule so that the $\xrightarrow{\#}$ transition is always verified.

Finally, the transition $\xrightarrow{\mathbb{E}}$ deals with stuck terms. An expression $E[\mathcal{G}_p x.e]$ is able to reduce if the surrounding context is able to provide a delimiter $\#_p$. However, it is possible only if p is available for the outside, and therefore is in Γ. If $p \notin \mathsf{sp}(\mathbb{E}[\Gamma])$, then $\mathbb{E}[E[\mathcal{G}_p x.e], \Gamma]$ remains stuck, and we have $\mathbb{E}[E[\mathcal{G}_p x.e], \Gamma] \xrightarrow{=} \mathbb{E}[E[\mathcal{G}_p x.e], \Gamma]$. Otherwise, it can reduce and we have $\mathbb{E}[E[\mathcal{G}_p x.e], \Gamma] \xrightarrow{=} e'$, where e' is the result after the capture. The rule for $\xrightarrow{\mathbb{E}}$ may seem demanding, as it tests stuck terms with all contexts $\mathbb{E}$, but up-to techniques will alleviate this issue (see Example 8).

For weak transitions, we define $\Rightarrow$ as $\xrightarrow{\tau}^*$, $\overset{\alpha}{\Rightarrow}$ as $\Rightarrow$ if $\alpha = \tau$ and as $\Rightarrow \xrightarrow{\alpha} \Rightarrow$ otherwise. We define bisimulation and bisimilarity using a more general notion of *progress*. Henceforth, we let $\mathcal{R}$, $\mathcal{S}$ range over relations on states.

▶ **Definition 4.** A relation $\mathcal{R}$ progresses to $\mathcal{S}$, written $\mathcal{R} \rightarrowtail \mathcal{S}$, if $\mathcal{R} \subseteq \mathcal{S}$ and $\Sigma \mathrel{\mathcal{R}} \Theta$ implies

- if $\Sigma \xrightarrow{\alpha} \Sigma'$, then there exists Θ' such that $\Theta \overset{\alpha}{\Rightarrow} \Theta'$ and $\Sigma' \mathrel{\mathcal{S}} \Theta'$;
- the converse of the above condition on Θ.

A *bisimulation* is a relation $\mathcal{R}$ such that $\mathcal{R} \rightarrowtail \mathcal{R}$, and *bisimilarity* $\approx$ is the union of all bisimulations.

3.2 Up-to Techniques, Soundness, and Completeness

Before defining the up-to techniques for $\lambda_{\mathcal{G}\#}$, we briefly recall the main definitions and results we use from [28, 25, 24]; see these works for more details and proofs. We use f, g to range over functions on relations on states. An *up-to technique* is a function f such that $\mathcal{R} \rightarrowtail f(\mathcal{R})$ implies $\mathcal{R} \subseteq \approx$. However, this definition can be difficult to use to prove that a given f is an up-to technique, so we rely on *compatibility* instead. A function f is monotone if $\mathcal{R} \subseteq \mathcal{S}$ implies $f(\mathcal{R}) \subseteq f(\mathcal{S})$. Given a set F of functions, we also write F for the function defined as $\bigcup_{f_i \in F} f_i$ (where $f \cup g$ is defined argument-wise, i.e., $(f \cup g)(R) = f(R) \cup g(R)$). Given a function f, f^ω is defined as $\bigcup_{n \in \mathbb{N}} f^n$.

▶ **Definition 5.** A function f *evolves* to g, written $f \rightsquigarrow g$, if for all $\mathcal{R} \rightarrowtail \mathcal{S}$, we have $f(\mathcal{R}) \rightarrowtail g(\mathcal{S})$. A set F of monotone functions is compatible if for all $f \in F$, $f \rightsquigarrow F^\omega$.

▶ **Lemma 6.** *Let F be a compatible set, and $f \in F$; f is an up-to technique, and $f(\approx) \subseteq \approx$.*

Proving that f is in a compatible set F is easier than proving it is an up-to technique, because we just have to prove that it evolves towards a combination of functions in F. Besides, the second property of Lemma 6 can be used to prove that $\approx$ is a congruence just by showing that bisimulation up to context is compatible.

The first technique we define allows to forget about prompt names; in a bisimulation relating (Γ, e_1) and (Δ, e_2), we remember that $\Gamma_i = p$ is related to $\Delta_i = q$ by their position i, not by their names. Consequently, we can apply different permutations to the two states to rename the prompts without harm, and bisimulation *up to permutations*[4] allows us to do so. Given a relation $\mathcal{R}$, we define $\mathsf{perm}(\mathcal{R})$ as $\Sigma\sigma_1 \mathrel{\mathsf{perm}(\mathcal{R})} \Theta\sigma_2$, assuming $\Sigma \mathrel{\mathcal{R}} \Theta$ and σ_1, σ_2 are any permutations.

We then allow to remove or add values from the states with, respectively, bisimulation *up to weakening* weak and bisimulation *up to strengthening* str, defined as follows

$$\frac{(\overrightarrow{v}, \Gamma, e_1) \mathrel{\mathcal{R}} (\overrightarrow{w}, \Delta, e_2)}{(\Gamma, e_1) \mathrel{\mathsf{weak}(\mathcal{R})} (\Delta, e_2)} \qquad\qquad \frac{(\Gamma, e_1) \mathrel{\mathcal{R}} (\Delta, e_2)}{(\Gamma, \mathbb{C}_{\mathsf{v}}[\Gamma], e_1) \mathrel{\mathsf{str}(\mathcal{R})} (\Delta, \mathbb{C}_{\mathsf{v}}[\Delta], e_2)}$$

Bisimulation up to weakening diminishes the testing power of states, since less values means less arguments to build from for the transitions $\xrightarrow{\lambda, i, \mathbb{C}_{\mathsf{v}}}$, $\xrightarrow{\ulcorner . \urcorner, i, \mathbb{C}}$, and $\xrightarrow{\mathbb{E}}$. This up-to technique is usual for environmental bisimulations, and is called "up to environment" in [29]. In contrast, str adds values to the state, but without affecting the testing power, since the added values are built from the ones already in Γ, Δ.

Finally, we define the well-known bisimulation up to context, which allows to factor out a common context when comparing terms. As usual for environmental bisimulations [29], we define two kinds of bisimulation up to context, depending whether we operate on values or

[4] Madiot defines a bisimulation "up to permutation" in [24] which reorders values in a state. Our bisimulation up to permutations operates on prompts.

any expressions. For values, we can factor out any common context $\mathbb{C}$, but for expressions that are not values, we can factor out only an evaluation context $\mathbb{E}$, since factoring out any context in that case would lead to an unsound up-to technique [24]. We define up to context for values ctx and for any expression ectx as follows:

$$\frac{\Gamma \mathrel{\mathcal{R}} \Delta}{(\Gamma, \mathbb{C}[\Gamma]) \mathrel{\mathsf{ctx}(\mathcal{R})} (\Delta, \mathbb{C}[\Delta])} \qquad \frac{(\Gamma, e_1) \mathrel{\mathcal{R}} (\Delta, e_2)}{(\Gamma, \mathbb{E}[e_1, \Gamma]) \mathrel{\mathsf{ectx}(\mathcal{R})} (\Delta, \mathbb{E}[e_2, \Delta])}$$

▶ **Lemma 7.** *The set* $\{\mathsf{perm}, \mathsf{weak}, \mathsf{str}, \mathsf{ctx}, \mathsf{ectx}\}$ *is compatible.*

The function ectx is particularly helpful in dealing with stuck terms, as we can see below.

▶ **Example 8.** Let $\Sigma \overset{\mathsf{def}}{=} (\Gamma, \mathcal{G}_p x.e_1)$ and $\Theta \overset{\mathsf{def}}{=} (\Delta, \mathcal{G}_q x.e_2)$ (for some e_1, e_2), so that $\Sigma \mathrel{\mathcal{R}} \Theta$. If p and q are not in Γ, Δ, then the two expressions remain stuck, as we have $\Sigma \xrightarrow{\mathbb{E}} (\Gamma, \mathbb{E}[\mathcal{G}_p x.e_1, \Gamma])$ and similarly for Θ. We have directly $(\Gamma, \mathbb{E}[\mathcal{G}_p x.e_1, \Gamma]) \mathrel{\mathsf{ectx}(\mathcal{R})} (\Delta, \mathbb{E}[\mathcal{G}_q x.e_2, \Delta])$. Otherwise, the capture can be triggered with a context $\mathbb{E}$ of the form $\mathbb{E}_1[\#_{\Box_i}\mathbb{E}_2]$, giving $\Sigma \xrightarrow{\mathbb{E}} (\Gamma, \mathbb{E}_1[e_1\{\ulcorner\mathbb{E}_2[\Gamma]\urcorner/x\}, \Gamma])$ and $\Theta \xrightarrow{\mathbb{E}} (\Delta, \mathbb{E}_1[e_2\{\ulcorner\mathbb{E}_2[\Delta]\urcorner/x\}, \Delta])$. Thanks to ectx, we can forget about $\mathbb{E}_1$ which does not play any role, and continue the bisimulation proof by focusing only on $(\Gamma, e_1\{\ulcorner\mathbb{E}_2[\Gamma]\urcorner/x\})$ and $(\Delta, e_2\{\ulcorner\mathbb{E}_2[\Delta]\urcorner/x\})$.

Because bisimulation up to context is compatible, Lemma 6 ensures that $\approx$ is a congruence w.r.t. all contexts for values, and w.r.t. evaluation contexts for all expressions. As a corollary, we can deduce that $\approx$ is sound w.r.t. $\equiv_E$; we can also prove that it is complete w.r.t. $\equiv_E$, leading to the following full characterization result.

▶ **Theorem 9.** $e_1 \equiv_E e_2$ *iff* $(\emptyset, e_1) \approx (\emptyset, e_2)$.

For completeness, we prove that $\{(\Gamma, e_1), (\Delta, e_2) \mid \forall \mathbb{E}, \mathbb{E}[e_1, \Gamma] \sim \mathbb{E}[e_2, \Delta]\}$ is a bisimulation up to permutation; the proof is in [1, Appendix A.1].

3.3 Example

As an example, we show a folklore theorem about delimited control [4], stating that the static operators shift and reset can be simulated by the dynamic operators $\mathsf{control}$ and prompt. In fact, what we prove is a more general and stronger result than the original theorem, since we demonstrate that this simulation still holds when multiple prompts are around.

▶ **Example 10** (Folklore theorem). We encode shift, reset, $\mathsf{control}$, and prompt as follows

$$\begin{array}{lcl@{\qquad}lcl}
\mathsf{shift}_p & \overset{\mathsf{def}}{=} & \lambda f.\mathcal{G}_p k.\#_p f\,(\lambda y.\#_p k \lhd y) & \mathsf{control}_p & \overset{\mathsf{def}}{=} & \lambda f.\mathcal{G}_p k.\#_p f\,(\lambda y.k \lhd y) \\
\mathsf{reset}_p & \overset{\mathsf{def}}{=} & \ulcorner\#_p\Box\urcorner & \mathsf{prompt}_p & \overset{\mathsf{def}}{=} & \ulcorner\#_p\Box\urcorner
\end{array}$$

Let $\mathsf{shift'}_p \overset{\mathsf{def}}{=} \lambda f.\mathsf{control}_p\ (\lambda l.f\ (\lambda z.\mathsf{prompt}_p \lhd l\ z))$; we prove that the pair $(\mathsf{shift}_p, \mathsf{reset}_p)$ (encoded as $\lambda f.f\,\mathsf{shift}_p\mathsf{reset}_p$) is bisimilar to $(\mathsf{shift'}_p, \mathsf{prompt}_p)$ (encoded as $\lambda f.f\,\mathsf{shift'}_p\mathsf{prompt}_p$).

Proof. We iteratively build a relation $\mathcal{R}$ closed under (#-check) such that $\mathcal{R}$ is a bisimulation up to context, starting with $(p, \mathsf{shift}_p) \mathrel{\mathcal{R}} (p, \mathsf{shift'}_p)$. The transition $\xrightarrow{\#,1,1}$ is easy to check. For $\xrightarrow{\lambda,2,\mathbb{C}_\mathsf{v}}$, we obtain states of the form $(p, \mathsf{shift}_p, e_1)$, $(p, \mathsf{shift'}_p, e_2)$ that we add to $\mathcal{R}$, where e_1 and e_2 are the terms below

$$\frac{\Gamma \mathrel{\mathcal{R}} \Delta}{(\Gamma, \mathcal{G}_p k.\#_p\mathbb{C}_\mathsf{v}[\Gamma]\ (\lambda y.\#_p k \lhd y)) \mathrel{\mathcal{R}} (\Delta, \mathcal{G}_p k.\#_p(\lambda l.\mathbb{C}_\mathsf{v}[\Delta]\ (\lambda z.\mathsf{prompt}_p \lhd l\ z))\ (\lambda y.k \lhd y))}$$

We use an inductive, more general rule, because we want $\xrightarrow{\lambda,2,\mathbb{C}_\mathsf{v}}$ to be still verified after we extend (p, shift_p) and $(p, \mathsf{shift'}_p)$. The terms e_1 and e_2 are stuck, so we test them with $\xrightarrow{\mathbb{E}}$. If $\mathbb{E}$ does not trigger the capture, we obtain $\mathbb{E}[e_1, \Gamma]$ and $\mathbb{E}[e_2, \Delta]$, and we can use ectx to conclude. Otherwise, $\mathbb{E} = \mathbb{E}'[\#_{\Box_1}\mathbb{E}'']$ (where $\#_{\Box_1}$ does not surround $\Box$ in $\mathbb{E}''$), and we get

$$\mathbb{E}'[\#_p\mathbb{C}_\mathsf{v}[\Gamma]\ (\lambda y.\#_p\ulcorner\mathbb{E}''[\Gamma]\urcorner \lhd y), \Gamma] \text{ and } \mathbb{E}'[\#_p\mathbb{C}_\mathsf{v}[\Delta]\ (\lambda z.\mathsf{prompt}_p \lhd (\lambda y.\ulcorner\mathbb{E}''[\Delta]\urcorner \lhd y)\ z), \Delta]$$

We want to use ctx to remove the common context $\mathbb{E}'[\#_{\Box_1}\mathbb{C}_\mathsf{v}\ \Box_i]$, which means that we have to add the following states in the definition of $\mathcal{R}$ (again, inductively):

$$\frac{\Gamma\ \mathcal{R}\ \Delta}{(\Gamma, \lambda y.\#_p\ulcorner\mathbb{E}''[\Gamma]\urcorner \lhd y)\ \mathcal{R}\ (\Delta, \lambda z.\mathsf{prompt}_p \lhd (\lambda y.\ulcorner\mathbb{E}''[\Delta]\urcorner \lhd y)\ z)}$$

Testing these functions with $\xrightarrow{\lambda,i,\mathbb{C}_\mathsf{v}}$ gives on both sides states where $\#_{\Box_1}\mathbb{E}''[\mathbb{C}_\mathsf{v}]$ can be removed with ctx. Because $(\varnothing, \lambda f.f\ \mathsf{shift}_p\ \mathsf{reset}_p)\ \mathsf{weak}(\mathsf{ctx}(\mathcal{R}))\ (\varnothing, \lambda f.f\ \mathsf{shift'}_p\ \mathsf{prompt}_p)$, it is enough to conclude. Indeed, $\mathcal{R}$ is a bisimulation up to context, so $\mathcal{R} \subseteq \approx$, which implies $\mathsf{weak}(\mathsf{ctx}(\mathcal{R})) \subseteq \mathsf{weak}(\mathsf{ctx}(\approx))$ (because weak and ctx are monotone), and $\mathsf{weak}(\mathsf{ctx}(\approx)) \subseteq \approx$ (by Lemma 6). Note that this reasoning works for any combination of monotone up-to techniques and any bisimulation (up-to). ◀

What makes the proof of Example 10 quite simple is that we relate (p, shift_p) and $(p, \mathsf{shift'}_p)$, meaning that p can be used by an outside observer. But the control operators $(\mathsf{shift}_p, \mathsf{reset}_p)$ and $(\mathsf{shift'}_p, \mathsf{prompt}_p)$ should be the only terms available for the outside, since p is used only to implement them. If we try to prove equivalent these two pairs directly, i.e., while keeping p private, then testing reset_p and prompt_p with $\xrightarrow{\ulcorner.\urcorner,2,\mathbb{C}}$ requires a cumbersome analysis of the behaviors of $\#_p\mathbb{C}[\Gamma]$ and $\#_p\mathbb{C}[\Delta]$. In the next section, we define a new kind of bisimilarity with a powerful up-to technique to make such proofs more tractable.

4 The $\star$-Bisimilarity

4.1 Motivation

Testing continuations. In Section 3, a continuation $\Gamma_i = \ulcorner E\urcorner$ is tested with $\Gamma \xrightarrow{\ulcorner.\urcorner,i,\mathbb{C}} (\Gamma, E[\mathbb{C}[\Gamma]])$. We then have to study the behavior of $E[\mathbb{C}[\Gamma]]$, which depends primarily on how $\mathbb{C}[\Gamma]$ reduces; e.g., if $\mathbb{C}[\Gamma]$ diverges, then E does not play any role. Consequently, the transition $\xrightarrow{\ulcorner.\urcorner,i,\mathbb{C}}$ does not really test the continuation directly, since we have to reduce $\mathbb{C}[\Gamma]$ first. To really exhibit the behavior of the continuation, we change the transition so that it uses a value context instead of a general one. We then have $\Gamma \xrightarrow{\ulcorner.\urcorner,i,\mathbb{C}_\mathsf{v}} (\Gamma, E[\mathbb{C}_\mathsf{v}[\Gamma]])$, and the behavior of the term we obtain depends primarily on E. However, this is not equivalent to testing with $\mathbb{C}$, since $\mathbb{C}[\Gamma]$ may interact in other ways with E if $\mathbb{C}[\Gamma]$ is a stuck term. If E is of the form $E'[\#_pE'']$, and p is in Γ, then $\mathbb{C}$ may capture E'', since p can be used to build an expression of the form $\mathcal{G}_px.e$. To take into account this possibility, we introduce a new transition $\Gamma \xrightarrow{\ulcorner.\urcorner,i,j} (\Gamma, \ulcorner E'\urcorner, \ulcorner E''\urcorner)$, which decomposes $\Gamma_i = E'[\#_pE'']$ into $\ulcorner E'\urcorner$ and $\ulcorner E''\urcorner$, provided $\Gamma_j = p$. The stuck term $\mathbb{C}[\Gamma]$ may also capture E entirely, as part of a bigger context of the form $\mathbb{E}_1[E[\mathbb{E}_2]]$. To take this into account, we introduce a way to build such contexts using captured continuations. This is also useful to make bisimulation up to context more expressive, as we explain in the next paragraph.

A more expressive bisimulation up to context. As we already pointed out in [7], bisimulation up to context is not very helpful in the presence of control operators. For example, suppose we prove the β_Ω axiom of [18], i.e., $(\lambda x.E[x])\ e$ is equivalent to $E[e]$ if $x \notin \mathsf{fv}(E)$ and $\mathsf{sp}(E) = \emptyset$. If e is a stuck term $\mathcal{G}_p y.e_1$, we have to compare $e_1\{\ulcorner E_1[(\lambda x.E[x])\ \square]\urcorner/y\}$ and $e_1\{\ulcorner E_1[E]\urcorner/y\}$ for some E_1. If e_1 is of the form $y \triangleleft (y \triangleleft e_2)$, then we get respectively $E_1[(\lambda x.E[x])\ E_1[(\lambda x.E[x])\ e_2]]$ and $E_1[E[E_1[E[e_2]]]]$. We can see that the two resulting expressions have the same shape, and yet we can only remove the outermost occurrence of E_1 with ectx. The problem is that bisimulation up to context can factor out only a *common* context. We want an up-to technique able to identify *related* contexts, i.e., contexts built out of related continuations. To do so, we modify the multi-hole contexts to include a construct $\star_i[\mathbb{C}]$ with a special hole $\star_i$, which can be filled only with $\ulcorner E \urcorner$ to produce a context $E[\mathbb{C}]$. As a result, if $\Gamma = (\ulcorner(\lambda x.E[x])\ \square\urcorner)$ and $\Delta = (\ulcorner E\urcorner)$, then $E_1[(\lambda x.E[x])\ E_1[(\lambda x.E[x])\ \square]]$ and $E_1[E[E_1[E[\square]]]]$ can be written $\mathbb{E}[\Gamma]$, $\mathbb{E}[\Delta]$ with $\mathbb{E} = E_1[\star_1[E_1[\star_1[\square]]]]$. We can then focus only on testing Γ and Δ.

However, such a bisimulation up to related context would be unsound if not restricted in some way. Indeed, let $\ulcorner E_1\urcorner$, $\ulcorner E_2\urcorner$ be any continuations, and let $\Gamma = (\ulcorner E_1\urcorner)$, $\Delta = (\ulcorner E_2\urcorner)$. Then the transitions $\Gamma \xrightarrow{\ulcorner.\urcorner,1,\mathbb{C}_\mathsf{v}} (\Gamma, E_1[\mathbb{C}_\mathsf{v}[\Gamma]])$ and $\Delta \xrightarrow{\ulcorner.\urcorner,1,\mathbb{C}_\mathsf{v}} (\Delta, E_2[\mathbb{C}_\mathsf{v}[\Delta]])$ produce states of the form $(\Gamma, \mathbb{C}[\Gamma])$, $(\Delta, \mathbb{C}[\Delta])$ with $\mathbb{C} = \star_1[\mathbb{C}_\mathsf{v}]$. If bisimulation up to related context was sound in that case, it would mean that $\ulcorner E_1\urcorner$ and $\ulcorner E_2\urcorner$ would be bisimilar for all E_1 and E_2, which, of course, is wrong.[5] To prevent this, we distinguish *passive* transitions (such as $\xrightarrow{\ulcorner.\urcorner,i,\mathbb{C}_\mathsf{v}}$) from the other ones (called *active*), so that only selected up-to techniques (referred to as *strong*) can be used after a passive transition. In contrast, any up-to technique (including this new bisimulation up to related context) can be used after an active transition. To formalize this idea, we have to extend Madiot et al.'s framework to allow such distinctions between transitions and between up-to techniques.

4.2 Labeled Transition System and Bisimilarity

First, we explain how we alter the LTS of Section 3.1 to implement the changes we sketched in Section 4.1. We extend the grammar of multi-hole contexts $\mathbb{C}$ (resp. $\mathbb{E}$) by adding the production $\star_i[\mathbb{C}]$ (resp. $\star_i[\mathbb{E}]$), where the hole $\star_i$ can be filled only with a continuation (the grammar of value contexts $\mathbb{C}_\mathsf{v}$ is unchanged). When we write $(\star_i[\mathbb{C}])[\Gamma]$, we assume Γ_i is a continuation $\ulcorner E\urcorner$, and the result of the operation is $E[\mathbb{C}[\Gamma]]$ (and similarly for $\mathbb{E}$).

We also change the way we deal with captured contexts, by using the following rules:

$$\frac{\Gamma_i = \ulcorner E\urcorner}{\Gamma \xrightarrow{\ulcorner.\urcorner,i,\mathbb{C}_\mathsf{v}} (\Gamma, E[\mathbb{C}_\mathsf{v}[\Gamma]])} \qquad \frac{\Gamma_i = \ulcorner E_1[\#_p E_2]\urcorner \qquad \Gamma_j = p \qquad p \notin \mathsf{sp}(E_2)}{\Gamma \xrightarrow{\ulcorner.\urcorner,i,j} (\Gamma, \ulcorner E_1\urcorner, \ulcorner E_2\urcorner)}$$

The transition $\xrightarrow{\ulcorner.\urcorner,i,\mathbb{C}_\mathsf{v}}$ is the same as in Section 3, except that it tests with an argument built with a value context $\mathbb{C}_\mathsf{v}$ instead of a regular context $\mathbb{C}$. We also introduce the transition $\xrightarrow{\ulcorner.\urcorner,i,j}$, which decomposes a captured context $\ulcorner E_1[\#_p E_2]\urcorner$ into sub-contexts $\ulcorner E_1\urcorner$, $\ulcorner E_2\urcorner$, provided that p is in Γ. This transition is necessary to take into account the possibility for an external observer to capture a part of a context, scenario which can no longer be tested with $\xrightarrow{\ulcorner.\urcorner,i,\mathbb{C}_\mathsf{v}}$, as explained in Section 4.1, and as illustrated with the next example.

[5] The problem is similar if we test continuations using contexts $\mathbb{C}$ (as in Section 3) instead of $\mathbb{C}_\mathsf{v}$.

▶ **Example 11.** Let $\Gamma = (p, \ulcorner \#_p \Box \urcorner)$, $\Delta = (q, \ulcorner \Box \urcorner)$; then $\Gamma \xrightarrow{\ulcorner . \urcorner, 2, \mathbb{C}_{\mathsf{v}}} (\Gamma, \#_p \mathbb{C}_{\mathsf{v}}[\Gamma]) \xrightarrow{\tau} (\Gamma, \mathbb{C}_{\mathsf{v}}[\Gamma])$ and $\Delta \xrightarrow{\ulcorner . \urcorner, 2, \mathbb{C}_{\mathsf{v}}} (\Delta, \mathbb{C}_{\mathsf{v}}[\Delta])$. Without the $\xrightarrow{\ulcorner . \urcorner, i, j}$ transition, Γ and Δ would be bisimilar, which would not be sound (they are distinguished by the context $\Box_2 \lhd \mathcal{G}_{\Box_1} x.\Omega$).

The other rules are not modified, but their meaning is still affected by the change in the contexts grammars: the transitions $\xrightarrow{\lambda, i, \mathbb{C}_{\mathsf{v}}}$ and $\xrightarrow{\mathbb{E}}$ can now test with more arguments. This is a consequence of the fact that an observer can build a bigger continuation from a captured context. For instance, if $\Gamma = (p, \ulcorner E \urcorner, \lambda x.x \lhd v)$, then with the LTS of Section 3, we have $\Gamma \xrightarrow{\ulcorner . \urcorner, 2, \mathbb{E}_1[\mathcal{G}_{\Box_1} x.x]} \xrightarrow{\#_{\Box_1} \mathbb{E}_2} \xrightarrow{\lambda, 3, \Box_4} (\Gamma, \ulcorner \mathbb{E}_1[E[\mathbb{E}_2[\Gamma]], \Gamma] \urcorner, \ulcorner \mathbb{E}_1[E[\mathbb{E}_2[\Gamma]], \Gamma] \urcorner \lhd v)$. In the new LTS, the first transition is no longer possible, but we can still test the λ-abstraction with the same argument using $\Gamma \xrightarrow{\lambda, 3, \mathbb{E}_1[\star_2[\mathbb{E}_2]]} (\Gamma, \ulcorner \mathbb{E}_1[E[\mathbb{E}_2[\Gamma]], \Gamma] \urcorner \lhd v)$.

As explained in Section 4.1, we want to prevent the use of some up-to techniques (like the bisimulation up to related context we introduce in Section 4.3) after some transitions, especially $\xrightarrow{\ulcorner . \urcorner, i, \mathbb{C}_{\mathsf{v}}}$. To do so, we distinguish the *passive* transitions $\xrightarrow{\ulcorner . \urcorner, i, \mathbb{C}_{\mathsf{v}}}$, $\xrightarrow{\mathsf{v}}$ from the other ones, called *active*. In a LTS, a visible action $\xrightarrow{\alpha}$ (where $\alpha \neq \tau$) usually corresponds to an interaction with an external observer. The transition $\xrightarrow{\mathsf{v}}$ does not fit that principle; similarly, $\xrightarrow{\ulcorner . \urcorner, i, \mathbb{C}_{\mathsf{v}}}$ does not correspond exactly to an observer interacting with a continuation, since we throw a value, and not any expression. In contrast, $\xrightarrow{\lambda, i, \mathbb{C}_{\mathsf{v}}}$ corresponds to function application, $\xrightarrow{\mathbb{E}}$ to context capture, $\xrightarrow{\ulcorner . \urcorner, i, j}$ to continuation decomposition, and $\xrightarrow{\#, i, j}$ to testing prompts equality. This is how we roughly distinguish the former transitions as passive, and the latter ones as active. We then change the definition of progress, to allow a relation $\mathcal{R}$ to progress towards different relations after passive and active transitions.

▶ **Definition 12.** A relation $\mathcal{R}$ diacritically progresses to $\mathcal{S}, \mathcal{T}$ written $\mathcal{R} \rightarrowtail\!\!\!\twoheadrightarrow \mathcal{S}, \mathcal{T}$, if $\mathcal{R} \subseteq \mathcal{S}$, $\mathcal{R} \subseteq \mathcal{T}$, and $\Sigma \mathrel{\mathcal{R}} \Theta$ implies that

- if $\Sigma \xrightarrow{\alpha} \Sigma'$ and $\xrightarrow{\alpha}$ is passive, then there exists Θ' such that $\Theta \overset{\alpha}{\Rightarrow} \Theta'$ and $\Sigma' \mathrel{\mathcal{S}} \Theta'$;
- if $\Sigma \xrightarrow{\alpha} \Sigma'$ and $\xrightarrow{\alpha}$ is active, then there exists Θ' such that $\Theta \overset{\alpha}{\Rightarrow} \Theta'$ and $\Sigma' \mathrel{\mathcal{T}} \Theta'$;
- the converse of the above conditions on Θ.

A $\star$-bisimulation is a relation $\mathcal{R}$ such that $\mathcal{R} \rightarrowtail\!\!\!\twoheadrightarrow \mathcal{R}, \mathcal{R}$, and $\star$-bisimilarity $\overset{\star}{\approx}$ is the union of all $\star$-bisimulations.

Note that with the same LTS, $\rightarrowtail$ and $\rightarrowtail\!\!\!\twoheadrightarrow$ entail the same notions of bisimulation and bisimilarity (but we use a different LTS in this section).

4.3 Up-to Techniques, Soundness, and Completeness

We now discriminate up-to techniques, so that regular up-to techniques cannot be used after passive transitions, while *strong* ones can. An up-to technique (resp. *strong* up-to technique) is a function f such that $\mathcal{R} \rightarrowtail\!\!\!\twoheadrightarrow \mathcal{R}, f(\mathcal{R})$ (resp. $\mathcal{R} \rightarrowtail\!\!\!\twoheadrightarrow f(\mathcal{R}), f(\mathcal{R})$) implies $\mathcal{R} \subseteq \overset{\star}{\approx}$. We also adapt the notions of evolution and compatibility.

▶ **Definition 13.** A function f evolves to g, h, written $f \rightsquigarrow g, h$, if for all $\mathcal{R} \rightarrowtail\!\!\!\twoheadrightarrow \mathcal{R}, \mathcal{T}$, we have $f(\mathcal{R}) \rightarrowtail\!\!\!\twoheadrightarrow g(\mathcal{R}), h(\mathcal{T})$.

A function f *strongly* evolves to g, h, written $f \rightsquigarrow_{\mathsf{s}} g, h$, if for all $\mathcal{R} \rightarrowtail\!\!\!\twoheadrightarrow \mathcal{S}, \mathcal{T}$, we have $f(\mathcal{R}) \rightarrowtail\!\!\!\twoheadrightarrow g(\mathcal{S}), h(\mathcal{T})$.

Strong evolution is very general, as it uses any relation $\mathcal{R}$, while regular evolution is more restricted, as it relies on relations $\mathcal{R}$ such that $\mathcal{R} \rightarrowtail\!\!\!\twoheadrightarrow \mathcal{R}, \mathcal{T}$. But the definition of *diacritical* *compatibility* below still allows to use any combinations of strong up-to techniques after a passive transition, even for functions which are not themselves strong. In contrast, regular functions can only be used once after a passive transition of an other regular function.

▶ **Definition 14.** A set F of monotone functions is *diacritically compatible* if there exists $S \subseteq F$ such that

- for all $f \in S$, we have $f \rightsquigarrow_{\mathsf{s}} S^\omega, F^\omega$;
- for all $f \in F$, we have $f \rightsquigarrow S^\omega \circ F \circ S^\omega, F^\omega$.

If S_1 and S_2 are subsets of F which verify the conditions of the definition, then $S_1 \cup S_2$ also does, so there exists the largest subset of F which satisfies the conditions, written $\mathsf{strong}(F)$. This (possibly empty) subset of F corresponds to the strong up-to techniques of F.

▶ **Lemma 15.** *Let F be a compatible set.*

- *If $\mathcal{R} \rightarrowtail\!\!\!\twoheadrightarrow \mathsf{strong}(F)^\omega(\mathcal{R}), F^\omega(\mathcal{R})$, then $F^\omega(\mathcal{R})$ is a bisimulation.*
- *If $f \in F$, then f is an up-to technique. If $f \in \mathsf{strong}(F)$, then f is a strong up-to technique.*
- *For all $f \in F$, we have $f(\approx) \subseteq \approx$.*

We now use this framework to define up-to techniques for the $\star$-bisimulation. The definitions of perm and weak are unchanged. We define bisimulation up to related contexts for values rctx and for any term rectx as follows:

$$\frac{\Gamma \mathrel{\mathcal{R}} \Delta}{(\Gamma, \overrightarrow{\mathbb{C}_{\mathsf{v}}}[\Gamma], \mathbb{C}[\Gamma]) \mathrel{\mathsf{rctx}(\mathcal{R})} (\Delta, \overrightarrow{\mathbb{C}_{\mathsf{v}}}[\Delta], \mathbb{C}[\Delta])} \qquad \frac{(\Gamma, e_1) \mathrel{\mathcal{R}} (\Delta, e_2)}{(\Gamma, \overrightarrow{\mathbb{C}_{\mathsf{v}}}[\Gamma], \mathbb{E}[e_1, \Gamma]) \mathrel{\mathsf{rectx}(\mathcal{R})} (\Delta, \overrightarrow{\mathbb{C}_{\mathsf{v}}}[\Delta], \mathbb{E}[e_2, \Delta])}$$

The definitions look similar to the ones of ctx and ectx, but the grammar of multi-hole contexts now include $\star_i$. Besides, we inline strengthening in the definitions of rctx and rectx, allowing Γ, Δ to be extended. This is necessary because, e.g., str and rectx cannot be composed after a passive transition (they are both not strong), so rectx have to include str directly. Note that the behavior of str can be recovered from rectx by taking $\mathbb{E} = \Box$.

▶ **Lemma 16.** $F \overset{\mathsf{def}}{=} \{\mathsf{perm}, \mathsf{weak}, \mathsf{rctx}, \mathsf{rectx}\}$ *is compatible, with* $\mathsf{strong}(F) = \{\mathsf{perm}, \mathsf{weak}\}$.

As a result, these functions are up-to techniques, and weak and perm can be used after a passive transition. Because of the last item of Lemma 15, $\overset{\star}{\approx}$ is also a congruence w.r.t. evaluation contexts, which means that $\overset{\star}{\approx}$ is sound w.r.t. $\equiv_E$. We can also prove it is complete the same way as for Theorem 9, leading again to full characterization.

▶ **Theorem 17.** $e_1 \equiv_E e_2$ *iff* $(\emptyset, e_1) \overset{\star}{\approx} (\emptyset, e_2)$.

4.4 Examples

We illustrate the use of $\overset{\star}{\approx}$, rctx, and rectx with two examples that would be much harder to prove with the techniques of Section 3.

▶ **Example 18** (β_Ω **axiom).** We prove $(\lambda x.E[x])\, e \overset{\star}{\approx} E[e]$ if $x \notin \mathsf{fv}(E)$ and $\mathsf{sp}(E) = \emptyset$. Define $\mathcal{R}$ starting with $(\ulcorner(\lambda x.E[x])\,\Box\urcorner) \mathrel{\mathcal{R}} (\ulcorner E \urcorner)$, and closing it under the (#-check) and the following rule:

$$\frac{\Gamma \mathrel{\mathcal{R}} \Delta}{(\Gamma, (\lambda x.E[x])\, \mathbb{C}_{\mathsf{v}}[\Gamma]) \mathrel{\mathcal{R}} (\Delta, E[\mathbb{C}_{\mathsf{v}}[\Delta]])}$$

Then $(\emptyset, (\lambda x.E[x])\ e)\ \mathsf{weak}(\mathsf{rctx}(\mathcal{R}))\ (\emptyset, E[e])$ and $\mathcal{R}$ is a bisimulation up to context, since the sequence $\Gamma \xrightarrow{\ulcorner.\urcorner,1,\mathbb{C}_{\mathsf{v}}} (\Gamma, (\lambda x.E[x])\ \mathbb{C}_{\mathsf{v}}[\Gamma]) \xrightarrow{\tau} (\Gamma, E[\mathbb{C}_{\mathsf{v}}[\Gamma]])$ fits $\Delta \xrightarrow{\ulcorner.\urcorner,1,\mathbb{C}_{\mathsf{v}}} (\Delta, E[\mathbb{C}_{\mathsf{v}}[\Delta]]) \overset{\tau}{\Rightarrow} (\Delta, E[\mathbb{C}_{\mathsf{v}}[\Delta]])$, where the final states are in rctx. Notice we use rctx after $\xrightarrow{\tau}$, and not after the passive $\xrightarrow{\ulcorner.\urcorner,1,\mathbb{C}_{\mathsf{v}}}$ transition.

▶ **Example 19** (Exceptions). A possible way of extending a calculus with exception handling is to add a construct $\mathsf{try}_r\ e\ \mathsf{with}\ v$, which evaluates e with a function raising an exception stored under the variable r. When e calls the function in r with some argument v', even inside another try block, then the computation of e is aborted and replaced by vv'. We can implement this behavior directly in $\lambda_{\mathcal{G}\#}$; more precisely, we write $\mathsf{try}_r\ e\ \mathsf{with}\ v$ as $\mathsf{handle}\ (\lambda r.e)\ v$, where handle is a function expressed in the calculus. One possible implementation of handle in $\lambda_{\mathcal{G}\#}$ is very natural and heavily relies on fresh-prompt generation:

$$\mathsf{handle} \overset{\mathsf{def}}{=} \lambda f.\lambda h.\mathcal{P}x.\#_x f\ (\lambda z.\mathcal{G}_{x}\_.h\ z)$$

The idea is to raise an exception by aborting the current continuation up to the corresponding prompt. The same function can be implemented using any comparable-resource generation and only one prompt p:

$$\begin{aligned} \mathsf{handle}_p &\overset{\mathsf{def}}{=} \lambda f.\lambda h.\mathcal{P}x.(\#_p \mathsf{let}\ r = f\ \mathsf{raise}_{p,x}\ \mathsf{in}\ \lambda\_.\lambda\_.r)\ x\ h \\ \mathsf{raise}_{p,x} &\overset{\mathsf{def}}{=} \mathsf{fix}\ r(z).\mathcal{G}_p\_.\lambda y.\lambda h.\mathsf{if}\ x \overset{?}{=} y\ \mathsf{then}\ h\ z\ \mathsf{else}\ r\ z \end{aligned}$$

Here the idea is to keep a freshly generated name x and a handler function h with the prompt corresponding to each call of handle_p. The exception-raising function $\mathsf{raise}_{p,x}$ iteratively aborts the current delimited continuation up to the nearest call of handle_p and checks the name stored there in order to find the corresponding handler. Note that this implementation also uses prompt generation, since it is the only comparable resource that can be dynamically generated in $\lambda_{\mathcal{G}\#}$, but the implementation can be easily translated to, e.g., a calculus with single-prompted delimited-control operators and first-order store.

Proof. We prove that both versions of handle are bisimilar. As in Example 10 we iteratively build a relation $\mathcal{R}$ closed under the ($\#$-check) rule, so that $\mathcal{R}$ is a bisimulation up to context. We start with $(\mathsf{handle})\ \mathcal{R}\ (\mathsf{handle}_p)$; to match the $\xrightarrow{\lambda,1,\mathbb{C}_{\mathsf{v}}}$ transition, we extend $\mathcal{R}$ as follows:

$$\frac{\Gamma\ \mathcal{R}\ \Delta}{(\Gamma, \lambda h.\mathcal{P}x.\#_x\mathbb{C}_{\mathsf{v}}[\Gamma]\ (\lambda z.\mathcal{G}_x\_.h\ z))\ \mathcal{R}\ (\Delta, \lambda h.\mathcal{P}x.(\#_p\mathsf{let}\ r = \mathbb{C}_{\mathsf{v}}[\Delta]\ \mathsf{raise}_{p,x}\ \mathsf{in}\ \lambda\_.\lambda\_.r)\ x\ h)}$$

We obtain two functions which are in turn tested with $\xrightarrow{\lambda,n+1,\mathbb{C}'_{\mathsf{v}}}$, and we obtain the states

$$(\Gamma, \#_{p_1}\mathbb{C}_{\mathsf{v}}[\Gamma]\ (\lambda z.\mathcal{G}_{p_1}\_.\mathbb{C}'_{\mathsf{v}}[\Gamma]\ z))\ \text{and}\ (\Delta, (\#_p\mathsf{let}\ r = \mathbb{C}_{\mathsf{v}}[\Delta]\ \mathsf{raise}_{p,p_2}\ \mathsf{in}\ \lambda\_.\lambda\_.r)\ p_2\ \mathbb{C}'_{\mathsf{v}}[\Delta]).$$

Instead of adding them to $\mathcal{R}$ directly, we decompose them into corresponding parts using up to context (with $\mathbb{C} = \star_{n+1}[\mathbb{C}_{\mathsf{v}}\ \Box_{n+2}]$), and we add these subterms to $\mathcal{R}$:

$$\frac{\Gamma\ \mathcal{R}\ \Delta \qquad p_1 \notin \#(\Gamma) \qquad p_2 \notin \#(\Delta)}{(\Gamma, \ulcorner\#_{p_1}\Box\urcorner, \lambda z.\mathcal{G}_{p_1}\_.\mathbb{C}'_{\mathsf{v}}[\Gamma]\ z)\ \mathcal{R}\ (\Delta, \ulcorner(\#_p\mathsf{let}\ r = \Box\ \mathsf{in}\ \lambda\_.\lambda\_.r)\ p_2\ \mathbb{C}'_{\mathsf{v}}[\Delta]\urcorner, \mathsf{raise}_{p,p_2})}\ (*)$$

Testing the two captured contexts with $\xrightarrow{\ulcorner.\urcorner,n+1,\mathbb{C}''_{\mathsf{v}}}$ is easy, because they both evaluate to the thrown value. We now consider $\lambda z.\mathcal{G}_{p_1}\_.\mathbb{C}'_{\mathsf{v}}[\Gamma]\ z$ and raise_{p,p_2}; after the transition $\xrightarrow{\lambda,n+2,\mathbb{C}_{\mathsf{v}}}$ we get the two control stuck terms

$$\mathcal{G}_{p_1}\_.\mathbb{C}'_{\mathsf{v}}[\Gamma]\ \mathbb{C}_{\mathsf{v}}[\Gamma] \quad \text{and} \quad \mathcal{G}_p\_.\lambda y.\lambda h.\mathsf{if}\ p_2 \overset{?}{=} y\ \mathsf{then}\ h\ \mathbb{C}_{\mathsf{v}}[\Delta]\ \mathsf{else}\ \mathsf{raise}_{p,p_2}\ \mathbb{C}_{\mathsf{v}}[\Delta]$$

Adding such terms to the relation will not be enough. The first one can be unstuck only using the corresponding context $\ulcorner \#_{p_1} \Box \urcorner$, but the second one can be unstuck using any context added by rule $(*)$, even for a different p_2. In such a case, it will consume a part of the context and evaluate to itself. To be more general we add the following rule:

$$\frac{\Gamma \mathrel{\mathcal{R}} \Delta \qquad \mathbb{E}[\mathcal{G}_{p_1}\text{-}.\mathbb{C}'_{\mathsf{v}}[\Gamma]\ \mathbb{C}_{\mathsf{v}}[\Gamma], \Gamma] \text{ is control-stuck}}{(\Gamma, \mathbb{E}[\mathcal{G}_{p_1}\text{-}.\mathbb{C}'_{\mathsf{v}}[\Gamma]\ \mathbb{C}_{\mathsf{v}}[\Gamma], \Gamma]) \mathrel{\mathcal{R}} (\Delta, \mathcal{G}_{p}\text{-}.\lambda y.\lambda h.\mathsf{if}\ p_2 \overset{?}{=} y \ \mathsf{then}\ h\ \mathbb{C}_{\mathsf{v}}[\Delta]\ \mathsf{else}\ \mathsf{raise}_{p,p_2}\ \mathbb{C}_{\mathsf{v}}[\Delta])}$$

The newly introduced stuck terms are tested with $\xrightarrow{\mathbb{E}'}$; if $\mathbb{E}'$ does not have $\star_i$ surrounding $\Box$, they are still stuck, and we can use up to evaluation context to conclude. Assume $\mathbb{E}' = \mathbb{E}_1[\star_i[\mathbb{E}_2]]$ where $\mathbb{E}_2$ has not $\star_j$ around $\Box$. If i points to the evaluation context added by $(*)$ for the same p_2, then they both evaluate to terms of the same shape, so we use up to context with $\mathbb{C} = \mathbb{E}_1[\mathbb{C}'_{\mathsf{v}}\ \mathbb{C}_{\mathsf{v}}]$. Otherwise, we know the second program compares two different prompts, so it evaluates to $\mathbb{E}_1[\mathcal{G}_{p}\text{-}.\lambda y.\lambda h.\mathsf{if}\ p_2 \overset{?}{=} y\ \mathsf{then}\ h\ \mathbb{C}_{\mathsf{v}}[\Delta]\ \mathsf{else}\ \mathsf{raise}_{p,p_2}\ \mathbb{C}_{\mathsf{v}}[\Delta], \Delta]$ and we use rectx with the last rule. ◀

5 Related Work and Conclusion

Related work. In previous works [5, 6, 7], we defined several bisimilarities for a calculus (called $\lambda_{\mathcal{S}}$) with the (less expressive) delimited-control operators shift and reset. The bisimilarity of Section 3 and the corresponding up-to techniques are close to the ones of [7, Section 3], except that in [7], we do not compare stuck terms using *all* evaluation contexts. However, there is no equivalent of bisimulation up to related contexts in [7], which makes the proof of the β_Ω axiom very difficult in that paper. The proof in Example 18 is as easy as the proof of the β_Ω axiom in [6], but the bisimilarity of [6] is not complete, unlike $\overset{\star}{\approx}$. As a matter of fact, following the developments of Section 4, we believe it is possible to define environmental bisimulations up to related contexts for the $\lambda_{\mathcal{S}}$-calculus.

Environmental bisimilarity has been defined in several calculi with dynamic resource generation, like stores and references [23, 22, 30], information hiding constructs [31, 32], or name creation [3, 26]. In these works, an expression is paired with its generated resources, and behavioral equivalences are defined on these pairs. Our approach is different since we do not carry sets of generated prompts when manipulating expressions (e.g., in the semantic rules of Section 2); instead, we rely on side-conditions and permutations to avoid collisions between prompts. This is possible because all we need to know is if a prompt is known to an outside observer or not, and the correspondences between the public prompts of two related expressions; this can be done through the environment of the bisimilarity. This approach cannot be adapted to more complex generated resources, which are represented by a mapping (e.g., for stores or existential types), but we believe it can be used for name creation in π-calculus [26].

A line of work on program equivalence for which relating evaluation contexts is crucial, as in our work, are logical relations based on the notion of biorthogonality [27]. In particular, this concept has been successfully used to develop techniques for establishing program equivalence in ML-like languages with call/cc [10], and for proving the coherence of control-effect subtyping [8]. Hur et al. combine logical relations and behavioral equivalences in the definition of *parametric bisimulation* [16], where terms are reduced to normal forms that are then decomposed into subterms related by logical relations. This framework has been extended to abortive control in [17], where *stuttering* is used to allow terms not to reduce for a finite amount of time when comparing them in a bisimulation proof. This is reminiscent of our distinction between active and passive transitions, as passive transitions can be seen as "not reducing", but there is still some testing involved in these transitions. Besides, the concern is different, since the active/passive distinction prevents the use of up-to techniques, while stuttering has been proposed to improve plain parametric bisimulations.

Conclusion and future work. We have developed a behavioral theory for Dybvig et al.'s calculus of multi-prompted delimited control, where the enabling technology for proving program equivalence are environmental bisimulations, presented in Madiot's style. The obtained results generalize our previous work in that they account for multiple prompts and local visibility of dynamically generated prompts. Moreover, the results of Section 4 considerably enhance reasoning about captured contexts by treating them as first-class objects at the level of bisimulation proofs (thanks to the construct $\star_i$) and not only at the level of terms. The resulting notion of bisimulation up to related contexts improves on the existing bisimulation up to context in presence of control operators, as we can see when comparing Example 18 to the proof of the same result in [7]. We believe bisimulation up to related contexts could be useful for constructs akin to control operators, like passivation in π-calculus [26]. The soundness of this up-to technique has been proved in an extension of Madiot's framework; we plan to investigate further this extension, to see how useful it could be in defining up-to techniques for other languages. Finally, it may be possible to apply the tools developed in this paper to [20], where a single-prompted calculus is translated into a multi-prompted one, but no operational correspondence is given to guarantee the soundness of the translation.

Acknowledgements. We would like to thank Jean-Marie Madiot for the insightful discussions about his work, and Małgorzata Biernacka, Klara Zielińska, and the anonymous reviewers for the helpful comments on the presentation of this work.

References

1 Andrés Aristizábal, Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. Environmental bisimulations for delimited-control operators with dynamic prompt generation. Research Report RR-8905, Inria, Nancy, France, April 2016. Available at `http://hal.inria.fr/hal-01305137`.

2 Vincent Balat, Roberto Di Cosmo, and Marcelo P. Fiore. Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums. In Xavier Leroy, editor, *Proceedings of the Thirty-First Annual ACM Symposium on Principles of Programming Languages*, pages 64–76, Venice, Italy, January 2004. ACM Press.

3 Nick Benton and Vasileios Koutavas. A mechanized bisimulation for the nu-calculus. Technical Report MSR-TR-2008-129, Microsoft Research, September 2008.

4 Dariusz Biernacki and Olivier Danvy. A simple proof of a folklore theorem about delimited control. *Journal of Functional Programming*, 16(3):269–280, 2006.

5 Dariusz Biernacki and Sergueï Lenglet. Applicative bisimulations for delimited-control operators. In Lars Birkedal, editor, *Foundations of Software Science and Computation Structures, 15th International Conference (FOSSACS'12)*, volume 7213 of *Lecture Notes in Computer Science*, pages 119–134, Tallinn, Estonia, March 2012. Springer.

6 Dariusz Biernacki and Sergueï Lenglet. Normal form bisimulations for delimited-control operators. In Tom Schrijvers and Peter Thiemann, editors, *Functional and Logic Programming, 13th International Symposium (FLOPS'12)*, volume 7294 of *Lecture Notes in Computer Science*, pages 47–61, Kobe, Japan, May 2012. Springer.

7 Dariusz Biernacki and Sergueï Lenglet. Environmental bisimulations for delimited-control operators. In Chung-chieh Shan, editor, *Proceedings of the 11th Asian Symposium on Programming Languages and Systems (APLAS'13)*, volume 8301 of *Lecture Notes in Computer Science*, pages 333–348, Melbourne, VIC, Australia, December 2013. Springer.

8 Dariusz Biernacki and Piotr Polesiuk. Logical relations for coherence of effect subtyping. In Thorsten Altenkirch, editor, *Typed Lambda Calculi and Applications, 13th International Conference, TLCA 2015*, volume 38 of *Leibniz International Proceedings in Informatics*, pages 107–122, Warsaw, Poland, July 2015. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

9 Olivier Danvy and Andrzej Filinski. Abstracting control. In Mitchell Wand, editor, *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pages 151–160, Nice, France, June 1990. ACM Press.

10 Derek Dreyer, Georg Neis, and Lars Birkedal. The impact of higher-order state and control effects on local relational reasoning. *Journal of Functional Programming*, 22(4-5):477–528, 2012.

11 R. Kent Dybvig, Simon Peyton-Jones, and Amr Sabry. A monadic framework for delimited continuations. *Journal of Functional Programming*, 17(6):687–730, 2007.

12 Matthias Felleisen. The theory and practice of first-class prompts. In Jeanne Ferrante and Peter Mager, editors, *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 180–190, San Diego, California, January 1988. ACM Press.

13 Andrzej Filinski. Representing monads. In Hans-J. Boehm, editor, *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Programming Languages*, pages 446–457, Portland, Oregon, January 1994. ACM Press.

14 Matthew Flatt, Gang Yu, Robert Bruce Findler, and Matthias Felleisen. Adding delimited and composable control to a production programming environment. In Norman Ramsey, editor, *Proceedings of the 2007 ACM SIGPLAN International Conference on Functional Programming (ICFP'07)*, pages 165–176, Freiburg, Germany, September 2007. ACM Press.

15 Carl Gunter, Didier Rémy, and Jon G. Riecke. A generalization of exceptions and control in ML-like languages. In Simon Peyton Jones, editor, *Proceedings of the Seventh ACM Conference on Functional Programming and Computer Architecture*, pages 12–23, La Jolla, California, June 1995. ACM Press.

16 Chung-Kil Hur, Derek Dreyer, Georg Neis, and Viktor Vafeiadis. The marriage of bisimulations and Kripke logical relations. In John Field and Michael Hicks, editors, *Proceedings of the Thirty-Ninth Annual ACM Symposium on Principles of Programming Languages*, pages 59–72, Philadelphia, PA, USA, January 2012. ACM Press.

17 Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. A logical step forward in parametric bisimulations. Technical Report MPI-SWS-2014-003, Max Planck Institute for Software Systems (MPI-SWS), Saarbrücken, Germany, January 2014.

18 Yukiyoshi Kameyama and Masahito Hasegawa. A sound and complete axiomatization of delimited continuations. In Olin Shivers, editor, *Proceedings of the 2003 ACM SIGPLAN International Conference on Functional Programming (ICFP'03)*, pages 177–188, Uppsala, Sweden, August 2003. ACM Press.

19 Oleg Kiselyov. Delimited control in OCaml, abstractly and concretely: System description. In Matthias Blume and German Vidal, editors, *Functional and Logic Programming, 10th International Symposium, FLOPS 2010*, volume 6009 of *Lecture Notes in Computer Science*, pages 304–320, Sendai, Japan, April 2010. Springer.

20 Ikuo Kobori, Yukiyoshi Kameyama, and Oleg Kiselyov. ATM without tears: prompt-passing style transformation for typed delimited-control operators. In Olivier Danvy, editor, *2015 Workshop on Continuations: Pre-proceedings*, London, UK, April 2015.

21 Vasileios Koutavas, Paul Blain Levy, and Eijiro Sumii. From applicative to environmental bisimulation. In Michael Mislove and Joël Ouaknine, editors, *Proceedings of the 27th Annual Conference on Mathematical Foundations of Programming Semantics (MFPS XXVII)*, volume 276 of *Electronic Notes in Theoretical Computer Science*, pages 215–235, Pittsburgh, PA, USA, May 2011.

22 Vasileios Koutavas and Mitchell Wand. Bisimulations for untyped imperative objects. In Peter Sestoft, editor, *15th European Symposium on Programming, ESOP 2006*, volume 3924 of *Lecture Notes in Computer Science*, pages 146–161, Vienna, Austria, March 2006. Springer.

23 Vasileios Koutavas and Mitchell Wand. Small bisimulations for reasoning about higher-order imperative programs. In J. Gregory Morrisett and Simon L. Peyton Jones, editors, *Proceedings of the 33rd Annual ACM Symposium on Principles of Programming Languages*, pages 141–152, Charleston, SC, USA, January 2006. ACM Press.

24 Jean-Marie Madiot. *Higher-Order Languages: Dualities and Bisimulation Enhancements*. PhD thesis, Université de Lyon and Università di Bologna, 2015.

25 Jean-Marie Madiot, Damien Pous, and Davide Sangiorgi. Bisimulations up-to: Beyond first-order transition systems. In Paolo Baldan and Daniele Gorla, editors, *25th International Conference on Concurrency Theory*, volume 8704 of *Lecture Notes in Computer Science*, pages 93–108, Rome, Italy, September 2014. Springer.

26 Adrien Piérard and Eijiro Sumii. A higher-order distributed calculus with name creation. In *Proceedings of the 27th IEEE Symposium on Logic in Computer Science (LICS 2012)*, pages 531–540, Dubrovnik, Croatia, June 2012. IEEE Computer Society Press.

27 Andrew Pitts and Ian Stark. Operational reasoning for functions with local state. In Andrew Gordon and Andrew Pitts, editors, *Higher Order Operational Techniques in Semantics*, pages 227–273. Publications of the Newton Institute, Cambridge University Press, 1998.

28 Damien Pous and Davide Sangiorgi. Enhancements of the bisimulation proof method. In Davide Sangiorgi and Jan Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*, chapter 6, pages 233–289. Cambridge University Press, 2011.

29 Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. *ACM Transactions on Programming Languages and Systems*, 33(1):1–69, January 2011.

30 Eijiro Sumii. A complete characterization of observational equivalence in polymorphic lambda-calculus with general references. In Erich Grädel and Reinhard Kahle, editors, *Computer Science Logic, CSL'09*, volume 5771 of *Lecture Notes in Computer Science*, pages 455–469, Coimbra, Portugal, September 2009. Springer.

31 Eijiro Sumii and Benjamin C. Pierce. A bisimulation for dynamic sealing. *Theoretical Computer Science*, 375(1-3):169–192, 2007.

32 Eijiro Sumii and Benjamin C. Pierce. A bisimulation for type abstraction and recursion. *Journal of the ACM*, 54(5), 2007.

Complexity of Acyclic Term Graph Rewriting*

Martin Avanzini[1] and Georg Moser[2]

1 Università di Bologna, Italy & INRIA Sophia Antipolis, France
martin.avanzini@uibk.ac.at

2 Universität Innsbruck, Austria
georg.moser@uibk.ac.at

Abstract

Term rewriting has been used as a formal model to reason about the complexity of logic, functional, and imperative programs. In contrast to term rewriting, *term graph rewriting* permits sharing of common sub-expressions, and consequently is able to capture more closely reasonable implementations of rule based languages. However, the automated complexity analysis of term graph rewriting has received little to no attention.

With this work, we provide first steps towards overcoming this situation. We present adaptions of two prominent complexity techniques from term rewriting, viz, the *interpretation method* and *dependency tuples*. Our adaptions are non-trivial, in the sense that they can observe not only term but also graph structures, i.e. take sharing into account. In turn, the developed methods allow us to more precisely estimate the runtime complexity of programs where sharing of sub-expressions is essential.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.3.2 Semantics of Programming Languages, F.4.1 Process Management, F.4.2 Storage Management

Keywords and phrases Program Analysis, Graph Rewriting, Complexity Analysis

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.10

1 Introduction

In recent years automated complexity analysis of term rewriting (see [20] for an early overview) has received increased attention, which has manifested itself in a number of significant breakthroughs. For brevity, we only mention recent work on direct methods [21, 3, 29], on worst case lower bounds [15], and on certification [6]. Furthermore the liveliness of the designated complexity competition clearly showcases the various activities in this area. [1] These activities have also triggered applications outside of rewriting. In particular term rewriting has been very successfully used as a formal model to reason about the complexity of logic, functional, and imperative programs, cf. [16, 28, 11, 2].

In contrast to term rewriting, *term graph rewriting* permits sharing of common sub-expressions, and consequently is able to capture more closely reasonable implementations of rule based languages. However, the automated complexity analysis of term graph rewriting has received little to no attention. This is somewhat surprising. On the one hand, term graph rewriting is typically motivated as *implementation* of term rewriting. Hence effectivity of the implementation should have been an issue. On the other hand, (term) graph rewriting is the rule in any kind of implementation of functional programs [27]. Consider for instance the

[*] This work was partially supported by FWF (Austrian Science Fund) projects J 3563, P 25781-N15, and by AFRL/DARPA.

[1] http://www.termination-portal.org/wiki/Termination_Competition.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 10; pp. 10:1–10:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

```
power x 0 = 1
power x n = y * y * (if n `mod` 2 == 0 then 1 else x)
  where y = power x (n `div` 2)
```

Figure 1 Fast Exponentiation in `Haskell`.

`Haskell` program depicted in Figure 1. The example showcases the necessity to implement non-strict evaluation via graph reduction. Indeed, if we assign unit cost to arithmetic operations as usual, then it is easy to see that graph reduction in general requires time linear in the bit-length of n. In contrast, when implemented naively, e.g. as a *term rewrite system* (*TRS* for short) under an outermost reduction strategy, in each step the recursive call is duplicated. Conclusively the runtime complexity becomes exponential in the setting of TRSs. Moreover, transformations from imperative languages to term rewriting would profit from a more direct representation of the heap as a graph, rather than a tree. Thus complexity analysis of term graph rewriting, automated if possible, should have significant impact.

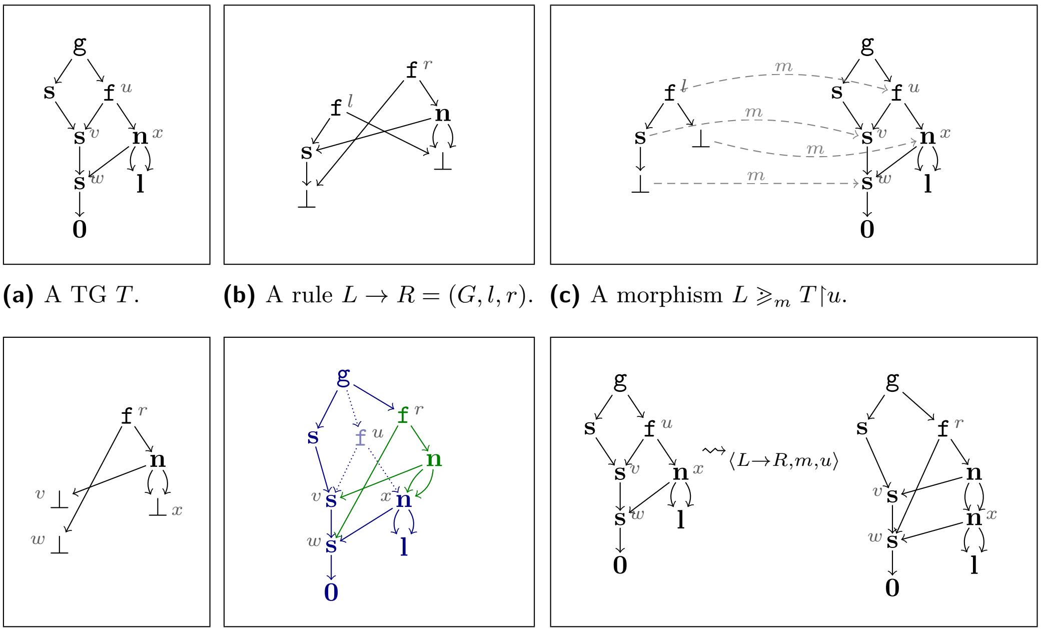
In this paper, we provide first steps towards overcoming this situation. We present adaptions of two prominent complexity techniques from term rewriting, viz, the *interpretation method* and *dependency tuples*. We summarise the contributions of this paper.

- We clarify and fix the notion of *runtime complexity* in the context of term graph rewriting (see Section 3). This is a non-trivial task, as we have to take care of succinct representations of start terms.
- We provide a novel *interpretation method* for term graph rewrite systems (Theorem 12). This method is obtained by a careful adaption of the notion of well-founded monotone algebra to term graphs.
- We show that in the context of sharing, existing restrictions of the dependency tuple approach to innermost evaluation can be overcome and establish a *dependency pair method* for term graph rewrite systems (Theorem 21).

The results above transfer two core techniques of the dependency pair framework to the complexity analysis of term graph rewrite systems. Great care has been taken to establish the correctness of these techniques in the more general setting of relative graph rewriting. Consequently, these methods are readily applicable in the *complexity pair framework* from [4], suited to term graph rewrite systems. Although not presented here, this in turn paves the way to transfer with relative ease a variety of complexity techniques applicable in the dependency pair setting from term to graph rewriting, notably, the *usable rules criterion*, *predecessor estimation*, *dependency graph decomposition* and various *simplification techniques*, see [4].

Our adaptions are non-trivial, in the sense that they can observe not only term but also graph structures, i.e. take sharing into account. In turn, the developed methods allow us to more precisely estimate the runtime complexity of programs where sharing of sub-expressions is essential.

This paper is structured as follows. In the next two sections we cover basics and clarify the notion of term graph rewriting employed. In Section 4, we present our interpretation method of term graph rewriting and in Section 5, we adapt the dependency tuple technique to this context. In Section 6 we discuss related work. Finally, in Section 7 we conclude and mention future work.



(a) A TG T. **(b)** A rule $L \to R = (G, l, r)$. **(c)** A morphism $L \geqslant_m T{\restriction}u$.

(d) The TG $m(R)$. **(e)** The TG $T\langle m(R)\rangle_u$. **(f)** A reduction step on T with rule $L \to R$.

Figure 2 Step-by-step construction of a graph rewrite step.

2 Preliminaries

We shortly recap basic definitions and notions. With $\mathbb{N}$ we denote the set of natural numbers $\{0, 1, 2, \ldots\}$. For a set S, we denote by S^* the set of finite sequences $[s_1, s_2, \ldots, s_n]$ over elements $s_i \in S$. A partial function f from A to B is denoted by $f : A \nrightarrow B$. Its *domain* is $\mathsf{dom}(f) := \{a \in A \mid f(a) \text{ is defined}\}$. For two partial functions $f, g : A \nrightarrow \mathbb{N}$ and $a \in A$ we define $f(a) \leqslant_{\mathsf{k}} g(a)$ if either $f(a)$ or $g(a)$ is undefined, or $f(a)$ and $g(a)$ are defined and $f(a) \leqslant g(a)$ holds.

Let $\to \subseteq S \times S$ be a binary relation. We denote by $\to^+$ the transitive and by $\to^*$ the transitive and reflexive closure of $\to$. We say that $\to$ is *well-founded* or *terminating*, if there is no infinite sequence $s_0 \to s_1 \to \ldots$. It is *finitely branching*, if the set $\{t \mid s \to t\}$ is finite for each $s \in S$. For two binary relations $\to_A$ and $\to_B$, the relation of $\to_A$ *relative* to $\to_B$ is defined by $\to_A/\to_B := \to_B^* \cdot \to_A \cdot \to_B^*$. The *derivation height* $dh_\to : S \nrightarrow \mathbb{N}$ with respect to $\to$ over S is defined by $dh_\to(s) := \max\{\ell \in \mathbb{N} \mid s = s_0 \to s_1 \to \cdots \to s_\ell\}$. Note that $dh_\to$ is total whenever $\to$ is terminating and finitely branching. Let $(S_i)_{i \in \mathbb{N}}$ denote a countably infinite family of monotonically increasing subsets of S, i.e. $S_i \subseteq S_{i+1}$ for all $i \in \mathbb{N}$, whose limit is S. For brevity, we denote the family $(S_i)_{i \in \mathbb{N}}$ by S. We define $\mathrm{rc}_\to^S : \mathbb{N} \nrightarrow \mathbb{N}$ by $\mathrm{rc}_\to^S(n) := \max\{dh_\to(s) \mid s \in S_n\}$.

3 Term Graph Rewriting and All That

We introduce central concepts and notions of term graph rewriting, see [7] for an overview.

Term Graphs

Let $\mathcal{F}$ denote a signature, i.e. a finite set of function symbols. Each $\mathtt{f} \in \mathcal{F}$ is associated with a natural number $\mathsf{ar}(\mathtt{f})$, its *arity*. Moreover, we suppose a partitioning of the signature $\mathcal{F}$

into *defined symbols* $\mathcal{D}$ and *constructors* $\mathcal{C}$. Defined symbols take over the role of *operations*, whereas constructors are used to build *values*. Throughout the following, the signature $\mathcal{F}$ and its separation into $\mathcal{D}$ and $\mathcal{C}$ is kept fixed. A *term graph* (*TG* for short) T over the signature $\mathcal{F}$ is a *directed acyclic* graph whose internal nodes are labeled by symbols in $\mathcal{F}$, and where outgoing edges are ordered. Formally, T is a triple $(N_T, \mathsf{suc}_T, \mathsf{lab}_T)$ consisting of *nodes* N_T, a *partial successors function* $\mathsf{suc}_T : N_T \rightharpoonup N_T^*$ from nodes to sequences of nodes, and a *partial labeling function* $\mathsf{lab}_T : N_T \rightharpoonup \mathcal{F}$. Unlabeled nodes take the role of variables in terms, and are collected in $V_T \subseteq N_T$. We require that TGs are *compatible with their labeling*, in the sense that for each node $u \in N_T$, if $\mathsf{lab}_T(u) = \mathtt{f}$ then $\mathsf{suc}_T(u) = [u_1, \dots, u_{\mathsf{ar}(\mathtt{f})}]$ and otherwise, $\mathsf{suc}_T(u)$ is undefined. In the former case, we also write $T(u) = \mathtt{f}(u_1, \dots, u_{\mathsf{ar}(\mathtt{f})})$. We define the *successor relation* $\rightharpoonup_T$ on nodes in T such that $u \rightharpoonup_T v$ holds if $\mathsf{suc}_T(u)$ is defined and v occurs in $\mathsf{suc}_T(u)$. If v occurs at the i^{th} position we also write $u \xrightharpoonup{i}_T v$. Throughout the following, we consider only *acyclic* TGs, that is, we demand that $\rightharpoonup_T$ is acyclic. If not mentioned otherwise, we also suppose that TGs are *rooted*, i.e. T contains a unique node $\mathsf{rt}(T) \in N_T$, the *root*, from which all nodes $v \in N_T$ are *reachable*: $\mathsf{rt}(T) \rightharpoonup_T^* v$. See Figure 2(a) for an example of a TG. Here, we depict nodes directly by their label, possibly annotating node identities. *Unfolding* a term graph T from its root results in a finite term over the signature $\mathcal{F}$ and variables V_T, and we sometimes use this term as a linear representation for the TG T. For instance, the TG depicted in Figure 2(a) unfolds to $\mathtt{g}(\mathbf{s}(\mathbf{s}(\mathbf{s}(\mathbf{0}))), \mathtt{f}(\mathbf{s}(\mathbf{s}(\mathbf{0})), \mathbf{n}(\mathbf{s}(\mathbf{0}), \mathbf{l}, \mathbf{l})))$.

The *size* $|T|$ of the TG T refers to the cardinality of N_T. The TG T is called *ground* if $V_T = \varnothing$. For a subset $\mathcal{G} \subseteq \mathcal{F}$ of function symbols, we collect in $N_T^{\mathcal{G}} \subseteq N_T$ all nodes u with $\mathsf{lab}_T(u) \in \mathcal{G}$. We call a node $u \in N_T$ *below* and *above*, respectively, of a node $v \in N_T$ in accordance to the topological ordering induced by $\rightharpoonup_T^*$. Two nodes are called *parallel* in T, if they are mutually unreachable. For instance, in Figure 2(a), the nodes v and x are parallel. The TG T is a *tree* if every node in T is reachable by precisely one path from its root $\mathsf{rt}(T)$. Thus, trees do not exhibit sharing.

We denote by $T{\upharpoonright}u$ the *sub-graph* of T *rooted* at node $u \in N_T$. With $T[v \leftarrow u]$ we denote the graph obtained by *redirecting* all edges pointing to u to point to the node v. That is, $T[v \leftarrow u]$ denotes the TG with nodes $N_T \cup \{v\}$, labeling $\mathsf{lab}_{T[v\leftarrow u]} := \mathsf{lab}_T$ and the successor function $\mathsf{suc}_{T[v\leftarrow u]}$ is defined such that (i) $w \xrightharpoonup{i}_{T[v\leftarrow u]} v$ holds for each edge $w \xrightharpoonup{i}_T u$ in T, and (ii) $w \xrightharpoonup{i}_{T[v\leftarrow u]} w'$ holds whenever $w \xrightharpoonup{i}_T w'$ for $w' \neq u$. Note that if $v \notin N_T$, then v is considered a variable node in $T[v \leftarrow u]$. The notion is naturally extended to sequences, i.e. $T[v_1, \dots, v_n \leftarrow u_1, \dots, u_n]$ denotes the TG obtained by redirecting edges pointing to u_i to v_i for all $1 \leqslant i \leqslant n$. Here, we assume that for all $1 \leqslant i, j \leqslant n$, the node u_i is distinct from u_j (if $j \neq i$) and v_j. We denote by $S \cup T$ the *union* of two TGs S and T. To avoid ambiguities, we require that if $u \in N_S \cap N_T$ then $\mathsf{lab}_S(u)$ or $\mathsf{lab}_T(u)$ is undefined. We define

$$\mathsf{suc}_{S\cup T}(u) := \begin{cases} \mathsf{suc}_S(u) & \text{if } u \in N_S \text{ and } \mathsf{lab}_S(u) \in \mathcal{F}, \\ \mathsf{suc}_T(u) & \text{if } u \in N_T \text{ and } \mathsf{lab}_T(u) \in \mathcal{F}, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Similarly, we define the labeling $\mathsf{lab}_{S\cup T}$. We write $T\langle S\rangle_u$ to denote the replacement of the subgraph in T at node u by S:

$$T\langle S\rangle_u := \begin{cases} S & \text{if } u = \mathsf{rt}(T), \\ \big(T[\mathsf{rt}(S) \leftarrow u] \cup S\big){\upharpoonright}\,\mathsf{rt}(T) & \text{otherwise.} \end{cases} .$$

For two rooted TGs $S = (N_S, \mathsf{suc}_S, \mathsf{lab}_S)$ and $T = (N_T, \mathsf{suc}_T, \mathsf{lab}_T)$, a mapping $m : N_S \to N_T$ is called *morphic* in $u \in N_S$ if (i) $\mathsf{lab}_S(u) = \mathsf{lab}_T(m(u))$ whenever $\mathsf{lab}_S(u)$ is defined, and

(ii) $u \overset{i}{\rightharpoonup}_S v$ implies $m(u) \overset{i}{\rightharpoonup}_T m(v)$ for all appropriate i. A (*rooted*) *homomorphism* from S to T is a mapping $m : N_S \to N_T$ that (i) maps the root of S to the root of T and that (ii) is morphic in all nodes $u \in N_S^{\mathcal{F}}$. We write $S \geqslant_m T$ to indicate that m is a homomorphism from S to T. We denote by $\underline{m}$ its extension outside of its domain, such that $\underline{m}(u) = u$ whenever u is not in the domain of m. Two TGs are *isomorphic*, in notation $S \cong T$, if there exist two morphisms m_1, m_2 with $S \geqslant_{m_1} T$ and $T \geqslant_{m_2} S$, or equivalently, if $S \geqslant_m T$ holds for a bijective morphism m. Observe that two isomorphic TGs are equal up to renaming of nodes.

Term Graph Rewriting

A *graph rewrite rule* over the signature $\mathcal{F}$ is a triple (G, l, r) where G is a TG over $\mathcal{F}$ and $l, r \in N_G$ are two distinguished nodes, denoting the root of the left- and right-hand side, respectively. We require that all nodes in G are reachable from l or r. This way, a graph rewrite rule can be denoted by $L \to R$, for the *left-hand side* $L := G{\restriction}l$ and *right-hand side* $R := G{\restriction}r$. Furthermore, we demand that the root l of the left-hand side L is labeled by a defined symbol, and that all variable nodes in R occur also in L. The label of l is called the *defined symbol* of $L \to R$. The maximal sub-graph of G contained in both L and R is called the *interface* of the rule $L \to R$. See Figure 2(b) that depicts a graph rewrite rule defining $\mathtt{f}$, with left-hand side $\mathtt{f}(\mathbf{s}(x), y)$ and right-hand side $\mathtt{f}(x, \mathbf{n}(\mathbf{s}(x), y, y))$, for x and y denoting the two variable nodes. The interface of this rule consists of the two variable nodes, as well as the node labeled by $\mathbf{s}$. Isomorphisms are naturally extended to graph rewrite rules. An isomorphic copy of a rule $L \to R$ is also called a *renaming*. If R lies within L, we also say that $L \to R$ is a *collapsing* rule. A *graph rewrite system* (*GRS* for short) $\mathcal{G}$ over $\mathcal{F}$ is a finite set of graph rewrite rules over $\mathcal{F}$.

Let S be a ground TG and let $L \to R$ be a graph rewrite rule with nodes disjoint from those in S. Application of $L \to R$ on S involves the identification of a *redex*, i.e. homomorphic copy of L in S, replacing this copy with a copy of R, retaining interface nodes, and finally *garbage collecting* nodes that became inaccessible. All of this is formalised as follows. We say that the graph rewrite rule $L \to R$ *matches* the TG S at node u if $L \geqslant_m S{\restriction}u$ holds. The triple $\langle L \to R, m, u\rangle$ is called a *redex* in S, and the node u of S the *redex node*, compare Figure 2(c). With $m(R)$ we denote the *instantiation* of the right-hand side of R by the matching morphism $L \geqslant_m S{\restriction}u$. This is done by redirecting edges according to the morphism m, from R to S, and then removing inaccessible nodes. Formally, let $u_1, \dots, u_n$ denote all nodes of the interface in $L \to R = (G, l, r)$. Then

$$m(R) := R[m(u_1), \dots, m(u_n) \leftarrow u_1, \dots, u_n]{\restriction}\underline{m}(r) \; .$$

Observe that $\underline{m}(r) \neq r$ only when R is collapsing, i.e. when r is an interface node u_i. In this case, $m(R)$ consists of the single variable node $m(r)$. Compare Figure 2(d) which depicts $m(R)$ with respect to the right-hand side R of the rule from Figure 2(b) and the matching morphism drawn in Figure 2(c).

We define $S \rightsquigarrow_{\langle L \to R, m, u\rangle} T$, if $\langle L \to R, m, u\rangle$ is a redex in S and $T := S\langle m(R)\rangle_u$, and call $S \rightsquigarrow_{\langle L \to R, m, u\rangle} T$ a *pre-reduction step*. We write $S \rightsquigarrow_{L \to R} T$ when the precise redex is unimportant. Since nodes of S and R are disjoint, $S\langle m(R)\rangle_u$ is well-defined and acyclic. Observe that by construction, the right-hand side R is embedded in T at node $\mathsf{rt}(R)$, more precise, $R \geqslant_{\underline{m}} T{\restriction}\mathsf{rt}(R)$ holds. Compare Figures 2(e) and 2(f).

For a rule $L \to R = (G, l, r)$, we define their *difference set* $\Delta(L \to R) := (N_R \setminus N_L) \cup \{l\}$ if $l \in N_R$, and $\Delta(L \to R) := N_R \setminus N_L$ otherwise. For symbols $\mathcal{D} \subseteq \mathcal{F}$ we denote by $\Delta^{\mathcal{D}}(L \to R)$ the restriction of $\Delta(L \to R)$ to nodes labeled by symbols from $\mathcal{D}$. The following technical result will be useful later.

▶ **Proposition 1.** *If $S \rightsquigarrow_{\langle L\to R,m,u\rangle} T$ then $N_T^{\mathcal{D}} \subseteq (N_S^{\mathcal{D}} \setminus \{u\}) \cup \{\underline{m}(v) \mid v \in \Delta^{\mathcal{D}}(L \to R)\}$.*

Proof. Observe that $\underline{m}$ is the identity function on $N_R \setminus N_L$, whereas $\underline{m}(l) = u$ for l the root of the left-hand side L. Further, $N_T^{\mathcal{D}} \subseteq N_S^{\mathcal{D}} \cup \Delta^{\mathcal{D}}(L \to R) \setminus \{l\}$ where moreover, $u \notin N_T^{\mathcal{D}}$ whenever l does not occur in the right-hand side R. From this, the claim follows by case analysis on $l \in N_R$. ◀

To every TG T, we can identify an isomorphic, *canonical* TG $\mathcal{C}(T)$ where nodes are sets of *positions*, i.e. finite sequences of integers [26]. In particular, if two TG S and T are isomorphic, then $\mathcal{C}(S) = \mathcal{C}(T)$. To avoid reasoning modulo TG isomorphisms below, we define the *graph rewrite relation* $\to_{\mathcal{G}}$ induced by the GRS $\mathcal{G}$ over canonical TGs. We define $S \to_{L\to R} T$ if $S \rightsquigarrow_{\langle L'\to R',m,u\rangle} T'$ and $\mathcal{C}(T') = T$ holds for a renaming $L' \to R'$ of $L \to R$, some morphism m and node u. Renaming ensures that nodes in $L \to R$ are fresh with respect to S. Notice that independent of the particular renaming $L \to R$, the reduct T is unique. Finally, we define $S \to_{\mathcal{G}} T$ if $S \to_{L\to R} T$ holds for some rule $L \to R \in \mathcal{G}$.

Runtime Complexity

To measure the complexity of an operation $\mathtt{f} \in \mathcal{D}$ we adopt a *unitary cost model*, where each reduction step has unit cost. To this end, we look at calls to $\mathtt{f}$ when supplied with values. In short, we restrict our attention to reductions starting from *basic TGs* $\Diamond(\mathcal{D},\mathcal{C})$, i.e. TGs whose root is labeled by a defined symbol $\mathcal{D}$, and whose arguments are values formed from $\mathcal{C}$. Similarly, the set $\triangle(\mathcal{D},\mathcal{C}) \subseteq \Diamond(\mathcal{D},\mathcal{C})$ of *basic trees* is defined. We abbreviate $\Diamond(\mathcal{D},\mathcal{C})$ and $\triangle(\mathcal{D},\mathcal{C})$ by $\Diamond$ and $\triangle$, respectively.

Let S be a set of TGs, parameterised in their size. For simplicity, we assume that S denotes the limit of a family of TGs $(S_i)_{i=\mathbb{N}}$, where $S_i \subseteq S$ collects all TGs in $\bigcup_{i\in\mathbb{N}} S$ of size up to i. As above we denote the family simply by S. Recall that $\mathrm{rc}^{S}_{\to_{\mathcal{G}}}(n) = \max\{dh_{\to_{\mathcal{G}}}(s) \mid s \in S_n\}$, cf. page 3. The *runtime complexity (function)* of $\mathcal{G}$ with respect to *starting graphs* S is defined as $\mathrm{rc}^{S}_{\mathcal{G}}(n) := \mathrm{rc}^{S}_{\to_{\mathcal{G}}}(n)$.

Furthermore, we set $dh_{\mathcal{G}}(T) := dh_{\to_{\mathcal{G}}}(T)$. Of particular interest will be the runtime complexity $\mathrm{rc}^{\Diamond}_{\mathcal{G}}(n)$ of $\mathcal{G}$ on basic TGs, and the runtime complexity $\mathrm{rc}^{\triangle}_{\mathcal{G}}(n)$ of $\mathcal{G}$ on basic trees.

Relative Term Graph Rewriting

Rather than focusing solely on a GRS $\mathcal{G}$, we also consider the graph rewrite relation of a GRS $\mathcal{G}$ relative to a GRS $\mathcal{H}$, in notation $\mathcal{G}/\mathcal{H}$. This way, we can seamlessly adopt the combination framework for complexity analysis underlying our tool TCT [4] and the certifier CeTA [6]. The relation $\to_{\mathcal{G}}/\to_{\mathcal{H}}$ is abbreviated by $\to_{\mathcal{G}/\mathcal{H}}$. Note that $\to_{\mathcal{G}/\mathcal{H}}$ specialises to $\to_{\mathcal{G}}$ for the case $\mathcal{H} = \varnothing$. Similar to above, we set $dh_{\mathcal{G}/\mathcal{H}}(T) := dh_{\to_{\mathcal{G}/\mathcal{H}}}(T)$ and $\mathrm{rc}^{S}_{\mathcal{G}/\mathcal{H}}(n) := \mathrm{rc}^{S}_{\to_{\mathcal{G}/\mathcal{H}}}(n)$ for a set of TGs S. Note that the derivation height of a TG T with respect to $\to_{\mathcal{G}/\mathcal{H}}$, if defined, corresponds to the number of applications of rules from $\mathcal{G}$ in a $\mathcal{G} \cup \mathcal{H}$-derivation. Thus, intuitively, $\mathrm{rc}^{S}_{\mathcal{G}/\mathcal{H}}(n)$ measures the complexity of $\mathcal{G} \cup \mathcal{H}$, where applications of rules from $\mathcal{H}$ are free.

The following is a straight forward adaption of H. Zankl and M. Korp [30, Theorem 3.6].

▶ **Proposition 2.** *For three GRSs $\mathcal{G}_1, \mathcal{G}_2$ and $\mathcal{H}$ and any set of TGs S, we have*

$$\mathrm{rc}^{S}_{\mathcal{G}_1\cup\mathcal{G}_2/\mathcal{H}}(n) \leqslant_{\mathsf{k}} \mathrm{rc}^{S}_{\mathcal{G}_1/\mathcal{G}_2\cup\mathcal{H}}(n) + \mathrm{rc}^{S}_{\mathcal{G}_2/\mathcal{G}_1\cup\mathcal{H}}(n) \;, \quad \textit{for all } n \in \mathbb{N}.$$

Proof of Proposition 2. Fix a TG T such that both $dh_{\mathcal{G}_1/\mathcal{G}_2\cup\mathcal{H}}(T)$ and $dh_{\mathcal{G}_2/\mathcal{G}_1\cup\mathcal{H}}(T)$ are defined, i.e., T neither admits a $\mathcal{G}_1 \cup \mathcal{G}_2 \cup \mathcal{H}$ derivation with infinitely many applications of rules from $\mathcal{G}_1$ or from $\mathcal{G}_2$. Clearly, in any such reduction of T, we can estimate the number $l \in \mathbb{N}$ of applications of rule from $\mathcal{G}_1$ by $dh_{\mathcal{G}_1/\mathcal{G}_2 \cup \mathcal{H}}(T)$. Likewise, the number $k \in \mathbb{N}$ of applications of rule from $\mathcal{G}_2$ is bounded by $dh_{\mathcal{G}_2/\mathcal{G}_1 \cup \mathcal{H}}(T)$. In total thus, the considered reduction admits $k + l \leqslant dh_{\mathcal{G}_1/\mathcal{G}_2 \cup \mathcal{H}}(T) + dh_{\mathcal{G}_2/\mathcal{G}_1 \cup \mathcal{H}}(T)$ applications of rules from $\mathcal{G}_1 \cup \mathcal{G}_2$. As the considered reduction was arbitrary, this implies

$$dh_{\mathcal{G}_1 \cup \mathcal{G}_2/\mathcal{H}}(T) \leqslant_{\mathsf{k}} dh_{\mathcal{G}_1/\mathcal{G}_2 \cup \mathcal{H}}(T) + dh_{\mathcal{G}_2/\mathcal{G}_1 \cup \mathcal{H}}(T) \ ,$$

which then easily generalises to the runtime complexity function. ◀

4 An Interpretation Method for Graph Rewriting

In this section we introduce an *interpretation method* for graph rewrite systems. We start with a trivial extension of *quasi-interpretations* from terms to term graphs.

▶ **Definition 3.** An $\mathcal{F}$*-algebra* $\mathcal{A}$ for a signature $\mathcal{F}$ consists of a set A, the *carrier*, together with operations $\mathtt{f}_{\mathcal{A}} : A^k \to A$ for every k-ary function symbol $\mathtt{f} \in \mathcal{F}$. For an $\mathcal{F}$-algebra $\mathcal{A}$, *assignment* $\alpha : V_T \to A$ and TG T, the *interpretation* $[\![u]\!]_T^{\mathcal{A},\alpha}$ *of a node* u in T is defined as follows.

$$[\![u]\!]_T^{\mathcal{A},\alpha} := \begin{cases} \mathtt{f}_{\mathcal{A}}([\![u_1]\!]_T^{\mathcal{A},\alpha}, \dots, [\![u_k]\!]_T^{\mathcal{A},\alpha}) & \text{if } T(u) = \mathtt{f}(u_1, \dots, u_k), \\ \alpha(u) & \text{otherwise.} \end{cases}$$

Throughout the following, we fix the algebra $\mathcal{A}$ and write $[\![u]\!]_T^{\alpha}$ instead of $[\![u]\!]_T^{\mathcal{A},\alpha}$. Similar, we may drop the reference to α when T is ground. Note that the interpretation $[\![u]\!]_T$ corresponds to the interpreting of the term obtained unfolding the subgraph $T{\upharpoonright}u$. As such, it cannot observe sharing. This, in turn, allows us to prove the following, even in the case where matching is non-injective.

▶ **Lemma 4.** *Suppose $S \geqslant_m T$ holds for two TGs S and T, where T is ground. For all $u \in V_S$, define $\alpha(u) := [\![m(u)]\!]_T$. Then $[\![u]\!]_S^{\alpha} = [\![m(u)]\!]_T$ holds for all nodes u of S.*

Proof. The proof is by induction on the term graph structure of S. In the base case, where we consider a variable node u of S, we have $[\![u]\!]_S^{\alpha} = \alpha(u) = [\![m(u)]\!]_T$ as desired. In the inductive step, we consider a node u in S with $S(u) = \mathtt{f}(u_1, \dots, u_k)$. As by assumption the function m is morphic in u, we see that $T(m(u)) = \mathtt{f}(m(u_1), \dots, m(u_k))$. Thus by definition and induction hypothesis,

$$[\![u]\!]_S^{\alpha} = \mathtt{f}_{\mathcal{A}}([\![u_1]\!]_S^{\alpha}, \dots, [\![u_k]\!]_S^{\alpha}) = \mathtt{f}_{\mathcal{A}}([\![m(u_1)]\!]_T, \dots, [\![m(u_k)]\!]_T) = [\![m(u)]\!]_T \ . \qquad ◀$$

Let $>_A$ denote a *proper*, i.e. transitive and irreflexive, order on the carrier A of an $\mathcal{F}$-algebra $\mathcal{A}$, and denote by $\geqslant_A$ its reflexive closure. Then $(\mathcal{A}, >_A)$ is a *weakly monotone* $\mathcal{F}$*-algebra* (*WMA* for short) if all interpretations $\mathtt{f}_{\mathcal{A}} : A^k \to A$ are monotone with respect to $\geqslant_A$. Here $\mathtt{f}_{\mathcal{A}}$ is *monotone with respect to an order* $\succ$ on A if

$$a_i \succ b \implies \mathtt{f}_{\mathcal{A}}(a_1, \dots, a_i, \dots, a_k) \succ \mathtt{f}_{\mathcal{A}}(a_1, \dots, b, \dots, a_k) \ .$$

▶ **Definition 5.** A WMA $(\mathcal{A}, >_A)$ is called a *quasi-model* for a GRS $\mathcal{G}$ if $[\![l]\!]_G^{\mathcal{A},\alpha} \geqslant_A [\![r]\!]_G^{\mathcal{A},\alpha}$ holds for all rules $(G, l, r) \in \mathcal{G}$ and all assignments $\alpha : V_G \to A$.

By weak monotonicity, the quasi-model condition on $\mathcal{G}$ extends to $\to_{\mathcal{G}}$. More precise, the following lemma holds. Here, for a step $S \rightsquigarrow_{\langle L \to R, m, u \rangle} T$, where by definition $T = S\langle m(R) \rangle_u$, we say that a node $w \in N_T$ *originates from a node* v in S if either $w = v \in N_S$, or $w = \mathsf{rt}(m(R))$ and $v = u$. In particular, the root of T always originates from the root of S.

▶ **Lemma 6.** *Let $(\mathcal{A}, >_A)$ be a quasi-model for a GRS $\mathcal{G}$, and consider a step $S \to_{\mathcal{G}} T$. Then $[\![v]\!]_S \geqslant_A [\![w]\!]_T$ holds for every node $w \in N_T$ that originates from a node $v \in N_S$.*

Proof. Suppose $S \rightsquigarrow_{\langle L \to R, m, u\rangle} T$ for a renaming $L \to R$ of a rule in $\mathcal{G}$, and fix a node $w \in N_T$ that originates from $v \in N_S$. The only non-trivial case is when the node w lies along the path from the root of T to the root of the plugged graph $m(R)$, as otherwise $T{\restriction}w = S{\restriction}v$ and thus trivially $[\![v]\!]_S = [\![w]\!]_T$.

Hence fix a node w along this path. The proof is by induction on the distance of w to the $\mathsf{rt}(m(R))$. In the base case, $w = \mathsf{rt}(m(R))$. We distinguish two cases. In the first case, w is a node of S, and thus w originates from itself. Thus $L \to R$ is a collapsing rule which implies $S{\restriction}w = T{\restriction}w$, and hence the claim follows in this case. Otherwise, $w = \mathsf{rt}(m(R)) = \mathsf{rt}(R)$ is a fresh node and w originates from the redex node u. Recall that by construction, the right-hand side R is embedded via $\underline{m}$ in T at node $\mathsf{rt}(R)$, i.e. $R \geqslant_{\underline{m}} T{\restriction}\,\mathsf{rt}(R)$ holds. Define the assignment α by $\alpha(v') := [\![m(v')]\!]_S = [\![m(v')]\!]_T$ for all variable nodes v' of L. As we also have $L \geqslant_m S{\restriction}u$, two applications of Lemma 4 and the quasi-model condition yields:

$$[\![u]\!]_S = [\![m(\mathsf{rt}(L))]\!]_S = [\![\mathsf{rt}(L)]\!]_L^\alpha \geqslant_A [\![\mathsf{rt}(R)]\!]_R^\alpha = [\![\underline{m}(\mathsf{rt}(R))]\!]_T = [\![\mathsf{rt}(R)]\!]_T \ .$$

This concludes the base case. The inductive step then follows directly from the induction hypothesis and weak monotonicity of the quasi-model. ◀

Incorporating Sharing

Our approach to sharing is simple but effective. Conceptually, the semantics imposed by a quasi-model $\mathcal{A}$ are used to associate a notion of *size* to term graphs. An alternative interpretation $\mathcal{B}$ on operation symbols $\mathtt{f} \in \mathcal{D}$ can then be used to measure the complexity of calls to $\mathtt{f}$, where the *size* of the arguments is given by $\mathcal{A}$. A term graph T is then interpreted by summing up the interpretation of all calls to defined symbols in T. Conditions put on rewrite rules then ensure that T interpreted gives a bound on the length of reductions of T. However, as soon as we move to the dependency pair setting, the separation into two algebras $\mathcal{A}$ and $\mathcal{B}$ becomes inessential. In the following, we therefore restrict ourselves to a single algebra that is used to measure sizes as well as the complexity of function calls. This intuition is formalised as follows.

Let $\mathcal{A}$ be an $\mathcal{F}$-algebra, equipped with a binary operation $\oplus$ and constant 0_A such that $(A, \oplus, 0_A)$ forms a commutative monoid, that is, $\oplus :\ A \times A \to A$ is associative and commutative, with identity 0_A. Then $(\mathcal{A}, \oplus, 0_A)$ is called an *abelian* $\mathcal{F}$-algebra. Furthermore, if $(\mathcal{A}, >_A)$ is a WMA, and $\oplus$ is monotone with respect to $>_A$ (and hence also with respect to $\geq_A$), then $((\mathcal{A}, \oplus, 0_A), >_A)$ is called a *weakly-monotone abelian algebra* (*WMAA* for short). Notice that addition ($\oplus$) extends in the obvious way to summation $\sum$ over finite multisets over A, in particular, the summation over an empty set is 0_A.

▶ **Definition 7.** Let $(\mathcal{A}, \oplus, 0_A)$ be an abelian $\mathcal{F}$-algebra. For a TG T, an assignment $\alpha :\ V_T \to A$ we define the $\mathcal{D}$-*interpretation* $[\![T]\!]_{\mathcal{D}}^{\mathcal{A},\alpha}$ *of* T by

$$[\![T]\!]_{\mathcal{D}}^{\mathcal{A},\alpha} := \sum_{u \in N_T^{\mathcal{D}}} [\![u]\!]_T^{\mathcal{A},\alpha} \ .$$

As before, we drop the index $\mathcal{A}$ in $[\![T]\!]_{\mathcal{D}}^{\mathcal{A},\alpha}$ when clear from context, and the assignment α for ground term graphs. Note that as a particular consequence of Lemma 4, the interpretation of isomorphic term graphs coincides.

▶ **Lemma 8.** *Let S and T be two ground TGs. If $S \cong T$ then $[\![S]\!]_{\mathcal{D}} = [\![T]\!]_{\mathcal{D}}$.*

Proof. By assumption, there exists a bijective morphism $S \geqslant_m T$. Lemma 4 yields $[\![u]\!]_S = [\![m(u)]\!]_T$, for all nodes u of S. Since m is bijective and respects the labeling of nodes, we conclude

$$[\![S]\!]_{\mathcal{D}} = \sum_{u \in N_S^{\mathcal{D}}} [\![u]\!]_S = \sum_{u \in N_S^{\mathcal{D}}} [\![m(u)]\!]_T = \sum_{v \in N_T^{\mathcal{D}}} [\![v]\!]_T = [\![T]\!]_{\mathcal{D}} \ . \qquad ◀$$

In the following, we give a sufficient criterion for an embedding of the rewrite relation $\rightarrow_{\mathcal{G}}$ into $>_A$ via the interpretation $[\![\cdot]\!]_{\mathcal{D}}$, that is, $S \rightarrow_{\mathcal{G}} T$ implies $[\![S]\!] >_A [\![T]\!]$. A first attempt might be to require that $[\![L]\!]_{\mathcal{D}}^{\alpha} >_A [\![R]\!]_{\mathcal{D}}^{\alpha}$ holds for all rules $L \rightarrow R \in \mathcal{G}$ and assignments α. The following example clarifies that such an orientation of rewrite rules is not sufficient, even when $(\mathcal{A}, \oplus, 0_A)$ is a quasi-model for $\mathcal{G}$.

▶ **Example 9.** Consider the one-rule GRS $\mathcal{G}_1 := \{\mathtt{f}(\mathtt{g}) \rightarrow \mathbf{c}\}$, over a signature $\mathcal{F}_1$ consisting of defined symbols $\mathcal{D} = \{\mathtt{f}, \mathtt{g}\}$, the constructor $\mathbf{c}$, and an additional binary constructor $\mathbf{d}$. We consider the WMAA $((\mathcal{A}_1, +, 0), >_{\mathbb{N}})$ over $\mathbb{N}$, where the interpretation $\mathcal{A}_1$ is defined by:

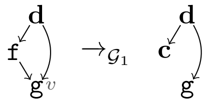
$$\mathtt{g}_{\mathcal{A}_1} := 1 \qquad \mathtt{f}_{\mathcal{A}_1}(x) := 0 \qquad \mathbf{c}_{\mathcal{A}_1} := 0 \qquad \mathbf{d}_{\mathcal{A}_1}(x, y) := 0 \ .$$

Then $((\mathcal{A}_1, +, 0), >_{\mathbb{N}})$ constitutes a quasi-model for $\mathcal{G}_1$. Let G denote the graph underlying the rule $\mathtt{f}(\mathtt{g}) \rightarrow \mathbf{c}$, and let $u_{\mathtt{g}}$ be the node labeled by $\mathtt{g}$ in G. Then

$$[\![\mathtt{f}(\mathtt{g})]\!]_{\mathcal{D}}^{\mathcal{A}_1} = \mathtt{f}_{\mathcal{A}_1}([\![u_{\mathtt{g}}]\!]_G) + \mathtt{g}_{\mathcal{A}_1} = 0 + 1 = 1 >_{\mathbb{N}} 0 = [\![\mathbf{c}]\!]_{\mathcal{D}}^{\mathcal{A}_1} \ .$$

On the other hand, for a binary constructor $\mathbf{d}$, the GRS $\mathcal{G}_1$ gives rise to a rewrite step

$S :=$



$=: T$.

However this step is not embedded into $>_{\mathbb{N}}$:

$$[\![S]\!]_{\mathcal{D}}^{\mathcal{A}_1} = \mathtt{f}_{\mathcal{A}_1}([\![v]\!]_S) + \mathtt{g}_{\mathcal{A}_1} = 1 \not>_{\mathbb{N}} 1 = \mathtt{g}_{\mathcal{A}_1} = [\![T]\!]_{\mathcal{D}}^{\mathcal{A}_1} \ .$$

The inequality $[\![L]\!]_{\mathcal{D}}^{\alpha} >_A [\![R]\!]_{\mathcal{D}}^{\alpha}$ is not suitably reflecting upon the replacements of nodes underlying a step $S \rightsquigarrow_{\langle L \rightarrow R, m, u\rangle} T$, in the presence of sharing. Although in the above example the shared node labeled by $\mathtt{g}$ of the reduct lies within the matched pattern but not in the right-hand side of the applied rule, it does not vanish in the reduct.

We overcome this issue via the notion of difference set, introduced in Section 3, that characterises the node replacements underlying a rewrite step (see Proposition 1).

▶ **Definition 10.** Let $(\mathcal{A}, \oplus, 0_A)$ be an abelian $\mathcal{F}$-algebra and fix an order $\succ$ on the carrier A. We say that a rule $L \rightarrow R = (G, l, r)$ *is oriented by* $\succ$ (with respect to the algebra $\mathcal{A}$ and defined symbols $\mathcal{D}$) if

$$[\![l]\!]_G^{\mathcal{A},\alpha} \succ \sum_{u \in \Delta^{\mathcal{D}}(L \rightarrow R)} [\![u]\!]_G^{\mathcal{A},\alpha} \quad \text{holds for all assignments } \alpha.$$

A GRS $\mathcal{G}$ is called *oriented by* $\succ$ if all rules in $\mathcal{G}$ are oriented by $\succ$.

The following constitutes the main technical lemma of this section.

▶ **Lemma 11.** *Let $((\mathcal{A}, \oplus, 0_A), >_A)$ be an abelian quasi-model for a GRS $\mathcal{G}$ and suppose the rule $L \to R \in \mathcal{G}$ is oriented by some $\succ \in \{\geqslant_A, >_A\}$. Then*

$$S \to_{L \to R} T \implies [\![S]\!]_{\mathcal{D}} \succ [\![T]\!]_{\mathcal{D}} \ .$$

Proof. Fix $\succ \in \{\geqslant_A, >_A\}$ and a rule $L \to R = (G, l, r)$ which is oriented by $\succ$:

$$[\![l]\!]_G^{\mathcal{A},\alpha} \succ \sum_{v \in \Delta^{\mathcal{D}}(L \to R)} [\![v]\!]_G^{\mathcal{A},\alpha} \ . \tag{$\star$}$$

We prove that for a pre-reduction step $S \rightsquigarrow_{\langle L \to R, m, u\rangle} T$, we have $[\![S]\!]_{\mathcal{D}} \succ [\![T]\!]_{\mathcal{D}}$. As Lemma 4 allows us to lift the inequality ($\star$) to renamings of $L \to R$, and since the interpretation of isomorphic term graphs coincides (Lemma 8), the lemma follows from this. Define P as the restriction of nodes $N_S \setminus \{u\}$ to nodes in T, and define $Q := \{\underline{m}(v) \mid v \in \Delta^{\mathcal{D}}(L \to R)\}$. Thus $N_T^{\mathcal{D}} \subseteq P \cup Q$, by Proposition 1. By Lemma 6 we have

$$[\![v]\!]_S \geqslant_A [\![v]\!]_T \qquad \text{for all } v \in P. \tag{$\dagger$}$$

Furthermore, since $R \geqslant_{\underline{m}} T{\restriction}\underline{m}(\mathsf{rt}(R))$, by Lemma 4 we see that

$$[\![v]\!]_G^{\alpha} = [\![\underline{m}(v)]\!]_T \qquad \text{for all } v \in \Delta^{\mathcal{D}}(L \to R). \tag{$\ddagger$}$$

We conclude:

$$\begin{aligned}
[\![S]\!]_{\mathcal{D}} \geqslant_A \sum_{v \in P} [\![v]\!]_S \oplus [\![u]\!]_S &= \sum_{v \in P} [\![v]\!]_S \oplus [\![l]\!]_G^{\alpha} && \textit{by definition and Lemma 4} \\
&\succ \sum_{v \in P} [\![v]\!]_S \oplus \sum_{v \in \Delta^{\mathcal{D}}(L \to R)} [\![v]\!]_G^{\alpha} && \textit{using the assumption } (\star) \\
&\geqslant_A \sum_{v \in P} [\![v]\!]_T \oplus \sum_{v \in \Delta^{\mathcal{D}}(L \to R)} [\![\underline{m}(v)]\!]_T && \textit{using Equalities } (\dagger) \textit{ and } (\ddagger) \\
&= \sum_{v \in P} [\![v]\!]_T \oplus \sum_{v \in Q} [\![v]\!]_T = [\![T]\!]_{\mathcal{D}} && \underline{m} \textit{ is injective on } \Delta^{\mathcal{D}}(L \to R).
\end{aligned}$$ ◀

The following is then a straight forward consequence of Lemma 11.

▶ **Theorem 12.** *Let $((\mathcal{A}, \oplus, 0_A), >_A)$ be an abelian quasi-model for the GRSs $\mathcal{G}$ and $\mathcal{H}$. If $\mathcal{G}$ is oriented by $>_A$ and $\mathcal{H}$ is oriented by $\geqslant_A$, then $dh_{\mathcal{G}/\mathcal{H}}(T) \leqslant_{\mathsf{k}} dh_{>_A}([\![T]\!]_{\mathcal{D}}^{\mathcal{A}})$ holds for every term graph T.*

Proof. Fix a TG T with $dh_{>_A}([\![T]\!]_{\mathcal{D}}^{\mathcal{A}})$ defined. It suffices to show that every sequence of TGs $T = T_0, T_1, T_2, \ldots$ such that

$$T = T_0 \quad \to_{\mathcal{H}}^* \cdot \to_{\mathcal{G}} \cdot \to_{\mathcal{H}}^* \quad T_1 \quad \to_{\mathcal{H}}^* \cdot \to_{\mathcal{G}} \cdot \to_{\mathcal{H}}^* \quad T_2 \quad \to_{\mathcal{H}}^* \cdot \to_{\mathcal{G}} \cdot \to_{\mathcal{H}}^* \quad \cdots \ .$$

is bounded in length by $dh_{>_A}([\![T]\!]_{\mathcal{D}}^{\mathcal{A}})$. Using the assumptions on $\mathcal{G}$ and $\mathcal{H}$, Lemma 11 translates the above sequence to

$$[\![T]\!]_{\mathcal{D}}^{\mathcal{A}} = [\![T_0]\!]_{\mathcal{D}}^{\mathcal{A}} \quad \geqslant_A^* \cdot >_A \cdot \geqslant_A^* \quad [\![T_1]\!]_{\mathcal{D}}^{\mathcal{A}} \ \geqslant_A^* \cdot >_A \cdot \geqslant_A^* \quad [\![T_2]\!]_{\mathcal{D}}^{\mathcal{A}} \ \geqslant_A^* \cdot >_A \cdot \geqslant_A^* \quad \cdots \ .$$

As $\geqslant_A^* \cdot >_A \cdot \geqslant_A^* = >_A^+$, the claim is then easy to establish by definition of $dh_{>_A}$. ◀

We emphasise that in conjunction with Proposition 2, the theorem can be applied in an iterative fashion, moving successively rules from $\mathcal{G}$ two $\mathcal{H}$ until $\mathcal{G}$ is empty (see Example 15 below).

Polynomial Term Graph Interpretations

We now instantiate Theorem 12 to make it applicable in the context of *polynomial runtime complexity analysis*. With respect to term rewrite systems, various forms of interpretation have been used to determine quantitative properties, most prominently, restricted forms of *polynomial* [8] and *matrix interpretations* [22, 19]. Via Theorem 12, these techniques extend naturally to graph rewrite systems. For brevity, we focus here on polynomial interpretations into the naturals.

▶ **Lemma 13.** *Let $\mathcal{A}$ be an $\mathcal{F}$-algebra with carrier $\mathbb{N}$, such that every interpretation function $\mathtt{f}_\mathcal{A}$ is given by a polynomial of degree $k \in \mathbb{N}$. Then the following properties hold.*

1. *Suppose $\mathbf{c}_\mathcal{A}(x_1, \dots, x_k) \leqslant \sum_{1 \leqslant i \leqslant n} x_i + \delta$ holds for every $\mathbf{c} \in \mathcal{C}$ and for some $\delta \in \mathbb{N}$. Then there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ of degree k such that $dh_{>_\mathbb{N}}([\![T]\!]^\mathcal{A}_\mathcal{D}) \leqslant p(|T|)$ holds for every basic tree $T \in \triangle$.*
2. *Suppose $\mathbf{c}_\mathcal{A}(x_1, \dots, x_k) \leqslant \max_{1 \leqslant i \leqslant n} x_i + \delta$ holds for every $\mathbf{c} \in \mathcal{C}$ and for some $\delta \in \mathbb{N}$. Then there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ of degree k such that $dh_{>_\mathbb{N}}([\![T]\!]^\mathcal{A}_\mathcal{D}) \leqslant p(|T|)$ holds for every basic TG $T \in \diamondsuit$.*

Proof. Fix a TG $T \in \diamondsuit$ with root r, thus $T(r) = \mathtt{f}(u_1, \dots, u_k)$ for some $\mathtt{f} \in \mathcal{D}$ and some nodes u_i $(1 \leqslant i \leqslant k)$. Note that $dh_{>_\mathbb{N}}(n) = n$, as moreover only the root of T is labeled by a defined symbol, we conclude

$$dh_{>_\mathbb{N}}([\![T]\!]^\mathcal{A}_\mathcal{D}) = [\![T]\!]^\mathcal{A}_\mathcal{D} = [\![r]\!]_T = \mathtt{f}_\mathcal{A}([\![u_1]\!]_T, \dots, [\![u_k]\!]_T) \ .$$

Define $\gamma \in \mathbb{N}$ as the maximal constant $\delta \in \mathbb{N}$ occurring in the interpretation $\mathbf{c}_\mathcal{A}$ of a constructor. By assumption on $\mathtt{f}_\mathcal{A}$, we conclude Property 1 by observing that $[\![u_i]\!]_T \leqslant |T{\restriction}u_i| \cdot \delta$ holds for all $1 \leqslant i \leqslant k$ whenever $T \in \triangle$. This follows by a standard induction on $T{\restriction}u_i$. Note that in the inductive step we makes essential use of the tree shape of T. Concerning Property 2, the form put on interpretations of constructors allows us to dispense the assumption $T \in \triangle$. Indeed, here $[\![u_i]\!]_T$ is bounded by a linear function in the depth of the graphs $T{\restriction}u_i$ $(1 \leqslant i \leqslant k)$. ◀

If the pre-conditions of Lemma 13(1) (Lemma 13(2), respectively) are satisfied, we call $\mathcal{A}$ a $\triangle$*-restricted* ($\diamondsuit$*-restricted*) *polynomial interpretation* of degree k. The following, then, is a consequence of Theorem 12 and Lemma 13. In essence, a $\triangle$-restricted polynomial interpretation permits the interpretation of values linearly in their size, whereas $\diamondsuit$-restricted polynomial interpretations measures values in their depth.

▶ **Corollary 14.** *Let $((\mathcal{A}, +, 0), >_\mathbb{N})$ be an abelian quasi-model for GRSs $\mathcal{G}$ and $\mathcal{H}$. Suppose $\mathcal{G}$ is oriented by $>_\mathbb{N}$ and $\mathcal{H}$ is oriented by $\geqslant_\mathbb{N}$ (with respect to defined symbols $\mathcal{D}$).*

1. *If $\mathcal{A}$ is a $\triangle$-restricted polynomial interpretation of degree k, then $\mathrm{rc}^\triangle_{\mathcal{G}/\mathcal{H}}(n) \in \mathrm{O}(n^k)$.*
2. *If $\mathcal{A}$ is a $\diamondsuit$-restricted polynomial interpretation of degree k, then $\mathrm{rc}^\diamondsuit_{\mathcal{G}/\mathcal{H}}(n) \in \mathrm{O}(n^k)$.*

▶ **Example 15.** Consider the following GRS that flattens trees to lists:

$$\begin{array}{llll} 1\colon \mathtt{flatten}(\mathbf{l}) \to [\,] & 2\colon \mathtt{flatten}(\mathbf{n}(e,s,t)) \to e : (\mathtt{flatten}(s) +\!\!+ \mathtt{flatten}(t)) \\ 3\colon \quad [\,] +\!\!+ ys \to ys & 4\colon \quad (x : xs) +\!\!+ ys \to x : (xs +\!\!+ ys) \ . \end{array}$$

Collect in $\mathcal{G}_\mathtt{flatten}$ the rules defining $\mathtt{flatten}$, likewise collect in $\mathcal{G}_{+\!\!+}$ the ones defining $+\!\!+$. Define the $\triangle$-restricted polynomial interpretation $\mathcal{A}_2$ such that

$$\begin{array}{lll} \mathbf{l}_{\mathcal{A}_2} := 1 & \mathbf{n}_{\mathcal{A}_2}(e,s,t) := 1 + s + t & \mathtt{flatten}_{\mathcal{A}_2}(t) := t \\ [\,]_{\mathcal{A}_2} := 0 & x :_{\mathcal{A}_2} xs := 1 + xs & xs +\!\!+_{\mathcal{A}_2} ys := xs + ys \end{array}$$

Then it can be verified that $((\mathcal{A}_2, +, 0), >_{\mathbb{N}})$ is a quasi-model for the considered GRS, and moreover, orients the rules in $\mathcal{G}_{\mathtt{flatten}}$ strictly, and rules from $\mathcal{G}_{+\!\!+}$ weakly. Note that $\mathcal{A}_2$ is a $\triangle$-restricted polynomial interpretation of degree 1. By Proposition 2 and Corollary 14, the runtime complexity of $\mathcal{G}$ on trees is bounded by $\mathrm{O}(n) + \mathrm{rc}^{\triangle}_{\mathcal{G}_{+\!\!+}/\mathcal{G}_{\mathtt{flatten}}}(n)$. Now define a second quasi-model $\mathcal{A}_3$ like $\mathcal{A}_2$, but with $[\,]_{\mathcal{A}_3} := 1$ and $xs +\!\!+_{\mathcal{A}_3} ys := xs$ instead. This interpretation orients rules from $\mathcal{G}_{+\!\!+}$ strictly, and rules from $\mathcal{G}_{\mathtt{flatten}}$ weakly, and thus $\mathrm{rc}^{\triangle}_{\mathcal{G}_{+\!\!+}/\mathcal{G}_{\mathtt{flatten}}}(n) \in \mathrm{O}(n)$ by Corollary 14. Conclusively, the overall runtime is linear on trees.

Note that neither of the two interpretations is a $\Diamond$-restricted polynomial interpretation, due to the interpretation of the constructor $\mathbf{n}$. Indeed, the runtime complexity of the system on general term graphs is exponential, e.g., consider the flattening of a fully collapsed graph.

5 Dependency Pairs for Complexity Analysis

In the following, we suite *dependency tuples* [23], a variant of dependency pairs admissible for the innermost runtime complexity analysis of term rewrite systems, to graph rewrite systems. In the context of term graph rewriting, we will see that soundness of the method is independent of a particular reduction strategy.

For each k-ary defined symbol $\mathtt{f} \in \mathcal{D}$, let $\mathtt{f}^\sharp$ denote a fresh function symbol also of arity k, the *dependency pair symbol* of $\mathtt{f}$. Marked defined symbols are collected in $\mathcal{D}^\sharp$. Furthermore, let $\mathcal{C}\mathsf{om}$ denote the countable infinite set of *compound symbols* c_k for all $k \in \mathbb{N}$. The arity of c_k is k. For a TG T, symbol $\mathtt{f}$ and nodes $\{u_1, \dots, u_{\mathsf{ar}(\mathtt{f})}\} \in N_T$, we write $\mathtt{f}(T{\upharpoonright}u_1, \dots, T{\upharpoonright}u_{\mathsf{ar}(\mathtt{f})})$ for the term graph $S{\upharpoonright}u_{\mathtt{f}}$, where S is defined as the extension of the TG T by a fresh node $u_{\mathtt{f}} \notin N_T$ with $S(u_{\mathtt{f}}) = \mathtt{f}(u_1, \dots, u_{\mathsf{ar}(\mathtt{f})})$. For a TG T rooted in a defined symbol $\mathtt{f} \in \mathcal{D}$, i.e. $T(\mathsf{rt}(T)) = \mathtt{f}(u_1, \dots, u_k)$, the *marking* $T^\sharp$ of T is defined as $\mathtt{f}^\sharp(T{\upharpoonright}u_1, \dots, T{\upharpoonright}u_{\mathsf{ar}(\mathtt{f})})$. This notation is naturally extended to sets.

▶ **Definition 16.** Let $L \to R$ be a rule with $\Delta^{\mathcal{D}}(L \to R) = \{u_1, \dots, u_k\}$. Then the rule

$$\mathsf{DP}(L \to R) := L^\sharp \to \mathsf{c}_k((R{\upharpoonright}u_1)^\sharp, \dots, (R{\upharpoonright}u_k)^\sharp) ,$$

is called the *dependency pair of* $L \to R$ (*DP* for short). We collect in $\mathsf{DP}(\mathcal{G})$ for each rule $L \to R$ a corresponding dependency pair $\mathsf{DP}(L \to R)$.

Kindly observe that according to the definition we only consider those subgraph $R{\upharpoonright}u_i$ of the right-hand side R, that are outside of the interface of the rule $L \to R$. This is akin to a similar condition for dependency pairs in termination of term rewrite systems, first observed by Dershowitz [14]. Further, observe that the definition of dependency pairs is devoid of an intermediate sharing or collapsing step. I.e. a shared node in $\mathsf{DP}(L \to R)$ will have been shared in $L \to R$ already. However, utilising the notion of *tops* [25] an alternative definition could be formalised in a straightforward way. This however, would render the step-by-step simulation shown below impossible.

▶ **Example 17.** Reconsider our motivating example from the introduction, depicted in Figure 1. Represent positive integers using two unary constructors $\mathbf{0}$, $\mathbf{1}$ and a constant ϵ. Then the definition of power is expressible as the GRS $\mathcal{G}_{\mathsf{power}}$ consisting of the following rules:

$$\begin{aligned}
&\mathtt{power}(x, \mathbf{0}(\epsilon)) \to \mathbf{1}(\epsilon)\\
&\mathtt{power}(x, \mathbf{0}(n)) \to y * y \ \textit{where}\ y = \mathtt{power}(x, n)\\
&\mathtt{power}(x, \mathbf{1}(n)) \to y * (y * x) \ \textit{where}\ y = \mathtt{power}(x, n) \ .
\end{aligned}$$

In the last two rules, the *where*-clause indicates that the recursive call is shared, i.e. represented by the node y. For brevity, we leave multiplication abstract, however, in the following we consider the symbol $(*)$ defined. Thus $\mathsf{DP}(\mathcal{G}_{\mathtt{power}})$ consists of the following three rules:

$$\begin{aligned}
&\mathtt{power}^\sharp(x, \mathbf{0}(\epsilon)) \to \mathbf{c}_0\\
&\mathtt{power}^\sharp(x, \mathbf{0}(n)) \to \mathbf{c}_2(\mathtt{power}^\sharp(x,n), y *^\sharp y) \textit{ where } y = \mathtt{power}(x,n)\\
&\mathtt{power}^\sharp(x, \mathbf{1}(n)) \to \mathbf{c}_3(\mathtt{power}^\sharp(x,n), y *^\sharp (y * x), y *^\sharp x) \textit{ where } y = \mathtt{power}(x,n)\ .
\end{aligned}$$

In the following, we establish a simulation of $\mathcal{G}$ via $\mathsf{DP}(\mathcal{G})/\mathcal{G}$. In this simulation, we will consider very specific term graphs over the new signature, so called *DP graphs*: a term graph U over the signature $\mathcal{F} \cup \mathcal{D}^\sharp \cup \mathcal{C}\mathsf{om}$ is called a *DP graph* if nodes above marked nodes, i.e. nodes u with $\mathsf{lab}_U(u) = \mathtt{f}^\sharp$, are labeled by compound symbols, and all nodes below are labeled by unmarked symbols $\mathcal{F}$. Thus one can adopt the intuition that a DP graph T denotes a finite sequence of term graphs whose root is marked, where the sequence itself is constructed via compound symbols. Note that DP graphs are closed under reductions:

▶ **Lemma 18.** *Let U be a DP graph. If $U \to_{\mathsf{DP}(\mathcal{G})\cup\mathcal{G}} V$ then V is again a DP graph.*

The following notion relates term graphs T to DP graphs U. In essence, it states that every potential redex in T is represented by its marked version in U. We drive our simulation precisely via this correspondence.

▶ **Definition 19.** Let T be a ground term graph, and let V be DP graph. Then V is *good* for T, in notation $T \ggg V$, if there is an injective function $d\,:\, N_T^{\mathcal{D}} \to N_V^{\mathcal{D}^\sharp}$ such that for all $u \in N_T^{\mathcal{D}}$, $(T{\restriction}u)^\sharp \cong V{\restriction}d(u)$ holds.

Observe that $T^\sharp$ is good for T, if T is a basic term graph. Furthermore, the right-hand side of $\mathsf{DP}(L \to R)$ is good for the part of R whose nodes lie in the difference set $\Delta(L \to R)$. In the proof of the following lemma, we tacitly employ that the relation $\ggg$ is closed under isomorphisms, i.e. $\cong \cdot \ggg \cdot \cong\, \subseteq\, \ggg$.

▶ **Lemma 20.** *If $S \ggg U$ and $S \to_{L\to R} T$, then there exists a term graph V with $U \to_{L\to R}^* \cdot \to_{DP(L\to R)} V$ and $T \ggg V$.*

Proof. Let $L \to R = (G, l, r)$. Fix a pre-reduction step $S \rightsquigarrow_{\langle L\to R,m,u\rangle} T$ and suppose $S \ggg U$, as witnessed by the injective mappings $d\,:\, N_S^{\mathcal{D}} \to N_U^{\mathcal{D}^\sharp}$.

Denote by $v_1, \dots, v_k$ all nodes labeled by a defined symbol that lie strictly above the redex node u in S, i.e. $v_i \rightharpoonup_S^+ u$ with $v_i \in N_S^{\mathcal{D}}$ holds for all $1 \leqslant i \leqslant k$. By the assumption $S \ggg U$, the markings of $S{\restriction}v_i$ are isomorphic to $U{\restriction}d(v_i)$, i.e. $(S{\restriction}v_i)^\sharp \cong_{m_i} U{\restriction}d(v_i)$ holds. This again implies that the assumed rewrite can be carried out in $U{\restriction}d(v_i)$. More precise, $\langle L \to R, m_i \circ m, m_i(u)\rangle$ is a redex in U, where the reduct of $U{\restriction}d(v_i)$ is isomorphic to the marking of $T{\restriction}v_i$, by construction. Kindly note that the nodes $m_i(u)$ are not necessarily pairwise distinct, however if two nodes $m_i(u)$ and $m_j(u)$ are distinct, then these two nodes are parallel. Moreover the nodes $m_i(u)$ $(1 \leqslant i \leqslant k)$ lie outside of $U{\restriction}d(v)$ for all $v \in N_S^{\mathcal{D}}$ with $v \notin \{v_1, \dots, v_k\}$. The latter is a simple fact following from $S \ggg U$ and the relative position of v to the redex node u in S, i.e. parallel or below. In conclusion, all rewrite steps on $m_i(u)$ can be carried out independently and in sequence, resulting in a DP graph W,

$$U \rightsquigarrow_{L_1\to R_1} \cdots \rightsquigarrow_{L_n\to R_n} W\ ,$$

for suitable renamings $L_j \to R_j$ of $L \to R$. By construction, we have (i) $(S{\restriction}v)^\sharp \cong W{\restriction}d(v)$ for all nodes $v \in N_S^{\mathcal{D}}$ strictly above u, and (ii) $U{\restriction}d(v) = W{\restriction}d(v)$ for all nodes $v \in N_S$ parallel or below u (including u).

Next, assume wlog. that the nodes of $\mathsf{DP}(L \to R)$ are disjoint from those of W. Observe that by (ii), $S{\restriction}u$ and $W{\restriction}d(u)$ are isomorphic, modulo marking of $d(u)$. And thus, since L and $L^\sharp$ coincides on all but the marking of the root node, we obtain our final DP graph V, with

$$W \rightsquigarrow_{\langle \mathsf{DP}(L\to R), m^\sharp, d(u)\rangle} V \ ,$$

for a suitable matching morphism $m^\sharp$. Observe that since marked nodes in W are all in parallel, and by injectivity of d, it follows that (iii) $W{\restriction}d(v) = V{\restriction}d(v)$ holds for all $v \in N_S^{\mathcal{D}} \setminus \{u\}$.

It remains to verify that V is good for T. To this end, for each node $v \in \Delta^{\mathcal{D}}(L \to R)$, let $v_\sharp$ denote the root of the subgraph $(R{\restriction}v)^\sharp$ occurring in the right-hand side of $\mathsf{DP}(L \to R)$. Define $e \,:\, N_T^{\mathcal{D}} \to N_V^{\mathcal{D}^\sharp}$ by

$$e(v) := \begin{cases} d(v) & \text{if } v \in N_S, \\ v_\sharp & \text{if } v \in N_R^{\mathcal{D}} \setminus N_L^{\mathcal{D}} . \end{cases}$$

Observe that e is injective, moreover, it is total as a consequence of Proposition 1. We perform case analysis on $v \in N_T^{\mathcal{D}}$ and show that the marking of $T{\restriction}v$ is isomorphic to $V{\restriction}e(v)$:

- If v is strictly above the redex node u in S then the claim follows from (i) and (iii).
- If v occurs parallel or strictly below the redex node u in S, then $S{\restriction}v = T{\restriction}v$, and $U{\restriction}d(v) = V{\restriction}d(v)$ by (ii) and (iii). Then $S \ggg U$ and definition of e proves the case.
- If $v = u$, then L lies below R in $L \to R$, hence again $S{\restriction}u = T{\restriction}u$. Note that in the considered case, l occurs in the difference set $\Delta^{\mathcal{D}}(L \to R)$, and consequently $l^\sharp$ lies also below the right-hand side of $\mathsf{DP}(L \to R)$. Thus also $U{\restriction}d(u) = W{\restriction}d(u) = V{\restriction}d(u)$ where the first equality holds by (ii) and the second by construction. We conclude as above.
- Suppose $v \in N_R^{\mathcal{D}} \setminus N_L^{\mathcal{D}} \subseteq \Delta^{\mathcal{D}}(L \to R)$. We have to show that the marking of $T{\restriction}v$ is isomorphic to $V{\restriction}e(v) = V{\restriction}v_\sharp$, for $v_\sharp$ the root of the graph $(R{\restriction}v)^\sharp$ that occurs in the right-hand side of $\mathsf{DP}(L \to R)$. Compare the rewrite $S \rightsquigarrow_{\langle L\to R, m, u\rangle} T$ with $W \rightsquigarrow_{\langle \mathsf{DP}(L\to R), m^\sharp, d(u)\rangle} V$. Observe that $R{\restriction}v$ is embedded at node v in T, i.e. $R{\restriction}v \geqslant_{\underline{m}} T{\restriction}v$ where moreover, $\underline{m}$ is injective on all nodes $N_{R\restriction v} \setminus N_L$. In a similar fashion, $(R{\restriction}v)^\sharp$ is embedded at node $v_\sharp$ in V. Since the left-hand side of $L \to R$ is isomorphic to the left-hand side of $\mathsf{DP}(L \to R)$ modulo marking of the root, and since the redexes $S{\restriction}u$ and $W{\restriction}d(u)$ are isomorphic modulo marking of $d(u)$, it is then not difficult to conclude that $T{\restriction}v$ is isomorphic to $V{\restriction}v_\sharp$, modulo marking of $v_\sharp$.

By Proposition 1, we exhausted all cases and conclude the lemma. ◀

Note that for a rule $L \to R \in \mathcal{G}$, the sequence $U \to^*_{L\to R} \cdot \to_{\mathsf{DP}(L\to R)} V$ corresponds to a relative step $U \to_{\mathsf{DP}(\mathcal{G})/\mathcal{G}} V$. Thus, Lemma 20 is directly applicable in a relative setting.

▶ **Theorem 21.** *Let $\mathcal{G}$ and $\mathcal{H}$ be GRSs and define $\mathcal{Q} := \mathsf{DP}(\mathcal{G})/\mathsf{DP}(\mathcal{H}) \cup \mathcal{G} \cup \mathcal{H}$. Then*

$$S \ggg U \implies dh_{\mathcal{G}/\mathcal{H}}(S) \leqslant_{\mathsf{k}} dh_{\mathcal{Q}}(U) \ .$$

Proof. Consider first a relative step $S \to_{\mathcal{G}/\mathcal{H}} T$, i.e. $S \to^*_{\mathcal{H}} \cdot \to_{\mathcal{G}} \cdot \to^*_{\mathcal{H}} T$, and let U be a term graph that is good for S. As a consequence of Lemma 20, we obtain a term graph V that is good for T such that $U \to^*_{\mathsf{DP}(\mathcal{H})/\mathcal{H}} \cdot \to_{\mathsf{DP}(\mathcal{G})/\mathcal{G}} \cdot \to^*_{\mathsf{DP}(\mathcal{H})/\mathcal{H}} V$, i.e. $U \to_{\mathcal{Q}} V$, holds. From this, we conclude by following the structure of the proof of Theorem 12. ◀

Conclusively, we obtain the main result of this section.

▶ **Corollary 22.** *Let $\mathcal{G}$ and $\mathcal{H}$ be GRSs, let $\mathcal{P}$ denote the relative system $\mathcal{G}/\mathcal{H}$, and let $\mathcal{Q}$ denote the relative system $\mathsf{DP}(\mathcal{G})/\mathsf{DP}(\mathcal{H}) \cup \mathcal{G} \cup \mathcal{H}$. Then for any set $S \subseteq \diamondsuit$ of basic term graphs, $\mathrm{rc}_{\mathcal{P}}^{S}(n) \leqslant_{\mathsf{k}} \mathrm{rc}_{\mathcal{Q}}^{S^\sharp}(n)$ holds for all $n \in \mathbb{N}$.*

Suiting the Interpretation Method

Our interpretation method (Corollary 14) is readily applicable in the context of relative dependency pair systems of the form of $\mathcal{Q}$ from Corollary 22. However, the imposed constraints are unnecessarily strict, as we only need to account for steps due to dependency pairs, i.e. rewrites on marked symbols. We thus embed reductions via $[\![\cdot]\!]_{\mathcal{D}^\sharp}$ rather than $[\![\cdot]\!]_{\mathcal{D}^\sharp \cup \mathcal{D}}$. This, in turn, allows us to weaken the pre-conditions of Corollary 14. The following constitutes the central observation.

▶ **Lemma 23.** *Let $((\mathcal{A}, \oplus, 0_A), >_A)$ be a quasi-model for the GRS $\mathcal{G}$, and let U be a DP graph. Then $U \to_{\mathcal{G}} V$ implies $[\![U]\!]^{\mathcal{A}}_{\mathcal{D}^\sharp} \geqslant_A [\![V]\!]^{\mathcal{A}}_{\mathcal{D}^\sharp}$.*

Proof. Consider a step $U \to_{\mathcal{G}} V$. By the shape of U, this step has to take place strictly below marked symbols, and consequently $N := N_U^{\mathcal{D}^\sharp} = N_V^{\mathcal{D}^\sharp}$. Hence Lemma 6 yields $[\![u]\!]_U \geqslant_A [\![u]\!]_V$ for each $u \in N$, and thus $[\![U]\!]_{\mathcal{D}^\sharp} = \sum_{u \in N} [\![u]\!]_U \geqslant_A \sum_{u \in N} [\![u]\!]_V = [\![V]\!]_{\mathcal{D}^\sharp}$. ◀

Consider an abelian $\mathcal{F}$-algebra $(\mathcal{A}, \oplus, 0_A)$, and let $L \to R = (G, l, r)$ be a dependency pair with right-hand side $R = \mathsf{c}_k((R{\restriction}u_1)^\sharp, \dots, (R{\restriction}u_k)^\sharp)$. Then this dependency pair is oriented by an order $\succ$ on the carrier A with respect to marked defined symbols $\mathcal{D}^\sharp$ if

$$[\![l]\!]^{\mathcal{A},\alpha}_G \succ \sum_{u \in \Delta^{\mathcal{D}^\sharp}(L \to R)} [\![u]\!]^{\mathcal{A},\alpha}_G \Big(= \sum_{1 \leqslant i \leqslant k} [\![u_i]\!]^{\mathcal{A},\alpha}_G \Big) \quad \text{holds for all assignments } \alpha.$$

The following is then a simple corollary to Theorem 12, using in addition Lemma 23.

▶ **Corollary 24.** *Let $\mathcal{P}$ and $\mathcal{Q}$ be two sets of dependency pairs, and let $\mathcal{G}$ be a GRS. Let $((\mathcal{A}, \oplus, 0_A), >_A)$ be an abelian quasi-model for $\mathcal{G}$, and suppose that rules in $\mathcal{P}$ and $\mathcal{Q}$ are oriented by $>_A$ and $\geqslant_A$, respectively, with respect to the WMAA $(\mathcal{A}, \oplus, 0_A)$ and* defined symbols *$\mathcal{D}^\sharp$. Then $dh_{\mathcal{P}/\mathcal{Q} \cup \mathcal{G}}(U) \leqslant_{\mathsf{k}} dh_{>_A}([\![U]\!]_{\mathcal{D}})$ holds for every DP graph U.*

▶ **Example 25** (Continued from Example 17)**.** With the help of this final corollary, it is not difficult to bind the runtime complexity of $\mathsf{DP}(\mathcal{G}_{\mathtt{power}})/\mathcal{G}_{\mathtt{power}}$, using a $\diamondsuit$-restricted polynomial interpretation of degree one. Thus the GRS $\mathcal{G}_{\mathtt{power}}$ has linear complexity by Corollary 22, and under the assumption that multiplication has unit cost, this bound transfers to our motivating example.

6 Related Work

To the best of our knowledge this study is the first investigation towards an automated complexity analysis of *term graph rewriting*. However, in the wider scope of *graph rewriting*, complexity has been an issue.

In particular Bonfante et al. study the derivation height of specific graph rewrite systems, cf. [9]. A termination method for graph rewrite systems is established, which is based on weights. Termination induces polynomial bounds on the derivation height of the rewrite relation. As the considered graph rewrite systems always start in an initial configuration, we can roughly say that their methods establish polynomial runtime complexity of the graph computation. Due to the specific nature of the considered systems the technical results obtained in [9] cannot be compared to the results obtained in this paper. Still, the conceptional approach of proving termination of graph rewrite systems by weights and studying the induced runtime complexity is related to some degree.

In [12], H. J. Sander Bruggink, B. König and H. Zantema introduce a novel interpretation method applicable in the context of termination analysis of graph transformation systems.

Here a, possibly cyclic, graph is interpreted via its embedding into a *weighed type graph*. Sufficient orientation conditions are then put on transformation rules which ensure that the interpretation of graphs decreases during reductions. Interestingly, the method implies a linear bound on the runtime complexity of the analysed system. To overcome the limitation to such linear systems, the authors show that the method is also applicable in an iterated fashion. Then, however, sensible bounds on the runtime complexity cannot be derived. In recent work [13], H.J.S. Bruggink, B. König, D. Nolte and H. Zantema generalise the approach to type graphs over semirings. It is unclear how this generalisation relates to runtime complexity analysis, and whether it can be suited to term graphs.

Furthermore, in the literature the complexity of *interaction nets* [18] have been pondered. In contrast to term graphs considered here, interaction nets may admit cyclic structures, but on the other hand provide for more control in sharing or garbage collection, via the explicit use of duplication or erasing cells. First results on runtime complexity have been proposed by Perrinel in [24]. Furthermore, Gimenez and the second author study in [17] space and time complexities of sequential and parallel computations. The resource analysis is based on user-defined sized types in conjunction with potentials that are assigned to each cell in the net. While technically quite apart from the work presented here, there are conceptional similarities: potentials are conceivable as interpretations, and the dependency pair method implicitly combines a size analysis with a runtime analysis.

Finally, we also mention connections to [10]. Here G. Bonfante, J.-Y. Marion and J.-Y. Moyen couple a termination criterion with quasi-interpretations to derive bounds on the complexity of term rewrite systems, using memoisation to speed up computation. Partly inspired by this work, in [1] the first author together with Dal Lago introduce a machinery that incorporates sharing and memoization in order to get an even more efficient mechanism for evaluating term rewrite systems. It seems that a suitable adaptation of quasi-interpretations to term graphs, following along the lines of our adaption of polynomial interpretations, would allow the use of this machinery to strengthen the result of [10].

7 Conclusion and Future Work

In this paper we have transferred two seminal techniques in complexity analysis of term rewrite systems to term graph rewrite systems: (i) the interpretation method and (ii) the dependency pair method. Our adaptions are non-trivial, in the sense that they can observe not only term but also graph structures, i.e. take sharing into account. As our results have been obtained in the context of relative graph rewriting, we have thus established the core parts of a *complexity pair framework* for term graph rewrite systems. We expect that similar adaptions of existing processors, like for example *usable arguments* or *dependency graphs* are easily obtainable, based on the foundation provided in this paper.

An immediate concern for future work is the implementation of the proposed techniques. Using similar methods as in our existing implementation of complexity analysis of term rewrite systems [5], it is not difficult to see that all proposed methods are automatable and we do not expect any issues for the preparation of a prototype. Furthermore our motivating example highlights the interest of dedicated methods for outermost evaluation, which we want to study in the future. Also of essence is the extension of the approach to possibly cyclic graphs. More generally, one strong motivation for this work stems from our work on resource analysis of imperative programs. Existing transformations to rewrite systems, necessarily unfold the heap to a tree, as it has to be representable as a term [28]. Here we hope that the direct coupling with graph rewriting is of advantage and could provide us with a sophisticated shape analysis of the heap.

References

1 M. Avanzini and U. Dal Lago. On Sharing, Memoization, and Polynomial Time. *IC*, 2015.

2 M. Avanzini, U. Dal Lago, and G. Moser. Analysing the Complexity of Functional Programs: Higher-Order Meets First-Order. In *Proc. of 20th ICFP*, pages 152–164. ACM, 2015.

3 M. Avanzini, N. Eguchi, and G. Moser. A new Order-theoretic Characterisation of the Polytime Computable Functions. *TCS*, 585:3–24, 2015.

4 M. Avanzini and G. Moser. A Combination Framework for Complexity. *IC*, 2015.

5 M. Avanzini, G. Moser, and M. Schaper. TcT: Tyrolean Complexity Tool. In *Proc. of 22nd TACAS*, LNCS, 2016.

6 M. Avanzini, C. Sternagel, and R. Thiemann. Certification of Complexity Proofs using CeTA. In *Proc. of 26th RTA*, volume 36 of *LIPIcs*, pages 23–39, 2015.

7 E. Barendsen. Term Graph Rewriting. In *Term Rewriting Systems*, volume 55 of *CTTCS*, chapter 13, pages 712–743. Cambridge University Press, 2003.

8 G. Bonfante, A. Cichon, J.-Y. Marion, and H. Touzet. Algorithms with Polynomial Interpretation Termination Proof. *JFP*, 11(1):33–53, 2001.

9 G. Bonfante and B. Guillaume. Non-simplifying Graph Rewriting Termination. In *Proc. of 7th TERMGRAPH*, EPTCS, pages 4–16, 2013.

10 G. Bonfante, J.-Y. Marion, and J.-Y. Moyen. Quasi-interpretations: A Way to Control Resources. *TCS*, 412(25):2776–2796, 2011.

11 M. Brockschmidt, F. Emmes, S. Falke, C. Fuhs, and J. Giesl. Alternating Runtime and Size Complexity Analysis of Integer Programs. In *Proc. of 20th TACAS*, volume 8413 of *LNCS*, pages 140–155, 2014.

12 H. J. Sander Bruggink, B. König, and H. Zantema. Termination Analysis for Graph Transformation Systems. In *Proc. of 8th IFIP TC 1/WG 2.2, TCS*, volume 8705 of *LNCS*, pages 179–194, 2014.

13 H.J.S. Bruggink, B. König, D. Nolte, and H. Zantema. Proving Termination of Graph Transformation Systems Using Weighted Type Graphs over Semirings. In *Proc. of 8th ICGT*, volume 9151 of *LNCS*, pages 52–68, 2015.

14 N. Dershowitz. Termination Dependencies. In *Proc. of 6th WST*, pages 28–30. Universidad Politécnica de Valencia, 2003. Technical Report DSIC-II/15/03.

15 F. Frohn, J. Giesl, J. Hensel, C. Aschermann, and T. Ströder. Inferring Lower Bounds for Runtime Complexity. In *Proc. of 26th RTA*, LIPIcs, pages 334–349, 2015.

16 J. Giesl, T. Ströder, P. Schneider-Kamp, F. Emmes, and C. Fuhs. Symbolic Evaluation Graphs and Term Rewriting: A General Methodology for Analyzing Logic Programs. In *Proc. of 14th PPDP*, pages 1–12. ACM, 2012.

17 S. Gimenez and G. Moser. The Complexity of Interaction. In *Proc. of 43rd POPL*, pages 243–255. ACM, 2016.

18 Y. Lafont. Interaction nets. In *Proc. of 17th POPL*, pages 95–108. ACM, 1990.

19 A. Middeldorp, G. Moser, F. Neurauter, J. Waldmann, and H. Zankl. Joint Spectral Radius Theory for Automated Complexity Analysis of Rewrite Systems. In *Proc. of 4th CAI*, volume 6742 of *LNCS*, pages 1–20, 2011.

20 G. Moser. Proof Theory at Work: Complexity Analysis of Term Rewrite Systems. *CoRR*, abs/0907.5527, 2009. Habilitation Thesis.

21 G. Moser. KBOs, Ordinals, Subrecursive Hierarchies and All That. *JLC*, 2014. advance access.

22 G. Moser and A. Schnabl. Proving Quadratic Derivational Complexities using Context Dependent Interpretations. In *Proc. of 19th RTA*, volume 5117 of *LNCS*, pages 276–290, 2008.

23 L. Noschinski, F. Emmes, and J. Giesl. Analyzing Innermost Runtime Complexity of Term Rewriting by Dependency Pairs. *JAR*, 51(1):27–56, 2013.

24 M. Perrinel. On Context Semantics and Interaction Nets. In *Proc. of Joint 23rd CSL and 29th LICS*, page 73. ACM, 2014.

25 D. Plump. Simplification orders for term graph rewriting. In *Proc. 22nd MFCS*, volume 1295 of *LNCS*, pages 458–467, 1997.

26 D. Plump. Essentials of Term Graph Rewriting. *ENTCS*, 51:277–289, 2001.

27 S. L. Peyton Jones. *The Implementation of Functional Languages.* Prentice-Hall International, 1987.

28 M. Schaper. A Complexity Preserving Transformation from Jinja Bytecode to Rewrite Systems. Master's thesis, University of Innsbruck, Austria, 2014. URL: `http://cl-informatik.uibk.ac.at/users/c7031025/publications/masterthesis.pdf`.

29 J. Waldmann. Matrix Interpretations on Polyhedral Domains. In *Proc. of 26th RTA*, volume 36 of *LIPIcs*, pages 318–333, 2015.

30 H. Zankl and M. Korp. Modular Complexity Analysis for Term Rewriting. *LMCS*, 10(1:19):1–33, 2014.

Nominal Narrowing*

Mauricio Ayala-Rincón[1], **Maribel Fernández**[2], **and**
Daniele Nantes-Sobrinho[3]

1 **Departamentos de Ciência da Computação e Matemática, Universidade de Brasília, Brasília, Brazil**
ayala@unb.br

2 **Department of Informatics, King's College London, London, UK**
maribel.fernandez@kcl.ac.uk

3 **Departamentos de Ciência da Computação e Matemática, Universidade de Brasília, Brasília, Brazil**
dnantes@mat.unb.br

Abstract

Nominal unification is a generalisation of first-order unification that takes α-equivalence into account. In this paper, we study nominal unification in the context of equational theories. We introduce nominal narrowing and design a general nominal E-unification procedure, which is sound and complete for a wide class of equational theories. We give examples of application.

1998 ACM Subject Classification F.4.1 [Mathematical Logic] lambda calculus and related systems, D.3.3 [Language Constructs and Features] data types and structures, frameworks

Keywords and phrases Nominal Rewriting, Nominal Unification, Matching, Narrowing, Equational Theories

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.11

1 Introduction

This is a paper about nominal unification in the context of equational theories.

Nominal techniques [16] facilitate reasoning in systems with binding operators, where α-equivalence must be taken into account. In nominal syntax [11, 29], *atoms*, which are used to represent object-level variables in the intended applications, can be abstracted: $[a]t$ denotes the abstraction of the atom a in the term t. *Variables* in nominal terms represent unknown parts of terms and behave like first-order variables, but nominal variables may be decorated with atom permutations. Permutations act on terms, swapping atoms (e.g., $(a\ b) \cdot t$ means that a and b are swapped everywhere in t).

Nominal syntax has interesting properties. Nominal unification [29], that is, unification of nominal terms modulo α-equivalence, is decidable and unitary. Efficient nominal unification algorithms are available [3, 19]. Nominal matching, a key ingredient in the definition of nominal rewriting [11], is a particular case of nominal unification that can be solved in linear time [4]. Nominal rewriting [11] can be used to reason in nominal equational theories (see [12]; a completion procedure is described in [14]).

However, to our knowledge, the concept of *nominal* E*-unification*, i.e., nominal unification in the context of an equational theory E, has not been addressed in previous works. Nominal E-unification is needed to solve equations between nominal terms where the function symbols

* This work was partially supported by CAPES PVE 146/2012 and CNPq UNIVERSAL 476952/2013-1.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 11; pp. 11:1–11:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

satisfy properties defined by an equational theory. Nominal E-unification has applications in, e.g., functional-logical programming languages and analysis of cryptographic protocols.

The main contributions of this paper are:

- We define nominal E-unification problems, and the *nominal narrowing* relation, and study the relationship between nominal rewriting and nominal narrowing.
- We show that Hullot's results [17] (with the corrections from [1, 24]) relating first-order narrowing derivations and first-order E-unifiers can be transferred to nominal systems. Thus, we obtain a nominal E-unification procedure that is sound and complete for the class of convergent closed equational theories. We give examples to illustrate these results.
- We define *basic nominal narrowing* and provide sufficient conditions for termination of nominal narrowing derivations, which can be used to prove the decidability of nominal E-unification for certain equational theories.

Related Work. Narrowing has traditionally been used to solve equations in initial and free algebras modulo a set of equations. It is well-known that narrowing is a programming feature that allows integration of functional and logical programming languages [8, 20]. Narrowing was originally introduced for theorem proving [17], but nowadays it is used in type inference [27] and verification of cryptographic protocols [23], amongst other areas. Narrowing gives rise to a complete E-unification procedure if E is defined by a convergent rewrite system, but it is generally inefficient. Several strategies have been designed to make narrowing-based E-unification procedures more efficient by reducing the search space (e.g., basic narrowing [17] and variant narrowing [9], the latter inspired by the notion of E-variant [6]) and sufficient conditions for termination have been obtained [17, 9, 1]. In this paper we develop basic nominal narrowing strategies and associated termination conditions, and leave the study of other complete strategies for future work.

Nominal unification is closely related to higher-order pattern unification [18] and there is previous work addressing higher-order pattern E-unification: Prehofer [26] introduced higher-order narrowing and some variants (such as lazy narrowing, conditional narrowing, pattern narrowing), and considered applications of narrowing as an inference rule in logic and functional programming. Nominal extensions of logic and functional programming languages are already available (see, e.g., [28, 5]), and nominal narrowing could play a similar role in the definition of a functional-logic programming language.

Overview of the paper: Section 2 recalls basic concepts in nominal unification and rewriting. Section 3 introduces the notion of *nominal narrowing*, presents results relating nominal narrowing and nominal equational unification, and gives examples of application. Section 4 introduces *basic nominal narrowing* and the results regarding the termination of narrowing. Section 5 contains the conclusions and directions for future work.

2 Nominal Rewriting

We recall below the definitions of nominal unification and nominal rewriting; for more details we refer the reader to [11, 29].

2.1 Nominal terms and α-equivalence

A *nominal signature* Σ is a set of *function symbols* $f, g, \ldots$, each with a fixed *arity* $n \geq 0$. Fix a countably infinite set $\mathcal{X}$ of *variables* $X, Y, Z, \ldots$; these represent meta-level unknowns. Also, fix a countably infinite set $\mathcal{A}$ of *atoms* $a, b, c, n, x, \ldots$; these represent object-level variables. We assume that Σ, $\mathcal{X}$ and $\mathcal{A}$ are pairwise disjoint.

Nominal terms are generated by the grammar: $t ::= a \,|\, \pi \cdot X \,|\, [a]t \,|\, f(t_1, \ldots, t_n)$. Terms are called respectively *atoms*, *suspensions*, *abstractions* and *function applications*. We write $V(t)$ for the set of variables occurring in t, $A(t)$ for the set of atoms mentioned in t, and $atm(t)$ for the set of atoms that occur as subterms in t. For example, $A([a]b) = \{a, b\}$, $b \in atm([a]b)$, $a \notin atm([a]b)$. *Ground terms* are terms without variables, they may still contain atoms. The occurrences of a in a term are said to be *bound* (or abstracted) if they occur in the scope of an abstraction, otherwise they are said to be *free* (or unabstracted).

A *permutation* π is a bijection on atoms, with finite domain. $\pi \circ \pi'$ denotes *functional composition* of permutations and π^{-1} denotes the *inverse* of π. A *permutation action* $\pi \cdot t$ is defined by induction: $\pi \cdot a \equiv \pi(a)$, $\pi \cdot [a]t \equiv [\pi(a)](\pi \cdot t)$, $\pi \cdot (\pi' \cdot X) \equiv (\pi \circ \pi') \cdot X$ and $\pi \cdot f(t_1, \ldots, t_n) \equiv f(\pi \cdot t_1, \ldots, \pi \cdot t_n)$. We write $(a\ b)$ for the *swapping* permutation that maps a to b, b to a and all other atoms c to themselves, and Id for the *identity permutation*, so $Id(a) = a$. Note that X is not a term, but $Id \cdot X$ is. We abbreviate $Id \cdot X$ as X when there is no ambiguity.

A *substitution* σ is a mapping from variables to terms, with a finite domain denoted by $dom(\sigma)$; the image is denoted $Im(\sigma)$. Henceforth, if $X \notin dom(\sigma)$ then $\sigma(X)$ denotes $Id \cdot X$. Substitutions are generated by the grammar: $\sigma := Id \,|\, \{X \mapsto s\}\sigma$, where Id denotes the substitution with $dom(Id) = \emptyset$. We use the same notation for the identity permutation and the identity substitution, as there will be no ambiguity. For every substitution σ, we define $\sigma|_V$ (the *restriction of σ to V*) as the substitution that maps X to $\sigma(X)$ if $X \in V$ and to $Id \cdot X$ otherwise. The *substitution action $t\sigma$* is defined as follows: $a\sigma \equiv a$, $([a]t)\sigma \equiv [a](t\sigma)$, $f(t_1, \ldots, t_n)\sigma \equiv f(t_1\sigma, \ldots, t_n\sigma)$ and $(\pi \cdot X)\sigma \equiv \pi \cdot \sigma(X)$. If σ and θ are substitutions, $\theta \circ \sigma$ is the substitution that maps each X to $(X\sigma)\theta$. Note that substitution allows capture of free atoms (it behaves like first-order substitution, except that when instantiating $\pi \cdot X$, π applies).

On nominal terms, α-equivalence is defined using swappings and a notion of freshness. A *freshness constraint* is a pair $a\#t$ (read "a fresh in t") of an atom a and a term t. Intuitively, $a\#t$ means that if a occurs in t then it must be abstracted. An *α-equality constraint* is a pair $s \approx_\alpha t$ of two terms s and t. A *freshness context* is a set of freshness constraints of the form $a\#X$. Δ, Γ and ∇ will range over freshness contexts. A *freshness judgement* is a tuple of the form $\Delta \vdash a\#t$ whereas an *α-equivalence judgement* is a tuple of the form $\Delta \vdash s \approx_\alpha t$. The *derivable* freshness and α-equivalence judgements are defined by the rules in Figure 1. A set Pr of constraints is called a *problem*. We write $\Delta \vdash Pr$ when proofs exist for each $P \in Pr$, using the derivation rules given in Figure 1. The minimal Δ such that $\Delta \vdash Pr$, denoted by $\langle Pr \rangle_{nf}$, can be obtained by using a system of simplification rules [11, 29], which, given Pr, outputs Δ or fails.

2.2 Unification, Matching and Nominal Rewriting

Unification is about finding a substitution that makes two terms equal. For nominal terms the notion of equality is $\approx_\alpha$, which is defined in a freshness context; nominal unification takes this into account.

▶ **Definition 1.** A *solution* for a problem Pr is a pair (Γ, σ) such that $\Gamma \vdash Pr\sigma$, where $Pr\sigma$ is the problem obtained by applying the substitution σ to the terms in Pr.

We follow [11], defining nominal matching/unification problems *in context*. A *term-in-context* is a pair $\Delta \vdash t$ of a freshness context and a term. We may write $\vdash t$ or simply t if $\Delta = \emptyset$.

$$\frac{}{\Delta \vdash a\#b}\ (\#\mathbf{ab}) \qquad \frac{}{\Delta \vdash a\#[a]t}\ (\#[\mathbf{a}]) \qquad \frac{(\pi^{-1}(a)\#X) \in \Delta}{\Delta \vdash a\#\pi \cdot X}\ (\#\mathbf{X})$$

$$\frac{\Delta \vdash a\#t}{\Delta \vdash a\#[b]t}\ (\#[\mathbf{b}]) \qquad \frac{\Delta \vdash a\#t_1 \ \cdots\ \Delta \vdash a\#t_n}{\Delta \vdash a\#\mathsf{f}(t_1, \dots, t_n)}\ (\#\mathsf{f}) \qquad \frac{}{\Delta \vdash a \approx_\alpha a}\ (\approx_\alpha \mathbf{a})$$

$$\frac{\Delta \vdash b\#t \quad \Delta \vdash (b\ a) \cdot t \approx_\alpha u}{\Delta \vdash [a]t \approx_\alpha [b]u}\ (\approx_\alpha [\mathbf{b}]) \qquad \frac{(a\#X \in \Delta \text{ for all } a \text{ s.t. } \pi(a) \neq \pi'(a))}{\Delta \vdash \pi \cdot X \approx_\alpha \pi' \cdot X}\ (\approx_\alpha \mathbf{X})$$

$$\frac{\Delta \vdash t \approx_\alpha u}{\Delta \vdash [a]t \approx_\alpha [a]u}\ (\approx_\alpha [\mathbf{a}]) \qquad \frac{\Delta \vdash t_i \approx_\alpha u_i \quad (1 \leq i \leq n)}{\Delta \vdash \mathsf{f}(t_1, \dots, t_n) \approx_\alpha \mathsf{f}(u_1, \dots, u_n)}\ (\approx_\alpha \mathsf{f})$$

Figure 1 Freshness and α-equality.

The action of substitutions extends to freshness contexts, instantiating the variables in freshness constraints.

▶ **Definition 2.** A *unification problem (in context)* is a pair $(\nabla \vdash l)\ {}_?{\approx}_?\ (\Delta \vdash s)$ where Δ, ∇ are freshness contexts and l, s are nominal terms. The *solution* to this unification problem, if it exists, is a pair (Δ', θ) that solves the problem $\Delta, \nabla, l \approx_\alpha s$, that is, $\Delta' \vdash \Delta\theta, \nabla\theta, l\theta \approx_\alpha s\theta$.

A *matching problem (in context)* is a particular kind of unification problem, written $(\nabla \vdash l)\ {}_?{\approx}\ (\Delta \vdash s)$,[1] where s is ground, or contains variables not occurring in ∇, l. The solution (Δ', θ) is such that $X\theta \equiv X$ for $X \in V(\Delta, s)$ (i.e., θ can only instantiate variables in ∇ and l, therefore, $\Delta' \vdash \Delta, \nabla\theta$ and $\Delta' \vdash l\theta \approx_\alpha s$).

▶ **Example 3.** $(\vdash [a][b]X')\ {}_?{\approx}\ (\vdash [b][a]X)$ has solution $(\emptyset, \{X' \mapsto (a\, b) \cdot X\})$.

▶ **Definition 4.** Let Γ_1, Γ_2 be contexts, and σ_1, σ_2 substitutions. Then $(\Gamma_1, \sigma_1) \leq (\Gamma_2, \sigma_2)$ if there exists some σ' such that: $\forall X, \Gamma_2 \vdash X\sigma_1\sigma' \approx_\alpha X\sigma_2\, and\, \Gamma_2 \vdash \Gamma_1\sigma'$. If we want to be more specific, we may write $(\Gamma_1, \sigma_1) \leq_{\sigma'} (\Gamma_2, \sigma_2)$. The relation $\leq$ is a partial order

Nominal unification is decidable and unitary [29]: a solvable problem has a unique least solution according to $\leq$, called *principal solution* or *most general unifier*, denoted by $mgu(Pr)$.

Below we recall the definitions of nominal equational reasoning [15] and nominal rewriting [11] from [12], where a *position* C is defined as a pair $(s, \_)$ of a term and a distinguished variable $\_ \in \mathcal{X}$ that occurs precisely once in s, with permutation Id. C is also called a *context*. When there is no ambiguity, we equate C with s and write $C[t]$ for the result of applying the substitution $\{\_ \mapsto t\}$ to s[2]. $\mathcal{P}os(u)$ denotes the set of positions of the nominal term u, that is, all the positions C such that $u = C[t]$ for some t. $\overline{\mathcal{P}os}(u) = \{C \in \mathcal{P}os(u) |\, u = C[t] \text{ and } t \neq \pi \cdot X\}$ is the set of non-variable positions.

An *equality judgement* (resp. *rewrite judgement*) is a tuple $\Delta \vdash s = t$ (resp. $\Delta \vdash s \to t$) of a freshness context Δ and two nominal terms s, t. An *equational theory* $\mathsf{E} = (\Sigma, Ax)$ is a pair of a signature Σ and a possibly infinite set of equality judgements Ax in Σ; they are called *axioms*. A *rewrite theory* $\mathsf{R} = (\Sigma, Rw)$ is a pair of a signature Σ and a possibly infinite set of rewrite judgements Rw in Σ; they are called *rewrite rules*. Σ may be omitted, identifying E

[1] The ${}_?{\approx}$ indicates that the variables being instantiated occur in the left-hand side term.

[2] This definition of position is equivalent to the standard notion of a position as a path in a tree; here we exploit the fact that nominal substitution corresponds to the informal notion of replacement of a 'hole' in a context by a term.

$$\boxed{\begin{array}{rlcll}
 & \vdash \; app(lam([a]X), X') & \to & sub([a]X, X') & (Beta)\\
 & \vdash \; sub([a]a, X) & \to & X & \\
a\#Y & \vdash \; sub([a]Y, X) & \to & Y & \\
 & \vdash \; sub([a]app(X, X'), Y) & \to & app(sub([a]X, Y), sub([a]X', Y)) & \\
b\#Y & \vdash \; sub([a]lam([b]X), Y) & \to & lam([b]sub([a]X, Y)) &
\end{array}}$$

Figure 2 λ-calculus with names and explicit substitutions [13].

with Ax and R with Rw when the signature is clear from the context. See Figure 2 for an example of a rewrite theory for the λ-calculus.

▶ **Definition 5.**

- *Nominal rewriting:* The *one-step rewrite relation* $\Delta \vdash s \overset{\mathsf{R}}{\to}_{[C,R,\theta,\pi]} t$ is the least relation such that for any $R = (\nabla \vdash l \to r) \in \mathsf{R}$, position C, term s', permutation π, and substitution θ,

  $$\frac{s \equiv C[s'] \quad \Delta \vdash \big(\nabla\theta, s' \approx_\alpha \pi \cdot (l\theta), C[\pi \cdot (r\theta)] \approx_\alpha t\big)}{\Delta \vdash s \overset{\mathsf{R}}{\to}_{[C,R,\theta,\pi]} t}$$

  We may omit subindices if they are clear from the context, writing simply $\Delta \vdash s \overset{\mathsf{R}}{\to} t$. The *rewrite relation* $\Delta \vdash_{\mathsf{R}} s \to t$ is the reflexive transitive closure of the one-step rewrite relation, that is, the least relation that includes the one-step rewrite relation and such that: for all Δ, s, s': $\Delta \vdash_{\mathsf{R}} s \to s'$ if $\Delta \vdash s \approx_\alpha s'$ (the native notion of equality of nominal terms is α-equality)[3]; for all Δ, s, t, u: $\Delta \vdash_{\mathsf{R}} s \to t$ and $\Delta \vdash_{\mathsf{R}} t \to u$ implies $\Delta \vdash_{\mathsf{R}} s \to u$. If $\Delta \vdash_{\mathsf{R}} s \to t$ holds, we say that s rewrites to t in the context Δ. A *normal form* is a term-in-context $\Delta \vdash s$ that does not rewrite, that is, there is no t such that $\Delta \vdash s \overset{\mathsf{R}}{\to} t$. A rewrite theory R is convergent if the rewrite relation is confluent and terminating.
- *(Nominal algebra) equality:* $\Delta \vdash_{\mathsf{E}} s = t$ is the least transitive reflexive symmetric relation such that for any $(\nabla \vdash l = r) \in \mathsf{E}$, position C, permutation π, substitution θ, and fresh Γ (so if $a\#X \in \Gamma$ then a is not mentioned in Δ, s, t),

  $$\frac{\Delta, \Gamma \vdash \big(\nabla\theta, \quad s \approx_\alpha C[\pi \cdot (l\theta)], \quad C[\pi \cdot (r\theta)] \approx_\alpha t\big)}{\Delta \vdash_{\mathsf{E}} s = t}.$$

  Given an equational theory E and a rewrite theory R, we say that R is a *presentation* of E if: $\nabla \vdash s = t \in \mathsf{E} \Leftrightarrow (\nabla \vdash s \to t \in \mathsf{R} \vee \nabla \vdash t \to s \in \mathsf{R})$.

Nominal rewriting is not complete for equational reasoning in general; however, *closed nominal rewriting* is complete for equational reasoning with *closed* axioms (see [12]). Intuitively, no free atom occurs in a closed term, and closed axioms do not allow abstracted atoms to become free (a natural assumption). Closedness of a term can be easily checked by matching the term with a freshened copy of itself. For example, the term $f(a)$ is not closed (it is not possible to match $f(a)$ with a freshened variant $f(a')$); however, $f([a]a)$ is closed ($f([a]a) \approx_\alpha f([a']a')$). If there are variables, freshness contexts have to be taken into account. We recall below the definitions of freshened variant, closed rewrite rule and closed rewriting relation from [12].

[3] As in the case of conditional rewriting modulo an equivalence theory (see [22]), reflexivity takes into account the underlying equivalence relation, here $\approx_\alpha$.

If t is a term, we say that $t^{\text{\reflectbox{N}}}$ is a *freshened variant* of t when $t^{\text{\reflectbox{N}}}$ has the same structure as t, except that the atoms and unknowns have been replaced by 'fresh' atoms and unknowns. Similarly, if ∇ is a freshness context then $\nabla^{\text{\reflectbox{N}}}$ will denote a freshened variant of ∇ (so if $a\#X \in \nabla$ then $a^{\text{\reflectbox{N}}}\#X^{\text{\reflectbox{N}}} \in \nabla^{\text{\reflectbox{N}}}$, where $a^{\text{\reflectbox{N}}}$ and $X^{\text{\reflectbox{N}}}$ are chosen fresh for the atoms and unknowns appearing in ∇). We may extend this to other syntax, like equality and rewrite judgements. For example, $[a^{\text{\reflectbox{N}}}][b^{\text{\reflectbox{N}}}]X^{\text{\reflectbox{N}}}$ is a freshened variant of $[a][b]X$, $a^{\text{\reflectbox{N}}}\#X^{\text{\reflectbox{N}}}$ is a freshened variant of $a\#X$, and $\emptyset \vdash f([a^{\text{\reflectbox{N}}}]X^{\text{\reflectbox{N}}}) \to [a^{\text{\reflectbox{N}}}]X^{\text{\reflectbox{N}}}$ is a freshened variant of $\emptyset \vdash f([a]X) \to [a]X$.

▶ **Definition 6** (Closed terms and rules, closed rewriting)**.** A term-in-context $\nabla \vdash l$ is *closed* if there exists a solution for the matching problem $(\nabla^{\text{\reflectbox{N}}} \vdash l^{\text{\reflectbox{N}}}) \;\; {}_?{\approx} \;\; (\nabla, A(\nabla^{\text{\reflectbox{N}}}, l^{\text{\reflectbox{N}}})\#V(\nabla, l) \vdash l)$[4]. Call $R = (\nabla \vdash l \to r)$ and $Ax = (\nabla \vdash l = r)$ *closed* when $\nabla \vdash (l, r)$ is closed[5]. Given a rewrite rule $R = (\nabla \vdash l \to r)$ and a term-in-context $\Delta \vdash s$, write $\Delta \vdash s {\to}^{c}_{R} t$ when there is some $R^{\text{\reflectbox{N}}}$ a freshened variant of R (so fresh for R, Δ, s, and t), position C and substitution θ such that $s \equiv C[s']$ and $\Delta, A(R^{\text{\reflectbox{N}}}) \# V(\Delta, s, t) \vdash (\nabla^{\text{\reflectbox{N}}}\theta,\ s' {\approx_\alpha} l^{\text{\reflectbox{N}}}\theta,\ C[r^{\text{\reflectbox{N}}}\theta] {\approx_\alpha} t)$. We call this (one-step) *closed rewriting*. The *closed-rewrite relation* $\Delta \vdash_{\mathsf{R}} s {\to}^{c} t$ is the reflexive transitive closure as in Definition 5.

All the rewrite rules in Figure 2 are closed. Closed rewriting is an efficient mechanism to generate rewriting steps for closed rules (closed-rewriting steps can be generated simply using nominal matching; it is not necessary to find a permutation π to apply a rule). We refer the reader to [11, 12] for examples.

3 Nominal E-Unification and Narrowing

We start by generalising the notion of solution.

▶ **Definition 7** (Nominal E-unification)**.** An E-solution, or E-unifier, of a problem Pr is a pair (Γ, σ) of a freshness context and a substitution such that

1. $\Gamma \vdash_{\mathsf{E}} Pr'\sigma$ where Pr' is obtained from Pr by replacing each $\approx_\alpha$ by $=$, and $\Gamma \vdash_{\mathsf{E}} a\#t$ coincides with $\Gamma \vdash a\#t$.
2. $X\sigma = X\sigma\sigma$ for all X (i.e., σ is *idempotent*).

If there is no such (Γ, σ) then Pr is *unsolvable*. $\mathcal{U}_{\mathsf{E}}(Pr)$ is the *set of* E*-solutions* of Pr.

The notion of E-unification extends to terms-in-context in the natural way.

▶ **Definition 8.** A *nominal* E*-unification problem (in context)* is a pair $(\nabla \vdash l) \, {}_?{\overset{\mathsf{E}}{\approx}}_? \, (\Delta \vdash s)$. The pair (Δ', σ) is an E-solution, or E-unifier, of $(\nabla \vdash l) \, {}_?{\overset{\mathsf{E}}{\approx}}_? \, (\Delta \vdash s)$ iff (Δ', σ) is an E-solution of the problem $\nabla, \Delta, l \approx_\alpha s$, that is, $\Delta' \vdash_{\mathsf{E}} \nabla\sigma, \Delta\sigma, l\sigma = s\sigma$.

$\mathcal{U}_{\mathsf{E}}(\nabla \vdash l, \Delta \vdash s)$ denotes the set of all the E-solutions of $(\nabla \vdash l) \, {}_?{\overset{\mathsf{E}}{\approx}}_? \, (\Delta \vdash s)$. If ∇ and Δ are empty we write $\mathcal{U}_{\mathsf{E}}(l, s)$ for the set of E-unifiers of l and s.

Nominal E*-matching problems in context* are defined similarly, except that s is a ground term (or, if it has variables, the solution cannot instantiate them). E-matching problems in context are written $(\nabla \vdash l) \, {}_?{\overset{\mathsf{E}}{\approx}} \, (\Delta \vdash s)$.

▶ **Definition 9.** The ordering $\leq_{\mathsf{E}}$ is the extension of $\leq$ with respect to E: $(\Gamma_1, \sigma_1) \leq_{\mathsf{E}} (\Gamma_2, \sigma_2)$ iff there exists a substitution ρ such that $\forall X,\ \ \Gamma_2 \vdash_{\mathsf{E}} X\sigma_2 = (X\sigma_1)\rho$ and $\Gamma_2 \vdash \Gamma_1\rho$. We write $\leq^{V}_{\mathsf{E}}$ for the restriction of $\leq_{\mathsf{E}}$ to the set V of variables.

[4] $A(\nabla^{\text{\reflectbox{N}}}, l^{\text{\reflectbox{N}}})\#V(\nabla, l) = \{a\#X \mid a \in A(\nabla^{\text{\reflectbox{N}}}, l^{\text{\reflectbox{N}}}), X \in V(\nabla, l)\}$.

[5] Here we use the pair constructor as a term former and apply the definition above.

▶ **Definition 10** (Complete set of E-solutions of Pr). Let W be a finite set of variables containing $V = V(Pr)$. We say that $\mathcal{S} = \{(\Gamma_1, \theta_1), \ldots, (\Gamma_n, \theta_n)\}$ is a *complete set of* E*-solutions of* Pr *away from* W iff

1. $\forall(\Gamma, \theta) \in \mathcal{S}$, $dom(\theta) \subseteq V$ and $Im(\theta) \cap W = \emptyset$,
2. $\mathcal{S} \subseteq \mathcal{U}_{\mathsf{E}}(Pr)$ (correctness),
3. $\forall(\Gamma, \sigma) \in \mathcal{U}_{\mathsf{E}}(Pr)\ \exists(\Gamma_i, \theta_i) \in \mathcal{S}$, $(\Gamma_i, \theta_i) \leq^V_{\mathsf{E}} (\Gamma, \sigma)$ (completeness).

We are now ready to define the *nominal narrowing relation* generated by R. The definition of nominal narrowing is similar to nominal rewriting, but we need to solve unification problems instead of matching problems.

▶ **Definition 11** (Nominal Narrowing). The *one-step narrowing relation* $(\Delta \vdash s) \rightsquigarrow_{[C,R,\theta,\pi]} (\Delta' \vdash t)$ is the least relation such that for any $R = (\nabla \vdash l \to r) \in \mathsf{R}$, position C, term s', permutation π, and substitution θ,

$$\frac{s \equiv C[s'] \quad \Delta' \vdash \left(\nabla\theta,\ \Delta\theta,\ s'\theta \approx_\alpha \pi \cdot (l\theta),\ (C[\pi \cdot r])\theta \approx_\alpha t\right)}{(\Delta \vdash s) \rightsquigarrow_{[C,R,\theta,\pi]} (\Delta' \vdash t)}\ (\Delta', \theta) = mgu(\nabla, \Delta, s' \approx_\alpha \pi \cdot l).$$

We may omit subindices if they are clear from the context.
The *narrowing relation* $(\Delta \vdash s) \rightsquigarrow_{\mathsf{R}} (\Delta' \vdash t)$ is the reflexive transitive closure of the one-step narrowing relation, that is, the least relation that includes the one-step narrowing relation and such that: for all Δ, s, s': $(\Delta \vdash s) \rightsquigarrow_{\mathsf{R}} (\Delta \vdash s')$ if $\Delta \vdash s \approx_\alpha s'$; for all Δ, Δ', Δ'', s, t, u: $(\Delta \vdash s) \rightsquigarrow_{\mathsf{R}} (\Delta' \vdash t)$ and $(\Delta' \vdash t) \rightsquigarrow_{\mathsf{R}} (\Delta'' \vdash u)$ implies $(\Delta \vdash s) \rightsquigarrow_{\mathsf{R}} (\Delta'' \vdash u)$.

The Lifting Theorem given below relates nominal narrowing and nominal rewriting. It is an extension of Hullot's Theorem 1 [17], taking into account freshness contexts and α-equivalence. The notions of normalised substitution-in-context and satisfiability of freshness contexts play a key role. A substitution σ is normalised in Δ w.r.t. a rewrite theory R if $\Delta \vdash X\sigma$ is a normal form in R for every X. A substitution σ satisfies the freshness context Δ if there exists a freshness context ∇ such that $\nabla \vdash a\#X\sigma$ for each $a\#X \in \Delta$; the minimal such ∇ is $\langle\Delta\sigma\rangle_{nf}$.

▶ **Theorem 12** (Lifting). *Let* $\mathsf{R} = \{\nabla_i \vdash l_i \to r_i\}$ *be a convergent rewrite theory. Let* $\Delta_0 \vdash s_0$ *be a nominal term-in-context and* V_0 *a finite set of variables containing* $V = V(\Delta_0, s_0)$*. Let* η *be a substitution with* $dom(\eta) \subseteq V_0$ *and satisfying* Δ_0*, that is, there exists* Δ *such that* $\Delta \vdash \Delta_0\eta$*. Assume moreover that* η *is normalised in* Δ*. Consider a rewrite derivation:*

$$\Delta \vdash s_0\eta = t_0 \to_{[C_0,R_0]} \cdots \to_{[C_{n-1},R_{n-1}]} t_n \qquad (*)$$

There exists an associated nominal narrowing derivation:

$$(\Delta_0 \vdash s_0) \rightsquigarrow_{[C'_0,R_0,\sigma_0]} \cdots \rightsquigarrow_{[C'_{n-1},R_{n-1},\sigma_{n-1}]} (\Delta_n \vdash s_n) \qquad (**)$$

for each i*,* $0 \leq i \leq n$*, a substitution* η_i *and a finite set of variables* $V_i \supseteq V(s_i)$ *such that:*

1. $dom(\eta_i) \subseteq V_i$*,*
2. η_i *is normalised in* Δ*,*
3. $\Delta \vdash \eta|_V \approx_\alpha \theta_i\eta_i|_V$*,*
4. $\Delta \vdash s_i\eta_i \approx_\alpha t_i$*,*
5. $\Delta \vdash \Delta_i\eta_i$

where $\theta_0 = Id$ *and* $\theta_{i+1} = \theta_i\sigma_i$*.*

Conversely, to each nominal narrowing derivation of the form $(**)$ *and every* η *such that* $(\Delta_n, \theta_n) \leq^V (\Delta, \eta)$ *and* $\Delta \vdash s_i\eta_i \approx_\alpha t_i$ *we can associate a nominal rewriting derivation of the form* $(*)$*.*

$$\begin{tikzcd}
\Delta \vdash s\eta = t_0 \arrow[r] & \ldots & t_i \arrow[r] & t_{i+1} \arrow[r] & \ldots \arrow[r] & t_n \\
\Delta_0 \vdash s_0 \arrow[u, dashed, "\eta_0"] \arrow[r, rightsquigarrow] & \ldots & (\Delta_i \vdash s_i) \arrow[u, dashed, "\eta_i"] \arrow[r, rightsquigarrow] & (\Delta_{i+1} \vdash s_{i+1}) \arrow[u, dashed, "\eta_{i+1}"] \arrow[r, rightsquigarrow] & \ldots \arrow[r, rightsquigarrow] & (\Delta_n \vdash s_n) \arrow[u, dashed, "\eta_n"]
\end{tikzcd}$$

Figure 3 Corresponding Rewriting and Narrowing Steps.

Proof.

($\Longrightarrow$) The proof is by induction on the length of the derivation. Figure 3 illustrates the relation between the two derivations.

Base Case. For $n = 0$, take $\eta_0 = \eta$, $V_0 = V \cup dom(\eta)$. By assumption, $\Delta \vdash \Delta_0\eta_0$.

$$\Delta_0 \vdash s_0 \dashrightarrow_{\eta_0} \Delta \vdash s_0\eta = t_0$$

Induction Step. Assume conditions (1)-(5) hold for i, and $\Delta \vdash t_i \to_{[C_i,R_i]} t_{i+1}$ (see Figure 3). We have:

(a) $R_i = \nabla_i \vdash l_i \to r_i \in \mathsf{R}$, $V(R_i) \cap V(\Delta, t_i) = \emptyset$,
(b) $t_i \equiv C_i[t'_i]$ for some position $C_i[\_\,]$ and $\Delta \vdash \nabla_i\sigma, \pi \cdot (l_i\sigma) \approx_\alpha t'_i$.
(c) $\Delta \vdash C_i[\pi \cdot (r_i\sigma)] \approx_\alpha t_{i+1}$

Also, $dom(\sigma) \cap V_i = \emptyset$ since $V(R_i) \cap V(\Delta, t_i) = \emptyset$.

By IH, it follows from assumptions 2., 4. and 5. that $s_i \equiv C'_i[s'_i]$ where $C'_i[\_\,] \in \overline{\mathcal{P}os}(s_i)$ and $\Delta \vdash s'_i\eta_i \approx_\alpha t'_i \approx_\alpha \pi \cdot (l_i\sigma)$ (if $C'_i[\_\,]$ were a variable position the term s'_i would be a variable, from (4), $\Delta \vdash s'_i\eta_i \approx_\alpha t'_i \approx_\alpha \pi \cdot (l_i\sigma) \to_{R_i} \pi \cdot (r_i\sigma)$, contradicting that η_i is a normalised substitution).

Let us consider $\rho = \eta_i \cup \sigma$, we have $\Delta \vdash s'_i\rho \approx_\alpha \pi \cdot (l_i\rho)$. The pair (Δ, ρ) is a solution for $(\Delta_i \vdash s'_i)\ {}_?{\approx}_?\ (\nabla_i \vdash l_i)$:

(i) $\Delta \vdash \Delta_i\rho$, because, by hypothesis, $\Delta \vdash \Delta_i\eta_i$ and σ does not affect Δ_i ($dom(\sigma) \subseteq V(R_i)$).
(ii) $\Delta \vdash \nabla_i\rho$.
(iii) $\Delta \vdash s'_i\rho \approx_\alpha \pi \cdot (l_i\rho)$.

Now, take the principal solution (Δ_{i+1}, σ_i) of $(\Delta_i \vdash s'_i)\ {}_?{\approx}_?\ (\nabla_i \vdash \pi \cdot l_i)$. Then, $\Delta_{i+1} \vdash \Delta_i\sigma_i, \nabla_i\sigma_i, s'_i\sigma \approx_\alpha \pi \cdot (l_i\sigma_i)$. Let s_{i+1} be a nominal term such that $\Delta_{i+1} \vdash C'_i[\pi \cdot r_i]\sigma_i \approx_\alpha s_{i+1}$. Therefore, $(\Delta_i \vdash s_i) \rightsquigarrow_{[C_i,R_i,\sigma_i]} (\Delta_{i+1} \vdash s_{i+1})$.

Since (Δ_{i+1}, σ_i) is the least unifier of $(\Delta_i \vdash s'_i)\ {}_?{\approx}_?\ (\nabla_i \vdash \pi \cdot l_i)$, $(\Delta_{i+1}, \sigma_i) \leq (\Delta, \rho)$ and thus there exists a substitution η' such that $\mathit{for\ all}\ X,\ \Delta \vdash X\sigma_i\eta' \approx_\alpha X\rho\ \mathit{and}\ \Delta \vdash \Delta_{i+1}\eta'$. That is, $\Delta \vdash \sigma_i\eta' \approx_\alpha \rho$. Since $\rho = \eta_i \cup \sigma$ and $dom(\sigma) \cap V_i = \emptyset$, η_i is such that $\Delta \vdash \eta_i \approx_\alpha \sigma_i\eta'|_{V_i}$.

Now let $V_{i+1} = (V_i \cup Im(\sigma_i)) - dom(\sigma_i)$ and let η_{i+1} be such that $\Delta \vdash \eta_{i+1} \approx_\alpha \eta'|_{V_{i+1}}$. We get condition 1., that is, $dom(\eta_{i+1}) \subseteq V_{i+1}$ and from 3.: $\Delta \vdash \eta_i \approx_\alpha (\sigma_i\eta_{i+1})|_{V_i}$ (1).

(By hypothesis, $\Delta \vdash \eta|_V \approx_\alpha \theta_i\eta_i$. To illustrate, take $i = 4$, then $\eta|_V = \theta_4\eta_4 = \theta_5\eta_5$. Using the definition of θ_i, it follows that $\theta_4 = \sigma_0\sigma_1 \ldots \sigma_3$ and $\theta_5 = \sigma_0\sigma_1 \ldots \sigma_4$. Thus, $\theta_4\eta_4 = \sigma_0\sigma_1 \ldots \sigma_3\eta_4 = \sigma_0\sigma_1 \ldots \sigma_3\sigma_4\eta_5$ and $\eta_4 = \sigma_4\eta_5$.) Recall that we impose $dom(\sigma_i) \cap Im(\sigma_i) = \emptyset$.

To prove 5. for $i+1$, notice that from $\Delta \vdash \Delta_{i+1}\eta'$ it follows that $\Delta \vdash \Delta_{i+1}\eta_{i+1}$, since $\Delta \vdash \eta_{i+1} \approx_\alpha \eta'$.

To prove 2. for $i+1$, let us consider $X \in V_{i+1}$. There are two cases:

(i') $X \in V_i - dom(\sigma_i)$ then $\Delta \vdash X\eta_i \approx_\alpha X\sigma_i\eta' \approx_\alpha X\eta' \approx_\alpha X\eta_{i+1}$. Since η_i is a normalised substitution, by hypothesis, it follows that η_{i+1} is also a normalised substitution.

(ii') $X \in V(Im(\sigma_i))$, then there exists $Y \in dom(\sigma_i)$ such that $X \in V(Y\sigma_i)$. Then, $X\eta_{i+1}$ is a subterm of $Y\eta_i$ since $\Delta \vdash X\eta_{i+1} \approx_\alpha X\eta', Y\sigma_i\eta' \approx_\alpha Y\eta_i$, and since, by hypothesis, η_i is a normalised substitution, it follows that η_{i+1} is also normalised.

This proves (2) for $i+1$.

We now prove 3. for $i+1$, assuming it for i, i.e., $\Delta \vdash \eta|_V \approx_\alpha \theta_i\eta_i|_V$.

From equation (1) we get $\Delta \vdash \theta_i\eta_i|_V \approx_\alpha \theta_i(\sigma_i\eta_{i+1}|_{V_i})|_V$ From the definition of θ_i we have $Im(\theta_i) \subseteq V_i$ and $V_0 \subseteq V_i \cup dom(\theta_i)$. Therefore, $\Delta \vdash \underbrace{\theta_i\sigma_i}_{\theta_{i+1}}\eta_{i+1}|_V \approx_\alpha \theta_i\eta_i|_V \approx_\alpha \eta|_V$ proving condition 3) for $i+1$. Notice that, by 3), θ_i is normalised.

Finally, on the one hand $\Delta \vdash t_{i+1} \approx_\alpha C_i[\pi \cdot r_i\sigma] \approx_\alpha C_i[\pi \cdot r_i\rho] \approx_\alpha C_i[\pi \cdot r_i(\sigma_i\eta')] \approx_\alpha C_i[\pi \cdot r_i\eta_i]$. On the other hand, $\Delta \vdash s_{i+1}\eta_{i+1} \approx_\alpha (C'_i[\pi \cdot r_i]\sigma_i)\eta_{i+1} \approx_\alpha (C'_i[\pi \cdot r_i]\sigma_i)\eta' \approx_\alpha (C'_i(\sigma_i\eta'))[\pi \cdot r_i(\sigma_i\eta')] \approx_\alpha (C_i\eta_i)[\pi \cdot r_i\eta_i] \approx_\alpha C_i[\pi \cdot r_i\eta_i]$. Therefore, $\Delta \vdash s_{i+1}\eta_{i+1} \approx_\alpha t_{i+1}$, proving (4).

($\Longleftarrow$) Conversely, let us consider a derivation (**): $(\Delta_0 \vdash s_0) \rightsquigarrow_{[C'_0, R_0, \sigma_0]} \cdots \rightsquigarrow_{[C'_{n-1}, R_{n-1}, \sigma_{n-1}]} (\Delta_n \vdash s_n)$, and a substitution η such that $(\Delta_n, \theta_n) \leq^V (\Delta, \eta)$, that is, there exists ρ such that $\Delta \vdash X\eta|_V \approx_\alpha (X\theta_n)\rho|_V$ and $\Delta \vdash \Delta_n\rho$. We define substitutions η_i for $0 \leq i \leq n-1$ by: $\Delta \vdash \eta_i \approx_\alpha \sigma_i \ldots \sigma_{n-1}\rho$ (2). and a normalised substitution $\eta_n \equiv \rho$. By hypothesis, $\Delta \vdash \Delta_n\rho$, and by definition of narrowing step, it follows that $\Delta_{i+1} \vdash \Delta_i\sigma_i$ $(0 \leq i \leq n-1)$. Hence $\Delta \vdash \Delta_i\eta_i$, and in particular $\Delta \vdash \Delta_0\eta$. We define $s_i\eta_i \equiv t_i$ for $0 \leq i \leq n$, and show, by induction on i, that: $\Delta \vdash s_0\eta = t_0 \rightarrow_{[C_0, R_0]} \cdots \rightarrow_{[C_{n-1}, R_{n-1}]} t_n$.

Base Case. When $i = 0$: $\Delta \vdash s_0\eta_0 \approx_\alpha s(\theta_n\eta_n) \approx_\alpha s\eta$. By definition, $\eta_0 = \underbrace{\sigma_0\sigma_1 \ldots \sigma_{n-1}}_{\theta_n}\rho$.

Induction Step. Suppose that $(\Delta_i \vdash s_i) \rightsquigarrow_{[C'_i, R_i, \sigma_i]} (\Delta_{i+1} \vdash s_{i+1})$. By the definition of nominal narrowing we have

- $R_i = \nabla_i \vdash l_i \rightarrow r_i \in \mathsf{R}$, $V(R_i) \cap V(\Delta_i, s_i) = \emptyset$.
- $s_i \equiv C'_i[s'_i]$, for a non-variable position $C'_i[\_]$ of s_i, and such that (Δ_{i+1}, σ_i) is the least solution for $(\Delta_i \vdash s'_i)\ {}_?{\approx}_?\ (\nabla_i \vdash \pi \cdot l_i)$. That is, $\Delta_{i+1} \vdash s'_i\sigma_i \approx_\alpha \pi \cdot (l_i\sigma_i)$ and $\Delta_{i+1} \vdash \Delta_i\sigma_i, \nabla_i\sigma_i$.
- $\Delta_{i+1} \vdash C'_i[\pi \cdot r_i]\sigma_i \approx_\alpha s_{i+1}$.

By definition, $\Delta \vdash s_i\eta_i \approx_\alpha t_i$. Since $C_i[\_]$ is a non-variable position and η_i is a normalised substitution, we have that $\Delta \vdash s'_i\eta_i \approx_\alpha t'_i$. In addition, define $\eta' \equiv \sigma_{i+1} \ldots \sigma_{n-1}\rho$, by equation (2) $\Delta \vdash \eta_{i+1} \approx_\alpha \eta'|_{V_{i+1}}$. $\Delta \vdash t'_i \approx_\alpha s'_i\eta_i \approx_\alpha s'_i(\sigma_i\eta') \approx_\alpha (\pi \cdot l_i\sigma_i)\eta' \rightarrow_{R_i} (\pi \cdot r_i\sigma_i)\eta'$ Therefore, $\Delta \vdash t_i \equiv C_i[t'_i] \rightarrow_{R_i} C_i[\pi \cdot r_i\sigma_i\eta'] \approx_\alpha s_{i+1}\eta_{i+1} \approx_\alpha t_{i+1}$. ◀

In a similar way, we can associate closed nominal rewriting derivations (see Definition 6) with *closed nominal narrowing* derivations, where closed narrowing is defined as follows.

▶ **Definition 13** (Closed narrowing). Given a rewrite rule $R = (\nabla \vdash l \rightarrow r)$ and a term-in-context $\Delta \vdash s$, write $(\Delta \vdash s) \rightsquigarrow^c_R (\Delta' \vdash t)$ when there is some R^{N} a freshened variant of R (so fresh for R, Δ, s, and t), position C and substitution θ such that $s \equiv C[s']$ and $\Delta', A(R^{\mathsf{N}}) \# V(\Delta, s, t) \vdash (\nabla^{\mathsf{N}}\theta, \Delta\theta,\ s'\theta {\approx_\alpha} l^{\mathsf{N}}\theta,\ (C[r^{\mathsf{N}}])\theta {\approx_\alpha} t)$. We call this (one-step) *closed narrowing*. The *closed narrowing relation* $\Delta \vdash_{\mathsf{R}} s \rightsquigarrow^c \Delta' \vdash_{\mathsf{R}} t$ is the reflexive transitive closure as in Definition 5.

See Example 17 in Section 3.1 for examples of closed narrowing steps.

▶ Remark. We can state a "closed lifting" theorem by replacing nominal rewriting/narrowing for closed rewriting/narrowing. The proof is similar.

In the following we consider a *closed* nominal equational theory E, presented by a convergent set R of closed rules.

Let us consider an E-unification problem $(\Delta \vdash s)\,{}_{?}\overset{\mathsf{E}}{\approx}_{?}\,(\nabla \vdash t)$. To find a solution, we will apply *closed narrowing* on $\Delta \vdash s$ and $\nabla \vdash t$ in parallel. It will simplify matters to narrow the single term $u = (s,t)$[6] under Δ, ∇.

▶ **Lemma 14** (Soundness). *Let $\Delta \vdash s$ and $\nabla \vdash t$ be two nominal terms-in-context and $\Delta, \nabla \vdash (s,t) = u_0 \rightsquigarrow^c \ldots \rightsquigarrow^c \Delta_n \vdash u_n = (s_n, t_n)$ a closed narrowing derivation such that $\Delta_n, s_n \approx_\alpha t_n$ has a solution, say (Γ, σ). Then $(\Gamma, \theta_n\sigma)$ is an E-solution of the problem $\Delta, \nabla, s\,{}_{?}\overset{\mathsf{E}}{\approx}_{?}\,t$, where θ_n is the composition of substitutions along the narrowing derivation, as defined in Theorem 12.*

Proof. Using the $(\Leftarrow)$ part of the previous theorem with $\eta = \theta_n$, we can associate this narrowing derivation with the following rewriting derivation:
$\Gamma \vdash u_0\theta_n = v_0 \rightarrow^c v_1 \rightarrow^c v_2 \rightarrow^c \ldots \rightarrow^c v_n = (v_n^s, v_n^t)$. Thus, $\Gamma \vdash_{\mathsf{R}} s\theta_n \rightarrow^c v_n^s$ and $\Gamma \vdash_{\mathsf{R}} t\theta_n \rightarrow^c v_n^t$. Moreover, since $\eta_n = Id$ (because $\eta = \eta_n\theta_n$) it follows that $\Gamma \vdash v_n^s \approx_\alpha s_n$ and $\Gamma \vdash v_n^t \approx_\alpha t_n$, thus: $\Gamma \vdash_{\mathsf{E}} s\theta_n\sigma = t\theta_n\sigma$ and therefore, $(\Gamma, \theta_n\sigma)$ is an E-solution for $\Delta, \nabla, s\,{}_{?}\overset{\mathsf{E}}{\approx}_{?}\,t$. ◀

▶ **Lemma 15** (Completeness). *Let $\Delta \vdash s$ and $\nabla \vdash t$ be two nominal terms-in-context, such that the problem $(\Delta \vdash s)\ {}_{?}\overset{\mathsf{E}}{\approx}_{?}\ (\nabla \vdash t)$ has an E-solution, (Δ', ρ), and let V be a finite set of variables containing $V(\Delta, \nabla, s, t)$. Then there exists a closed narrowing derivation: $\nabla, \Delta \vdash u = (s,t) \rightsquigarrow^c \ldots \rightsquigarrow^c \Gamma_n \vdash (s_n, t_n)$, such that $\Gamma_n, s_n \approx_\alpha t_n$ has a solution. Let $(\Gamma, \mu) = mgu(\Gamma_n, s_n \approx_\alpha t_n)$, and θ_n the composition of the narrowing substitutions. Then, $(\Gamma, \theta_n\mu) \leq_{\mathsf{E}}^{V} (\Delta', \rho)$. Moreover, we are allowed to restrict our attention to $\rightsquigarrow^c$-derivations such that: $\forall i, 0 \leq i \leq n$, $\theta_i|_V$ is normalised.*

Proof. By Definition 8, $\Delta' \vdash_{\mathsf{E}} s\rho = t\rho, \nabla\rho, \Delta\rho$. Take $\eta = \rho\downarrow$, that is, ρ's normal form in Δ': $\Delta' \vdash X\eta \approx_\alpha (X\rho)\downarrow$. It follows that $\Delta' \vdash_{\mathsf{E}} s\eta = t\eta, \nabla\eta, \Delta\eta$ since the rules are closed.

Since E is a closed nominal theory presented by a convergent rewrite system R, and since closed rewriting is complete for equational reasoning in this case, $s\eta$ and $t\eta$ have the same normal form in Δ', which we will call r. Then, $\Delta' \vdash u\eta = (s\eta, t\eta) = t'_0 \rightarrow^c \ldots \rightarrow^c t'_n = (r,r)$. By Theorem 12 there exists a corresponding $\rightsquigarrow$-derivation ending with $\Gamma_n \vdash (s_n, t_n)$ such that: $\Delta' \vdash (s_n\eta_n, t_n\eta_n) \approx_\alpha t'_n = (r,r)$ and $\Delta' \vdash \Gamma_n\eta_n$. Thus, (Δ', η_n) is a solution of $\Gamma_n, s_n \approx_\alpha t_n$.

Since (Γ, μ) is the least unifier, it follows that $(\Gamma, \mu) \leq (\Delta', \eta_n)$ and: $\exists \xi : \forall X,\ \Delta' \vdash X\mu\xi \approx_\alpha X\eta_n$ and $\Delta' \vdash \Gamma\xi$. Therefore, by Theorem 12, $\Delta' \vdash (\theta_n\mu\xi)|_V \approx_\alpha \theta_n\eta_n|_V \approx_\alpha \eta|_V$ and $\Delta' \vdash_{\mathsf{E}} \eta|_V = \rho|_V$ that is, $(\Gamma, \theta_n\mu) \leq_{\mathsf{E}}^{V} (\Delta', \rho)$. ◀

Now we can describe how to build a complete set of E-unifiers for two terms-in-context.

▶ **Theorem 16.** *Let E be a closed nominal equational theory and R be an equivalent convergent nominal rewrite theory. Let $\Delta \vdash s$ and $\nabla \vdash t$ be two terms-in-context, and V be a finite set of variables containing $V(\Delta, s, \nabla, t)$. Let $\mathcal{S}$ be the set of pairs (Γ, σ) such that there exists a $\rightsquigarrow^c$-derivation: $\Gamma_0 \vdash u = (s,t) = u_0 \rightsquigarrow^c \ldots \rightsquigarrow^c \Gamma_n \vdash u_n = (s_n, t_n)$, where $(\Gamma_0 \equiv \Delta, \nabla)$, $\Gamma_n, s_n \approx_\alpha t_n$ has a least solution (Γ, μ), $\sigma \equiv \theta_n\mu$, and θ_n is the normalised composition of the narrowing substitutions. Then $\mathcal{S}$ is a complete set of E-unifiers of $\Delta \vdash s$ and $\nabla \vdash t$ away from V.*

Proof. Consequence of Lemmas 14 and 15. ◀

[6] Here we use the pair constructor as a term former.

$$\begin{array}{rll}
y\#F & \vdash & \mathit{diff}(lam([y]F), X) \to 0 \\
 & \vdash & \mathit{diff}(lam([y]y), X) \to 1 \\
 & \vdash & \mathit{diff}(lam([y]sin(F)), X) \to mult(cos(sub([y]F, X)), \mathit{diff}(lam([y]F), X)) \\
 & \vdash & \mathit{diff}(lam([y]plus(F, G)), X) \to plus(\mathit{diff}(lam([y]F), X), \mathit{diff}(lam([y]G), X)) \\
 & \vdash & \mathit{diff}(lam([y]mult(F, G)), X) \to plus(mult(\mathit{diff}(lam([y]F), X), sub([y]G, X)), \\
 & & \qquad\qquad\qquad\qquad mult(\mathit{diff}(lam([y]G), X), sub([y]F, X)))
\end{array}$$

Figure 4 Rewrite rules for symbolic differentiation.

An E-unification procedure follows from the construction of Theorem 16: enumerate all elements of $\mathcal{S}$. The set $\mathcal{S}$ may be infinite, one can organise the enumeration in such a way that if two nominal terms $\Delta \vdash s$ and $\nabla \vdash t$ are E-unifiable, then an E-solution will be produced in a finite number of steps. Thus, assuming E is presented by a convergent rewrite theory R, we have a semi-decision procedure for nominal E-unification.

3.1 An example: symbolic differentiation

The rewrite rules in Figure 2 define a λ-calculus with names and explicit substitutions [13]; the extension with numbers and operations (*plus*, *mult*, *sin*, *cos*) is straightforward.

Consider now symbolic differentiation [26]: $\mathit{diff}(F, X)$ computes the differential of a function F (meta-level unknown that can be instantiated by a λ-term) at a point X, using the rewrite rules given in Figure 4.

▶ **Example 17.** Let E be the theory defined by rewrite rules in Figures 2 and 4 together with standard rules for arithmetic operations. This system is closed but not convergent (we can simulate the untyped λ-calculus, which is non-terminating) so narrowing is not necessarily complete; however, we can still obtain the E-solution $(\emptyset, \{F \mapsto y\})$ for the nominal E-unification problem $lam([z]\mathit{diff}(lam([y]sin(F)), z)) \; {}_?\overset{\mathsf{E}}{\approx}_? \; lam([z]cos(z))$ as follows[7].

The first closed-narrowing step uses a freshened rule
$\vdash \mathit{diff}(lam([y']sin(F')), X') \to mult(cos(sub([y']F', X')), \mathit{diff}(lam([y']F'), X'))$
with the assumption $y'\#F$ (below the narrowed subterm is in bold, the substitution used is $\{F' \mapsto (y\ y') \cdot F, X' \mapsto z\}$):

$$\begin{array}{l}
lam([z]\mathit{diff}(\mathbf{lam}([\mathbf{y}]\mathbf{sin}(\mathbf{F})), \mathbf{z})) \; {}_?\approx_? \; lam([z]cos(z)) \\
\rightsquigarrow lam([z]mult(cos(sub([y'](y\ y') \cdot F, z)), \mathit{diff}(lam([y'](y\ y') \cdot F), z))) \; {}_?\approx_? \; lam([z]cos(z))
\end{array}$$

We now use the freshened rule $\vdash \mathit{diff}(lam([w]w), W) \to 1$ with substitution $\{F \mapsto y, W \mapsto z\}$ and assumption $w\#F$ to narrow the second argument of *mult*:

$$\rightsquigarrow lam([z]mult(cos(sub([y']y', z)), 1)) \; {}_?\approx_? \; lam([z]cos(z))$$

Using now the rules for *sub*, we can rewrite (hence also narrow) to

$$lam([z]mult(cos(z), 1)) \; {}_?\approx_? \; lam([z]cos(z))$$

and by rewriting with the usual rules for multiplication, we obtain two equal terms.

[7] Here we do not rely on *Beta*, *diff* uses just the substitution rules, which are terminating.

$$\begin{array}{llcrll}
(1) & \emptyset & \vdash & \pi_i(\langle X_1, X_2\rangle) & \to X_i & (i \in \{1,2\}) \\
(2) & \emptyset & \vdash & d(\{X\}_Y, Y^{-1}) & \to X & \\
(3) & \emptyset & \vdash & d(\{X\}_{Y^{-1}}, Y) & \to X & \\
(4) & \emptyset & \vdash & (X^{-1})^{-1} & \to X & \\
(5) & \emptyset & \vdash & subst_j^n([\vec{z}]z_k, \vec{X}) & \to X_k & (1 \le k \le j) \\
(6) & z\#Y & \vdash & subst_1^n([z]Y, X) & \to Y & \\
(7) & z_k\#Y & \vdash & subst_j^n(\overrightarrow{[z]}Y, \vec{X}) & \to subst_{j-1}^n(\overrightarrow{[z']}Y, \overrightarrow{X'}) & (1 \le k \le j, j > 1) \\
(8) & \emptyset & \vdash & subst_j^n(\overrightarrow{[z]}f(\overrightarrow{W}), \vec{X}) & \to f(\overrightarrow{subst_j^n(\overrightarrow{[z]}W, \vec{X})}) &
\end{array}$$

Figure 5 Rewrite theory DYT.

3.2 Application: Intruder Deduction Problem

In this section we present an application of nominal E-matching.

▶ **Definition 18** (Nominal Intruder Deduction Problem). Given a finite set of ground messages in normal form $\Gamma = \{t_1, \dots, t_n\}$, a ground message in normal form m (the secret), and private names $a_1, \dots, a_k$, we model the Intruder Deduction Problem (IDP) as a nominal E-matching problem with one unknown: $(\Delta \vdash subst([\vec{z}]X, \vec{t}))\ {}_?\overset{\mathsf{E}}{\approx} m$. Here m is short for $\emptyset \vdash m$ and $\Delta = \{a_1\#X, \dots, a_k\#X\}$ is a freshness context specifying that the names $a_1, \dots, a_n$ are fresh in the unknown term X, $subst$ is a term-former denoting the substitution of $z_1, \dots, z_n$ (denoted by $\vec{z}$) by $t_1, \dots, t_n$ (denoted by $\vec{t}$), $\vec{z}$ are abstracted in X, and $\vec{t}$ represent the messages in Γ.

To illustrate the results we consider a simple equational theory, namely the *Axiomatised Dolev-Yao Theory* (DYT). It is essentially the classical Dolev-Yao model with explicit destructors such as decryption and projections. It is well-known that IDP for this theory is decidable in polynomial time[8], the purpose here is to show how nominal narrowing could be used to solve this security problem.

The signature for DYT, Σ_{DYT}, includes function symbols $\langle \_, \_ \rangle, \pi_1(\_)$, $\pi_2(\_), d(\_, \_)$, $\{\_\}_\_, (\_)^{-1}$ for *pairing*, *projections*, *decryption*, *encryption* and *inverse*, respectively, as well as a family of symbols $subst_j^n$ $(n \ge 1, j \in \{1, \dots, n\})$ to perform substitution. Intuitively, projections are inverses of pairing and decrypting with k^{-1} a message encrypted with k gives back the plaintext.

The rewrite rules are given, in a schematic way, in Figure 5. The index j in $subst_j^n$ denotes the number of abstracted atoms in $[\vec{z}]$, for $j \in \{1, \dots, n\}$. In rule schemes (5) and (7), z_k is a term in $\{z_1, \dots, z_j\}$ and there is a rule for each k s.t. $1 \le k \le j$. In rule scheme (7), $j > 1$; in case $j = 1$ we use rule (6). In rules (7) and (8) we use the following abbreviations:

- $\overrightarrow{[z]} = [z_1, \dots, z_j]$ and $\overrightarrow{[z']} = [z_1 \dots, z_{k-1}, z_{k+1}, \dots, z_j]$;
- $\vec{X} = (X_1, \dots, X_j)$ and $\overrightarrow{X'} = (X_1, \dots, X_{k-1}, X_{k+1}, \dots, X_{j-1})$;
- $f \in \Sigma_{\mathsf{DYT}}$ is an r-ary function symbol (there is a version of rule (8) for each $f \neq subst$), and $f(\overrightarrow{subst_j^n(\overrightarrow{[z]}W, \vec{X})}) = f(subst_j^n(\overrightarrow{[z]}W_1, \vec{X}), \dots, subst_j^n(\overrightarrow{[z]}W_r, \vec{X}))$.

▶ **Proposition 19.** DYT *is a closed and convergent nominal rewrite system.*

Proof. The termination is obtained by a simplification ordering. It is convergent because the critical pairs obtained are joinable [11]. ◀

[8] This result was obtained using another approach [7]

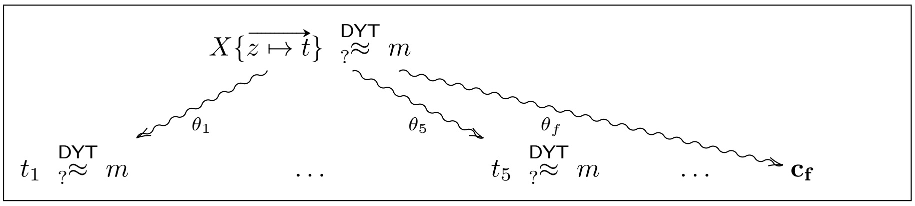

$X\{\overrightarrow{z \mapsto t}\} \overset{\mathsf{DYT}}{{}_?\approx} m$
θ_1
θ_5
θ_f
$t_1 \overset{\mathsf{DYT}}{{}_?\approx} m$
$\dots$
$t_5 \overset{\mathsf{DYT}}{{}_?\approx} m$
$\dots$
$\mathbf{C_f}$

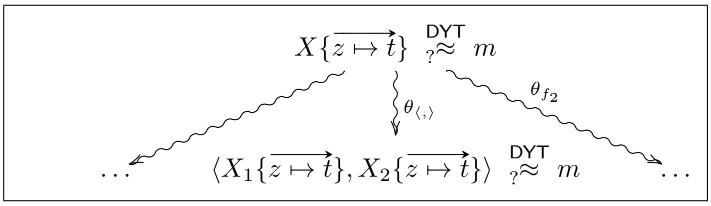
Figure 6 First level of the narrowing tree.

▶ Remark. Below, the notation $t\{z \mapsto t'\}$ is syntactic sugar for $subst([z]t, t')$.

▶ **Example 20.** Consider $\Gamma = \{\underbrace{\{\{m\}_b\}_c}_{t_1}, \underbrace{\{b^{-1}\}_k}_{t_2}, \underbrace{\{c^{-1}\}_r}_{t_3}, \underbrace{k^{-1}}_{t_4}, \underbrace{r^{-1}}_{t_5}\}$ and a secret m (a constant). Taking into account the theory DYT, this IDP can be stated as $X\{\overrightarrow{z \mapsto t}\} \overset{\mathsf{DYT}}{{}_?\approx} m$, where $\{\overrightarrow{z \mapsto t}\}$ denotes the substitution of t_i for z_i, $i = 1, \dots, 5$. Figure 6 shows part of the first level of the narrowing tree for this problem.

The substitutions θ_i are $\{X \mapsto z_i\}$, $i = 1, \dots, 5$ and the corresponding narrowing steps use rule (5). The result $t_i \overset{\mathsf{DYT}}{{}_?\approx} m$ is a ground problem, which can be decided by checking syntactic equality since each t_i and m are in normal form. The branch labelled with the substitution θ_f is an abbreviation for six branches, namely, one for each $f \in \Sigma_{\mathsf{DYT}}$ (except $subst$).

To illustrate, consider the case in which f is a constructor, for instance, $f = \langle\, , \,\rangle$:


$X\{\overrightarrow{z \mapsto t}\} \overset{\mathsf{DYT}}{{}_?\approx} m$
$\theta_{\langle,\rangle}$
θ_{f_2}
$\dots$
$\langle X_1\{\overrightarrow{z \mapsto t}\}, X_2\{\overrightarrow{z \mapsto t}\}\rangle \overset{\mathsf{DYT}}{{}_?\approx} m$
$\dots$

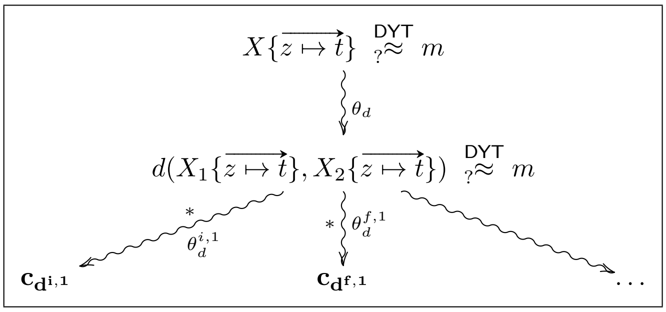
This branch is obtained via

- a version of rule (8): $\emptyset \vdash subst^5_j([\overrightarrow{w}]\langle W_1, W_2\rangle, \overrightarrow{Z}) \to \langle subst^5_j([\overrightarrow{w}]W_1, \overrightarrow{Z}), subst^5_j([\overrightarrow{w}]W_2, \overrightarrow{Z})\rangle$
- and substitution $\theta_{\langle,\rangle} = \{X \mapsto \langle X_1, X_2\rangle, W_1 \mapsto (w\ z) \cdot X_1, W_2 \mapsto (w\ z) \cdot X_2, \overrightarrow{Z} \mapsto \overrightarrow{T}\}$ with the assumption $w \# X$.

Consider the case in which f is a destructor, for instance, $f = d$. There is a narrowing step:

$$X\{\overrightarrow{z \mapsto t}\} \overset{\mathsf{DYT}}{{}_?\approx} m \rightsquigarrow_{\theta_d} d(X_1\{\overrightarrow{z \mapsto t}\}, X_2\{\overrightarrow{z \mapsto t}\}) \overset{\mathsf{DYT}}{{}_?\approx} m$$

obtained via substitution $\theta_d = \{X \mapsto d(X_1, X_2)\}$ and rule (8). From this node we can narrow with $\theta_d^{i,1} = \{X_1 \mapsto z_i\}$, or $\theta_d^{i,2} = \{X_2 \mapsto z_i\}$ $(i = 1, \dots, 5)$, or $\theta_d^{f,1} = \{X_1 \mapsto f(X_1')\}$ or $\theta_d^{f,2} = \{X_2 \mapsto f(X_2')\}$ $(f \in \Sigma_{\mathsf{DYT}})$:


$X\{\overrightarrow{z \mapsto t}\} \overset{\mathsf{DYT}}{{}_?\approx} m$
θ_d
$d(X_1\{\overrightarrow{z \mapsto t}\}, X_2\{\overrightarrow{z \mapsto t}\}) \overset{\mathsf{DYT}}{{}_?\approx} m$
$*$
$\theta_d^{i,1}$
$*$
$\theta_d^{f,1}$
$\mathbf{C_{d^{i,1}}}$
$\mathbf{C_{d^{f,1}}}$
$\dots$

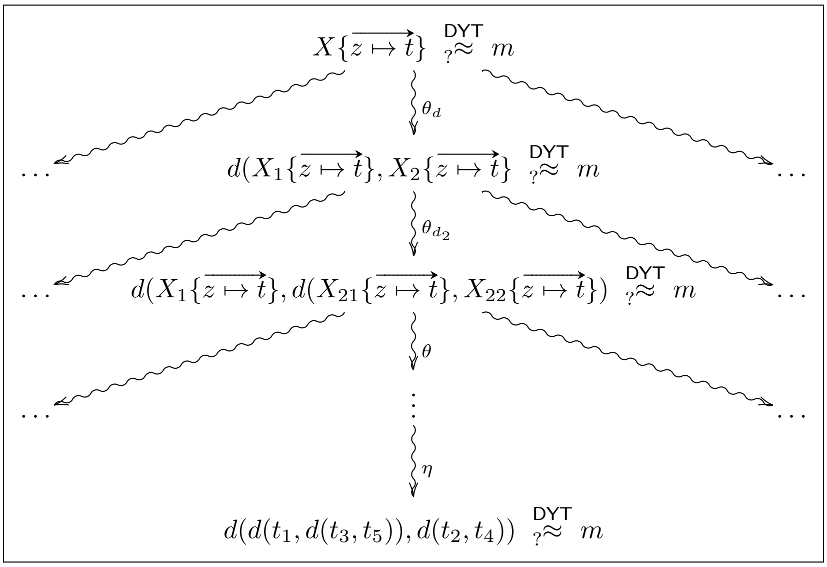

$X\{\overrightarrow{z \mapsto t}\} \overset{\mathsf{DYT}}{{}_?\approx} m$
θ_d
$d(X_1\{\overrightarrow{z \mapsto t}\}, X_2\{\overrightarrow{z \mapsto t}\} \overset{\mathsf{DYT}}{{}_?\approx} m$
θ_{d_2}
$d(X_1\{\overrightarrow{z \mapsto t}\}, d(X_{21}\{\overrightarrow{z \mapsto t}\}, X_{22}\{\overrightarrow{z \mapsto t}\}) \overset{\mathsf{DYT}}{{}_?\approx} m$
θ
η
$d(d(t_1, d(t_3, t_5)), d(t_2, t_4)) \overset{\mathsf{DYT}}{{}_?\approx} m$
...

Figure 7 Narrowing Subtree for the Solution.

The left branch represents 5 narrowing branches, one for each i. After applying rule (5) one has $\mathbf{c}_{\mathbf{d}^{i,1}} := d(t_i, X_2) \overset{\mathsf{DYT}}{{}_?\approx} m$. Similarly, $\mathbf{c}_{\mathbf{d}^{f,1}}$ represents 6 other possible branches, one for each function symbol from Σ_{DYT}. Iterating this reasoning, we obtain the narrowing branch shown in Figure 7, which leads to a ground problem whose solution is positive.

The previous example illustrates the fact that a series of narrowing steps might be necessary in order to obtain a solution. Variables might need to be instantiated with constructors for two reasons:

- either the term m contains a sequence of constructors in its structure, therefore, the variables in the term being matched have to be instantiated with the same sequence of constructors, and rule (8) applies;
- or a sequence of constructors matches a sequence of the corresponding destructors in a term in Γ, enabling a rewriting rule to be applied.

As a consequence, the number of applications of DYT rules is bounded by $|\Gamma| + |m|$.

▶ **Theorem 21.** *If a narrowing derivation* $(\Delta_0 \vdash (subst([\vec{z}]X, \vec{t}), m) \rightsquigarrow_{\sigma_0} \ldots \rightsquigarrow_{\sigma_{k-1}} (\Delta_k \vdash u_k)$ *has more than* $|\Gamma| + |m|$ *narrowing steps then* $height(subst([\vec{z}]X, \vec{t})\sigma_0\sigma_1 \ldots \sigma_{k-1}) > height(m)$. *Therefore, it does not lead to a solution.*

Proof. Each application of a Dolev-Yao rule eliminates one symbol from the term. In the worst case, in all terms from Γ all the function symbols can be eliminated by a rule, before several steps of composition (with a constructor that has not just been eliminated) can be applied until one reaches the size of m.

Notice that for infinite branches of the form $\Pi_i\Pi_j\Pi_i\Pi_j \ldots$ or $dddd\ldots$ either the term m would have to be headed with the same sequence of functions or rewrite rules would be applied. By the Lifting Theorem, we can assume the compositions of substitutions are normalised, therefore, the only way to apply rewrite rules is when the terms in Γ contain, in the first case, a sequence of pairings $\langle\langle\langle\ldots\rangle\rangle\rangle$ or, in the second case, a sequence of encryptions $\{\{\ldots\}\}$. We cannot introduce a destructor followed by its corresponding constructor with a substitution, e.g, $\Pi_i\Pi_j\Pi_i\langle\langle\ldots\rangle\rangle$, otherwise the substitution would not be normalised. Since all the terms in Γ are finite, only a finite number of destructive rewrite rules could be applied and the number of constructive rewrite rules that could be applied is bounded by the size of m. The same reasoning applies when we have interleaving of destructors $d\Pi_i d\Pi_j d\Pi_i d\Pi_j \ldots$ or even constructors and destructors of the form $d\{\}d\{\}d\{\}$, when the encryption/decryption keys do not correspond. ◀

As a consequence, we obtain the decidability of the nominal IDP for DYT.

4 Basic Nominal Narrowing

Hullot [17] introduced *basic narrowing* to eliminate redundant narrowing derivations in order to give sufficient conditions for the termination of the narrowing process. Following [17], with the corrections made in [1, 24], we define *basic (closed) nominal narrowing.* In the rest of this section, $\mathsf{R} = \{R_k \equiv \nabla \vdash l_k \to r_k\}$ is a closed nominal rewrite theory.

▶ **Definition 22.** Consider a nominal term s and a set U of positions that are proper prefixes of s, that is, $U = \overline{\mathcal{P}os}(r)$, for some subterm r of s. We define by induction what it means for a nominal rewriting derivation $\Delta \vdash s = s_0 \to_{[C_0,R_0]} s_1 \to_{[C_1,R_1]}\to \ldots \to_{[C_{n-1},R_{n-1}]} s_n$ to be *based on* U and construct sets of positions $U_i \subset \mathcal{P}os(s_i)$, $0 \leq i \leq n$, inductively: the empty derivation is based on U, and $U_0 = U$; if a derivation up to s_i is based on U, then the derivation obtained from it by adding one step $s_i \to_{[C_i,R_i]} s_{i+1}$ is based on U iff $C_i \in U_i$, and in this case we take: $U_{i+1} = (U_i - \{C \in U_i | C_i \leq C\}) \cup \{C_i.C | C \in \overline{\mathcal{P}os}(r_i)\}$, where r_i denotes the right-hand side of the rule R_i in R[9].

A nominal rewrite step $\Delta \vdash C[s] \to C[s']$ at position C is *innermost* if for any C_i such that $C < C_i$ and $C[s] = C_i[s_i]$, there is no rewrite step $\Delta \vdash C_i[s_i] \to C_i[t]$ at position C_i. In other words, there is no rewrite step inside s. An innermost nominal rewrite derivation contains only innermost rewrite steps.

▶ **Lemma 23.** *Let $\Delta \vdash s \approx_\alpha s_0\eta$, with η normalised in Δ. Every innermost nominal rewrite derivation from $\Delta \vdash s$ is based on $\overline{\mathcal{P}os}(s_0)$.*

▶ **Definition 24.** A nominal narrowing derivation $(\Delta_0 \vdash s_0) \rightsquigarrow_{[C_0,R_0,\sigma_0]} \cdots \rightsquigarrow_{[C_{i-1},R_{i-1},\sigma_{i-1}]} (\Delta_i \vdash s_i)$, is *basic* if it is based on $\overline{\mathcal{P}os(s_0)}$ (in the same sense as in the previous definition for nominal rewriting derivation).

▶ **Theorem 25.** *The narrowing derivations constructed in Theorem 12 are all basic.*

Proof. Let $(\Delta_0 \vdash s_0) \rightsquigarrow_{[C'_0,R_0,\sigma_0]} \cdots \rightsquigarrow_{[C'_{n-1},R_{n-1},\sigma_{n-1}]} (\Delta_n \vdash s_n)$ be the nominal narrowing derivation associated by Theorem 12 with $\Delta \vdash s_0\eta = t_0 \to_{[C_0,R_0]} \cdots \to_{[C_{n-1},R_{n-1}]} t_n$, such that η is normalised. Since R is confluent we may assume that the nominal rewriting sequence from $\Delta \vdash s_0\eta$ is innermost. By Lemma 23, this nominal rewriting derivation is based on $\overline{\mathcal{P}os}(s_0)$, and since the sets U_i in the two derivations are equivalent, it follows that the considered nominal narrowing derivation is basic. ◀

▶ Remark. Definition 22, Lemma 23 can also be stated for closed narrowing. Theorem 16 holds also for closed basic narrowing.

The main interest of closed basic narrowing is that we can give a sufficient condition for the termination of the narrowing process when we consider only basic $\rightsquigarrow$-derivations and therefore for the termination of the corresponding nominal E-unification procedure.

[9] Given $C_i = (s_i, \_)$ and $C = (s, \_)$, it follows $C_i.C = (s_i\{\_ \mapsto s\}, \_)$ and $C_i \leq C$ if $\exists t : s_i\{\_ \mapsto t\} = s$.

▶ **Proposition 26.** *Let* $\mathsf{R} = \{\nabla_k \vdash l_k \to r_k\}$ *be a convergent nominal rewriting system such that any basic* $\rightsquigarrow$*-derivation issuing from any of the right-hand sides* r_k *terminates. Then any basic* $\rightsquigarrow$*-derivation issuing from any nominal term terminates.*

The previous proposition also holds for basic closed narrowing.

▶ **Theorem 27.** *Basic closed nominal narrowing is complete for convergent closed nominal rewriting systems.*

Moreover, if R satisfies the hypothesis of Proposition 26, nominal basic narrowing leads to a complete and finite E-unification algorithm.

5 Conclusion and Future Work

We have introduced the *nominal narrowing* relation and designed a general nominal E-unification procedure, which is complete for a wide class of theories, namely, the theories defined by convergent closed nominal rewriting systems.

There is a lot of work to be done regarding nominal E-unification. A first step would be to study the relationship between nominal narrowing and pattern narrowing [26]. For the analysis of protocols, it would be interesting to study nominal unification modulo equational theories including associativity and commutativity axioms. From a practical point of view, narrowing strategies should be studied, such as lazy narrowing for nominal terms, and also more general versions of nominal narrowing such as conditional [26] and variant [10] narrowing, which have interesting applications [21, 23]. We would like to define conditions for termination of nominal narrowing similar to the *finite variant* and *boundedness* properties [6], to obtain an alternative way to study the security of protocols, via nominal narrowing.

Acknowledgements. We thank Santiago Escobar, Jesus Dominguez Alvarez and the FSCD reviewers for their valuable comments, which helped us improve the paper.

References

1 M. Alpuente, S. Escobar and J. Iborra. *Termination of Narrowing Revisited.* In *Theoretical Computer Science*, 410(46):4608–4625, 2009.

2 F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambr. Univ. Press, 1998.

3 C. Calvès and M. Fernández. *The First-Order Nominal Link. Logic-Based Program Synthesis and Transformation - 20th International Symposium (LOPSTR 2010), Hagenberg, Austria, 2010, Revised Selected Papers*, vol. 6564 of *LNCS*, 234–248, 2011.

4 C. Calvès and M. Fernández. *Matching and alpha-equivalence check for nominal terms. Journal of Computer and System Sciences*, 76(5):283–301, 2009.

5 J. Cheney and C. Urban. *Nominal Logic Programming*, *ACM Trans. Program. Lang. Syst.* 30(5), Article 26, 2008.

6 H. Comon-Lundh and S. Delaune. *The finite Variant Property: How to get rid of some algebraic properties.* In *Proc. of Rewriting Techniques and Applications (RTA'05)*, vol. 3467 of *LNCS*, 294–307, 2005.

7 S. Delaune and F. Jacquemard. *A Decision Procedure for the Verification of Security Protocols with Explicit Destructors.* In *Proc. of 11th ACM Conference on Computer and Communications Security (CCS'04)*, pages 278–287, 2004.

8 N. Dershowitz and D.A. Plaisted. *Equational Programming. Machine Intelligence 11*, Clarendon Press, Oxford, Ch. 2, 21–56, 1988.

9 S. Escobar, J. Meseguer and R. Sasse. *Variant Narrowing and Equational Unification. Electr. Notes Theor. Comput. Sci.,* 238(3):103–119, 2009.

10 S. Escobar, R. Sasse and J. Meseguer *Folding Variant Narrowing and Optimal Variant Termination.* In *Proc. 8th International Workshop Rewriting Logic and Its Applications (WRLA 2010)*, vol. 6381 of *LNCS*, 52–68, 2010.

11 M. Fernández and M. J. Gabbay. Nominal rewriting. *Information and Computation*, 205(6):917–965, 2007.

12 M. Fernández and M. J. Gabbay. Closed Nominal Rewriting and Efficiently Computable Nominal Algebra Equality In *Proc. of 5th International Workshop on Logical Frameworks and Meta-languages: Theory and Practice (LFMTP)*, 37–51, 2010.

13 M. Fernández, M. J. Gabbay, and I. Mackie. *Nominal Rewriting Systems.* In *Proc. 6th Int.Conf. on Principles and Practice of Declarative Programming (PPDP 2004)*, 2004.

14 M. Fernández and A. Rubio. *Nominal Completion for Rewrite Systems with Binders.* In *Proc. of 39th International Colloquium Automata, Languages, and Programming (ICALP' 2012)* , vol. 7392 of *LNCS*, 201–213, 2012.

15 M. J. Gabbay and A. Mathijssen. *Nominal universal algebra: equational logic with names and binding. Journal of Logic and Computation*, 19(6):1455–1508, 2009.

16 M. J. Gabbay and A. M. Pitts. *A New Approach to Abstract Syntax with Variable Binding. Formal Aspects of Computing*, 13(3–5):341–363, 2001.

17 J-M. Hullot. *Canonical Forms and Unification* In *Proc. 5th Conference on Automated Deduction (CADE'80)*, vol. 87 of *LNCS*, 318–324, 1980.

18 J. Levy and M. Villaret. *Nominal unification from a higher-order perspective.* In Proc. Rewriting Techniques and Applications (RTA'08), vol. 5117 in *LNCS*, 2008.

19 J. Levy and M. Villaret. *An efficient nominal unification algorithm.* In Proc. of Rewriting Techniques and Applications (RTA'10), vol. 6 of *LIPIcs*, 209–226, 2010.

20 J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.

21 C. Meadows. *Using Narrowing in the Analysis of Key Management Protocols.* In Proc. (IEEE) Symposium on Security and Privacy, 138–147, 1989.

22 J. Meseguer. *Strict Coherence of Conditional Rewriting Modulo Axioms.* Technical Report, Computer Science Department, University of Illinois at Urbana-Champaign, August 2014. Available from `http://hdl.handle.net/2142/50288`

23 J. Meseguer and P. Thati. *Symbolic Reachability Analysis Using Narrowing and its Application to Verification of Cryptographic Protocols.* In Higher-Order and Symbolic Computation, 20(1):123–160, 2007.

24 A. Middeldorp and E. Hamoen. *Completeness Results for Basic Narrowing.* Applicable Algebra in Engineering, Communication and Computing, 5(3-4):213–253, 1994.

25 D. Miller *A Logic Programming Language with Lambda-Abstraction, Function Variables and Simple Unification.* Journal of Logic and Computation, 1(4):497–536, 1991.

26 C. Prehofer. *Solving Higher-Order Equations from Logic to Programming.* In Progress in Theoretical Computer Science, Birkhäuser, 1997.

27 T. Sheard. *Type-Level Computation Using Narrowing in Ωmega.* In Proc. of Programming Languages meets Program Verification (PLPV), *Electr. Notes Theor. Comput. Sci.,* 174(7):105–128, 2006.

28 M. R. Shinwell, A. M. Pitts and M. J. Gabbay. *FreshML: Programming with Binders Made Simple. SIGPLAN Notices* 38(9):263–274, 2003.

29 C. Urban, A. M. Pitts and M. J. Gabbay. *Nominal Unification.* Theoretical Computer Science, 323(1–3):473–497, 2004.

Synthesis of Functional Programs with Help of First-Order Intuitionistic Logic[*]

Marcin Benke[1], Aleksy Schubert[2], and
Daria Walukiewicz-Chrząszcz[3]

1 Institute of Informatics, University of Warsaw, Warsaw, Poland
ben@mimuw.edu.pl
2 Institute of Informatics, University of Warsaw, Warsaw, Poland
alx@mimuw.edu.pl
3 Institute of Informatics, University of Warsaw, Warsaw, Poland
daria@mimuw.edu.pl

Abstract

Curry-Howard isomorphism makes it possible to obtain functional programs from proofs in logic. We analyse the problem of program synthesis for ML programs with algebraic types and relate it to the proof search problems in appropriate logics. The problem of synthesis for closed programs is easily equivalent to the proof construction in intuitionistic propositional logic and thus fits in the class of PSPACE-complete problems. We focus further attention on the synthesis problem relative to a given external library of functions. It turns out that the problem is undecidable for unbounded instantiation in ML. However its restriction to instantiations with atomic types only results in a case equivalent to proof search in a restricted fragment of intuitionistic first-order logic, being the core of Σ_1 level of the logic in the Mints hierarchy. This results in EXPSPACE-completeness for this special case of the ML program synthesis problem.

1998 ACM Subject Classification D.1.2 Automatic Programming, F.4.1 Mathematical Logic, I.2.2 Automatic Programming

Keywords and phrases ML, program synthesis

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.12

1 Introduction

In general, program synthesis is the problem of the following form: given a not necessarily executable specification, find an executable program satisfying that specification. The idea of mechanically constructed programs or more precisely programs correct-by-construction appeared already a long time ago and not only in functional programming but also in imperative programming [22, 5] and in logic programming. This idea arises naturally in the context of increasing demand for programmer's productivity.

In 2005 Augustsson created Djinn [1], "a small program that takes a (Haskell) type and gives you back a function of that type if one exists." As an example supporting usefulness of such program extraction, he used the key functions of the continuation monad:

```
# return, bind, and callCC in the continuation monad
   Djinn> type C a = (a -> r) -> r
   Djinn> returnC ? a -> C a
```

[*] This work is supported by NCN grant DEC-2012/07/B/ST6/01532.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 12; pp. 12:1–12:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Given this query he obtained an answer:

```
returnC :: a -> C a
returnC x1 x2 = x2 x1
```

Moreover, for

```
Djinn> bindC ? C a -> (a -> C b) -> C b
```

he obtained

```
bindC :: C a -> (a -> C b) -> C b
bindC x1 x2 x3 = x1 (\ c15 -> x2 c15 (\ c17 -> x3 c17))
```

and finally for

```
Djinn> callCC ? ((a -> C b) -> C a) -> C a
```

he got

```
callCC :: ((a -> C b) -> C a) -> C a
callCC x1 x2 = x1 (\ c15 _ -> x2 c15) (\ c11 -> x2 c11)
```

Indeed, in certain situations such as the one above, there exists only a handful of functions of a given type (sometimes — barring usage of functions such as *error* or *undefined* — just one). If the type is complex and specific enough, we may be content with any of them. In such cases the programmer should be liberated from the effort of coding and the code of the program should be given for acceptance as soon as the type of the function is given.

The reference point for type systems in functional programming languages with static typechecking is the model language of ML [9, 4]. This language brings the `let` $x = M$ `in` N construct into the inventory of program term constructs available in the standard Curry-style λ-calculus. This makes it possible to assign to the term M a polymorphic type scheme $\forall \alpha_1 \dots \alpha_n.\tau$ and use it within N in various places so that in each of them $\alpha_1, \dots, \alpha_n$ can be instantiated with different concrete types. This design of language leads to the situation that formal typing judgements use contexts that in addition to assertions $x : A$ about (monomorphic) types contain assertions about type schemes $x : \tau$.

However, the expressivity of ML types is very limited. Stronger type theories were applied to extend the reasoning possibilities for functional programs, including calculus of constructions [3] or Martin-Löf type theory [12]. Types correspond there to propositions in richer logics and one can, for example, specify sorting as

$$\forall x\, \exists y\ \mathrm{ordered}(y) \wedge \mathrm{permutation}(x, y)$$

A constructive proof of this specification should be turned into a sorting procedure by a program synthesis mechanisms. This view is supported by the Curry-Howard isomorphism, which identifies a proof (of a specification) with a program (meeting that specification).

In richer type systems the relation between proofs and programs is not simple and can take up the form of a program extraction procedure. In its course, all “computationally irrelevant” content is deleted while the remaining executable parts of the proof are guaranteed to be correct with respect to the specification formula proved by the initial proof.

Program extraction procedures are typically associated with proof assistants. In particular, Minlog [18], Isabelle/HOL [14], Coq [2] have such mechanisms. Let us examine closer their features based upon the extraction mechanism of Coq, designed by Paulin [15] and Letouzey [10]. The input for the extraction mechanism are Coq proofs and functions. The extracted programs are expressed in functional languages such as Ocaml, Haskell and Scheme. One delicate point concerns the typability of the extracted code, since neither Haskell nor Ocaml have dependent types. A first solution is to use a type-free language like Scheme. Another possibility is to use ML-like typing as long as possible and insert some unsafe type coercions when needed: `Obj.magic` in Ocaml and `unsafeCoerce` in Haskell. This feature implies that resulting types of functions may go beyond the realm of tautologies of the base logic associated with the ML type system.

In this paper we are interested in the problem of program generation for functional programs. For this we focus on the model language of ML. In its basic form, the problem has already been fully exploited in Djinn. However, the example of program generation with help of Djinn above used no external context of library functions, i.e. no additional symbols were available that could occur in the generated program except from those explicitly declared in the body of the generated function. A more realistic case is when the programmer expects that the program to be created should contain certain symbols that were defined beforehand in the available program libraries. This leads to the *problem of synthesis* for ML:

> Given a set Γ of library functions together with their types and a type τ of the goal program, find a term M that has type τ under the context Γ.

In the current paper we analyse unrestricted problem of program synthesis for ML with algebraic types. The problem turns out to be undecidable when we allow Γ to contain not only constructive tautologies, but symbols of arbitrary type. We further consider ML with restricted type instantiations so that types can be instantiated only with atomic types. We can prove that this case is equivalent to proof search for a restricted fragment of intuitionistic first-order logic, being the core of Σ_1 level of the logic in the Mints hierarchy. As a result we obtain EXPSPACE-completeness for the ML program synthesis problem for so constrained instantiations.

The current paper is constructed as follows. In Section 2 the type systems and logics used in the paper are defined. The problem of program synthesis is studied in Section 3. In Subsection 3.1 we present the undecidability proof for ML with unrestricted instantiations and in Subsection 3.2 we analyse the situation when the instantiations are restricted to atomic types. We present conclusions together with possible further work directions in Section 4.

2 Presentation of Logical Systems

2.1 The System of ML

We assume that an infinite set of object variables *Vars* is available. The expressions of ML are defined with the following grammar

$$M ::= x \mid \lambda x.M \mid M_1 M_2 \mid \texttt{let}\ x = M_1\ \texttt{in}\ M_2$$

where $x \in \mathit{Vars}$. The set $\mathrm{FV}(M)$ of *free variables in a term M* is defined inductively as

- $\mathrm{FV}(x) = \{x\}$, $\mathrm{FV}(\lambda x.M) = \mathrm{FV}(M)\backslash\{x\}$, $\mathrm{FV}(M_1 M_2) = \mathrm{FV}(M_1) \cup \mathrm{FV}(M_2)$, $\mathrm{FV}(\texttt{let}\ x = M_1\ \texttt{in}\ M_2) = \mathrm{FV}(M_1) \cup (\mathrm{FV}(M_2)\backslash\{x\})$.

This notion is extended naturally to sets of terms. The capture avoiding substitution that assigns M to a variable x is written $[x := M]$ and as usual extended to $[x_1 := M_1, \ldots, x_n := M_n]$ when many variables are involved. For the definition of types we assume that we have a finite, but of unbounded size, set of type constants *TConst* as well as an infinite set *TVars* of type variables. The types and type schemes are defined with the grammar

$$A ::= o \mid \alpha \mid A \to A \quad \text{(types)} \qquad\qquad \sigma ::= A \mid \forall \alpha.\sigma \quad \text{(type schemes)}$$

$$\frac{A \ \mathsf{inst}\ \sigma}{\Gamma, x : \sigma \vdash x : A}\ (var)$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}\ (\to I) \qquad \frac{\Gamma \vdash M_1 : A \to B \quad \Gamma \vdash M_2 : A}{\Gamma \vdash M_1 M_2 : B}\ (\to E)$$

$$\frac{\Gamma \vdash M_1 : B \quad \Gamma, x : \mathsf{gen}(\Gamma, B) \vdash M_2 : A}{\Gamma \vdash \mathtt{let}\ x = M_1\ \mathtt{in}\ M_2 : A}\ (\mathtt{let})$$

Figure 1 Rules of ML.

where $o \in TConst, \alpha \in TVars$. A *context* Γ is a finite set of pairs $x : \sigma$ such that when $x : \sigma$ and $y : \tau$ occur in Γ then $x \neq y$. The set $\mathrm{FTV}(\sigma)$ of *free type variables in type* σ is defined inductively as

- $\mathrm{FTV}(o) = \emptyset$, $\mathrm{FTV}(\alpha) = \{\alpha\}$, $\mathrm{FTV}(A \to B) = \mathrm{FTV}(A) \cup \mathrm{FTV}(B)$, $\mathrm{FTV}(\forall\alpha.\sigma) = \mathrm{FTV}(\sigma) \backslash \{\alpha\}$.

This notion extends naturally to sets of types and to contexts.

The relation $A\ \mathsf{inst}\ \forall\alpha_1 \ldots \alpha_n.A'$ holds when there is a (capture avoiding) substitution S that assigns types to type variables with $\{\alpha_1, \ldots, \alpha_n\} = \mathrm{dom}(S)$, where $\mathrm{dom}(S)$ is the domain of S, such that $A = S(A')$. The function $\mathsf{gen}(\cdot)$ is defined as $\mathsf{gen}(\Gamma, B) = \forall\alpha_1 \ldots \alpha_n.B$ where $\{\alpha_1, \ldots, \alpha_n\} = \mathrm{FTV}(B) \backslash \mathrm{FTV}(\Gamma)$. The type assignment rules for the system are presented in Figure 1. We write $\Gamma \vdash_{\mathrm{ML}} M : \tau$ to tell that the judgement $\Gamma \vdash M : \tau$ is derivable according to these rules.

This language can be extended to handle *algebraic types*. In that case we assume that there is a finite set of algebraic type constructors *TAlg*, and each $P \in TAlg$ has arity given by $arity(P)$. The types are formed according to the following grammar

$$\begin{array}{ll} A ::= o \mid \alpha \mid A \to A \mid P(A_1, \ldots, A_n) & \text{(types)} \\ \sigma ::= A \mid \forall\alpha.\sigma & \text{(type schemes)} \end{array}$$

where $o \in TConst, \alpha \in TVars$, $P \in TAlg$ and $arity(P) = n$. The algebraic types come usually equipped with term constants that make it possible to construct values of the algebraic types and destruct them. We omit the constructors from our account since they can be introduced to our setting as polymorphic object variables placed in contexts (however, the presence of them may change the complexity bounds, as the presence of any other class of variables can do). The algebraic type of the form $P(A_1, \ldots, A_n)$, type constant or type variable are called *atomic types*. If any of them occurs at the end of a type or type scheme, it is called the *target* of the type or type scheme, respectively.

This system is accompanied by a reduction relation $\to_\beta$ that is defined by the following redexes

$$(\lambda x.M)N \quad \to_\beta \quad M[x := N], \qquad\qquad \mathtt{let}\ x = M\ \mathtt{in}\ N \quad \to_\beta \quad N[x := M]$$

extended by syntactic closure. The transitive-reflexive closure of $\to_\beta$ is $\to_\beta^*$. The *normal form* of a term M is defined as a term $\mathrm{NF}(M)$ such that there is no M' such that $\mathrm{NF}(M) \to_\beta M'$. The system of ML enjoys the following proof-theoretic properties:

▶ **Theorem 1.**

- **(Subject reduction)** *If* $\Gamma \vdash_{\mathrm{ML}} M : A$ *and* $M \to_\beta N$ *then* $\Gamma \vdash_{\mathrm{ML}} N : A$.
- **(Church-Rosser)** *If* $M \to_\beta^* N_1$ *and* $M \to_\beta^* N_2$ *then there is a term* M' *such that* $N_1 \to_\beta^* M'$ *and* $N_2 \to_\beta^* M'$.
- **(Strong normalisation)** *For each term* N *such that* $\Gamma \vdash_{\mathrm{ML}} N : A$ *there is no infinite sequence* M_i *for* $i \in \mathbb{N}$ *such that* $N = M_0$ *and* $M_i \to_\beta M_{i+1}$ *for* $i \in \mathbb{N}$.

$$\frac{A \ \mathsf{inst}_1 \ \sigma}{\Gamma, x : \sigma \vdash x : A} \ (var)$$

Figure 2 The modified rule (var) of ML$_1$.

$$\frac{}{\Gamma, x : A \vdash x : A} \ (var)$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \ (\to I) \qquad \frac{\Gamma \vdash M_1 : A \to B \quad \Gamma \vdash M_2 : A}{\Gamma \vdash M_1 M_2 : B} \ (\to E)$$

Figure 3 Rules of the simply-typed lambda calculus.

Proof. The subject reduction property can be attributed to Dubois [6]. The rest is obtained as a folklore result resulting from an obvious embedding of the system to System F of Girard and Reynolds [8, 16]. ◀

By a straightforward inspection of cases we obtain the following proposition that describes the set of normal forms in ML.

▶ **Proposition 2.**

1. *If M is an* ML *term in normal form then $M = x$, $M = xM_1 \dots M_n$ for some $n \geq 1$, or $M = \lambda x.M_1$ where $M_1, \dots, M_n$ are in normal form.*
2. *If $\Gamma \vdash_{\mathrm{ML}} M : A \to B$ is in normal form then*
   a. *either $M = xM_1 \cdots M_n$ for $n \geq 0$ with $x : \forall \alpha_1 \dots \alpha_k.A_1 \to \cdots \to A_n \to C \in \Gamma$, $A \to B = C[\alpha_1 := B_1, \dots, \alpha_k := B_k]$ for some $B_1, \dots, B_k$ and $\Gamma \vdash M_i : A_i[\alpha_1 := B_1, \dots, \alpha_k := B_k]$ for $i = 1, \dots, n$,*
   b. *or $M = \lambda x.M_0$ where $\Gamma, x : A \vdash_{\mathrm{ML}} M_0 : B$.*
3. *If $\Gamma \vdash_{\mathrm{ML}} M : A$ where A is atomic type and M is in normal form then $M = xM_1 \cdots M_n$ for $n \geq 0$ with $x : \forall \alpha_1 \dots \alpha_k.A_1 \to \cdots \to A_n \to C \in \Gamma$, $A = C[\alpha_1 := B_1, \dots, \alpha_k := B_k]$ for some $B_1, \dots, B_k$ and $\Gamma \vdash M_i : A_i[\alpha_1 := B_1, \dots, \alpha_k := B_k]$ for $i = 1, \dots, n$.*

Proof. Standard arguments are left to the reader. ◀

Observe that normal forms in this system have no occurrences of the `let` · `in` · construct.

We consider here in more detail a restricted version of the system ML, namely ML$_1$, in which the terms, types and type schemes remain the same as in ML, but the instantiation relation `inst` is restricted to `inst`$_1$ where $A \ \mathsf{inst}_1 \ \forall \alpha_1 \dots \alpha_n.A'$ holds whenever there is a substitution S such that $\{\alpha_1, \dots, \alpha_n\} = \mathrm{dom}(S)$, $A = S(A')$, and for each $\alpha \in \mathrm{dom}(S)$ we have $|S(\alpha)| = 1$. In this system only one rule is modified, namely the (var) rule, and it takes the form presented in Figure 2.

Simply-typed Lambda Calculus. Simply-typed lambda calculus may be viewed as a restriction of ML, the terms and types of which respectively are

$$M ::= x \mid \lambda x.M \mid M_1 M_2 \qquad\qquad A ::= o \mid \alpha \mid A \to A$$

The rules are presented in Figure 3. Note that the (var) rule is a special case of the ML (var) rule for the empty string of quantifiers (i.e. when the instantiated type scheme is a type). Also note that the rule for `let` · `in` · is missing. We write $\Gamma \vdash_{\to} M : \tau$ to tell that the judgement $\Gamma \vdash M : \tau$ is derivable according to these rules in Figure 3.

$$\frac{}{\Gamma, x : \varphi \;\vdash x : \varphi}\ (Ax)$$

$$\frac{\Gamma, x : \varphi \vdash M : \psi}{\Gamma \vdash \lambda x : \varphi.M \;\;:\;\; \varphi \to \psi}\ (\to I) \qquad \frac{\Gamma \vdash M : \varphi \to \psi \quad \Gamma \vdash N : \varphi}{\Gamma \vdash MN : \psi}\ (\to E)$$

$$\frac{\Gamma \vdash M : \varphi}{\Gamma \vdash \lambda X\, M : \forall X \varphi}\ (\forall I)^{*} \qquad \frac{\Gamma \vdash M : \forall X \varphi}{\Gamma \vdash MY : \varphi[X := Y]}\ (\forall E)$$

[*] This rule is subject to the standard eigenvariable condition.

Figure 4 Proof assignment rules for intuitionistic first-order logic with →, ∀.

2.2 Intuitionistic First-order Logic

For the definition of formulas in intuitionistic first-order logic (IFOL) we need an infinite set of object variables Vars_{IFOL} as well as a finite, but of unbounded size set of predicates *Preds* such that each $P \in \mathit{Preds}$ has arity given by $\mathit{arity}(P)$ (we abuse the notation and use the same metavariables for predicates and algebraic types as they are translated bijectively in our approach, this also explains the overloading of the function $\mathit{arity}(\cdot)$). Formulas of intuitionistic first-order logic with →, ∀ are built using the following grammar

$$\varphi ::= P(X_1, \ldots, X_n) \mid \varphi_1 \to \varphi_2 \mid \forall X\, \varphi$$

where $X, X_1, \ldots, X_n \in \mathit{Vars}_{IFOL}$. In this article we use formulas of restricted form that can be described within the framework of the Mints hierarchy [17] where the presence of a formula on a particular level depends on the form of its classical prenex form. We can define syntactically the resulting classes of formulas $\mathbf{\Sigma}_n, \mathbf{\Pi}_n$ in the following way. The sets $\mathbf{\Sigma}_0 = \mathbf{\Pi}_0$ are equal to the set of quantifier-free formulas. Further,

$$\begin{aligned} \mathbf{\Sigma}_{n+1} &::= P(X_1, \ldots, X_n) \mid \mathbf{\Pi}_n \mid \mathbf{\Pi}_{n+1} \to \mathbf{\Sigma}_{n+1} \\ \mathbf{\Pi}_{n+1} &::= P(X_1, \ldots, X_n) \mid \mathbf{\Sigma}_n \mid \mathbf{\Sigma}_{n+1} \to \mathbf{\Pi}_{n+1} \mid \forall X\, \mathbf{\Pi}_{n+1} \end{aligned}$$

where $X_1, \ldots, X_n, X \in \mathit{Vars}_{IFOL}$. We introduce an additional class of formulas in natural form defined by the following grammar from the symbol $\mathbf{N}$

$$\mathbf{N} ::= \mathbf{O} \mid \mathbf{B} \to \mathbf{N} \qquad \mathbf{B} ::= \mathbf{O} \mid \forall X\, \mathbf{B} \qquad \mathbf{O} ::= P(X_1, \ldots, X_n) \mid \mathbf{O} \to \mathbf{O}$$

where $X_1, \ldots, X_n, X \in \mathit{Vars}_{IFOL}$. The formulas are natural in the sense that they avoid nested quantifiers, and people tend to avoid internal quantification. Again the predicate $P(X_1, \ldots, X_n)$, where $n \geq 0$, at the end of a formula is called *target* of the formula. We observe that $\mathbf{N} \subseteq \mathbf{\Sigma}_1$.

The proofs for valid formulas of the logic can be represented by terms. To define them we need an infinite set of proof variables Vars_P. The terms are generated by the grammar

$$M ::= x \mid \lambda x : \varphi.M \mid M_1 M_2 \mid \lambda X\, M \mid MX$$

where $x \in \mathit{Vars}_P$ and $X \in \mathit{Vars}_{IFOL}$. The proof assignment rules for the logic are presented in Figure 4.

This system is again accompanied by a reduction relation $\to_\beta$ that is defined through the redex $(\lambda x.M)N \to_\beta M[x := N]$ as well as $(\lambda X M)Y \to_\beta M[X := Y]$ extended by syntactic closure. The transitive-reflexive closure of $\to_\beta$ is $\to_\beta^*$. The *normal form* of a term M is defined as a term $\mathrm{NF}(M)$ such that there is no M' with $\mathrm{NF}(M) \to_\beta M'$. We also define the notion of a proof term in *long normal form*, abbreviated *lnf*.

- If N is an lnf of type φ then $\lambda X\, N$ is an lnf of type $\forall X\, \varphi$.
- If N is an lnf of type ψ then $\lambda x : \varphi.\, N$ is an lnf of type $\varphi \to \psi$.
- If $N_1, \dots, N_n$ are lnf or object variables, and $xN_1 \dots N_n$ is of an atom type, then $xN_1 \dots N_n$ is an lnf.

We have now (see e.g. the paper by Schubert et al [17]) the following basic properties.

▶ **Proposition 3.**
1. *If φ is intuitionistically derivable from Γ then $\Gamma \vdash N : \varphi$, for some lnf N.*
2. *If $\Gamma \vdash N : P(\vec{x})$, where $P(\vec{x})$ is an atomic formula and N is an lnf, then $N = X\vec{D}$, where $(X : \psi) \in \Gamma$ with target$(\psi) = P$, and $\vec{D}$ is a sequence that may contain proof terms and object variables.*

▶ **Problem 4** (Provability Problem)**.** *Given a context Γ and formula φ check if there is a proof term M such that $\Gamma \vdash_{\mathrm{IFOL}} M : \varphi$ holds.*

This problem is known to be EXPSPACE-complete for formulas in $\mathbf{\Sigma}_1$. We have even more.

▶ **Theorem 5.** *The provability problem when formulas are restricted to come from $\mathbf{\Sigma}_1$ is EXPSPACE-complete. The same holds for formulas in $\mathbf{\Sigma}_1 \cap \mathbf{N} = \mathbf{N}$.*

The paper by Schubert et al [17] states the above result only for $\mathbf{\Sigma}_1$. However, the hardness proof uses formulas from $\mathbf{N}$ so the result holds for this restricted class.

3 The Problem of Synthesis

The program synthesis problem in its most basic formulation is as follows

▶ **Problem 6** (Closed Program Synthesis for ML)**.** *Given a type A of* ML *check if there is a term M such that $\vdash_{\mathrm{ML}} M : A$.*

We can restate it in the vocabulary of programming languages as follows: given a type τ find a program M that has the type.

One may be tempted to ask why we demand type instead of type scheme in the problem, but this is easily explained by the fact that there are no terms of any type scheme in the language of ML. Functional programming languages make it possible to define functions for type schemes, but they do it so by implicit introduction of `let` · `in` · construct.

For completeness we present here a proof that the problem of closed program synthesis is PSCPACE-complete, but this is rather a folklore result, which is difficult to attribute to a particular publication. The proof is done by reduction of the type inhabitation problem for the simply-typed lambda calculus, which is known to be PSCPACE-complete [21]. Actually, the work of Augustsson [1] is based on the same observation we explicate here.

▶ **Lemma 7.**
1. *If $\Gamma \vdash_{\to} M : A$ then $\Gamma \vdash_{\mathrm{ML}} M : A$.*
2. *If $\Gamma \vdash_{\mathrm{ML}} M : A$ where Γ does not contain type schemes and $M = \mathrm{NF}(M)$ then $\Gamma \vdash_{\to} M : A$.*

Proof. The first claim follows easily by induction over M and observation that the simply-typed lambda calculus is a subsystem of ML.

The second claim follows by observation that a normal form of an ML term does not contain occurrences of `let` · `in` ·. As the (`let`) rule can only be applied to a term of the form `let` · `in` ·, this rule cannot occur in the derivation. The remaining rules are the rules of the simply-typed lambda calculus so the claim follows. ◀

▶ **Theorem 8.** *The closed program synthesis for* ML *is PSPACE-complete. This holds even for programs with algebraic types.*

Proof. By subject reduction and strong normalisation properties we may restrict our search in the closed program synthesis problem for ML to terms in normal form.

Suppose we are given a type A in the simply-typed lambda calculus. Since simply-typed lambda calculus is a subsystem of ML, we can use an algorithm for the closed program synthesis for ML on this input. In case the algorithm answers positively, there is a term M such that $\vdash_{\mathrm{ML}} M : A$. By the strong normalisation property, we may assume M is in normal form. By Lemma 7(2) we obtain $\vdash_{\rightarrow} M : A$. As a result, the answer for ML is correct for the simply-typed lambda calculus. In case the algorithm answers negatively, i.e. there is no ML term of type A, we cannot have a term of the simply-typed lambda calculus of the type either. In case there is a term M such that $\vdash_{\rightarrow} M : A$, we immediately translate M to ML by Lemma 7(1) and obtain contradiction with the correctness of the algorithm for ML. This gives a polynomial time reduction of the inhabitation problem for the simply-typed lambda calculus to the program synthesis for ML. As the inhabitation problem is PSPACE-hard for this calculus [21], we obtain that closed program synthesis for ML is PSPACE-hard.

A similar argument proves that the program synthesis problem for ML is reduced to the inhabitation problem for the simply-typed lambda calculus. As a result, we obtain that the synthesis problem is in PSPACE, which concludes the proof that the problem is PSPACE-complete.

The algebraic types do not invalidate the argument as they only serve as new atoms. ◀

3.1 Program Synthesis in Context

The above-discussed version of the program synthesis problem does not cover the issue of program synthesis in full. Actually, programmers do not want their programs to be composed entirely from scratch. Instead they want to use a libraries with functions that offer more refined functionality than the basic language. This consideration leads to the following version of the program synthesis problem for ML.

▶ **Problem 9** (Program Synthesis for ML). *Given a context* Γ *and type* A *of* ML *check if there is a term* M *such that* $\Gamma \vdash_{\mathrm{ML}} M : A$.

One may be curious why the context Γ is not restricted to contain only types that are inhabited within the original type theory, i.e. ML in this case — procedures in a library must have been written in the same language. This broadening of the scope of possible contexts has two reasons. First, languages such as Haskell or Ocaml use, as mentioned in the introductory Section 1, page 3, extensions that make it possible to go beyond ML type discipline. Second, the libraries may be the result of incorporating some libraries written in foreign languages, e.g. Qt or GTK, which may use some internal global state and in this way enable presence of originally non-inhabited types.

One can observe that the synthesis problem of this kind was posed before in terms of Hilbert-style propositional logic. In such systems axiom schemes may be regarded as polymorphic operations that are available in the context. The provability problem (that may be viewed as the synthesis problem through the Curry-Howard isomorphism) is undecidable there, which is stated in the Linial-Post theorem [11] that holds even for the calculae with arrow only [19], which are very close to ML. However, these Hilbert-style systems use only two rules, namely the *modus ponens* together with the substitution rule and these are not enough to guarantee that the deduction theorem holds. Therefore, these results cannot be applied directly to the case of ML and we give here a direct reduction of the halting problem for two-counter automata.

A *two-counter automaton* $\mathcal{A}$ (introduced by Minsky [13]) is a tuple $\langle Q, q_I, F, \delta\rangle$ where Q is a finite set of states, $q_I \in Q$ is an initial state, $F \subseteq Q$ is a set of final states, and δ is a set of rules of the form

$$q, T \mapsto q', k_0, k_1 \tag{1}$$

where $q, q' \in Q$, $T \in \{Z_0, Z_1, NZ_0, NZ_1\}$, $k_0, k_1 \in \{0, -1, +1\}$. The value T describes a test on one of the available two counters that enables the rule to fire. It can be described precisely by means of configurations. A *configuration* of the automaton is a triple $\langle q, l_0, l_1\rangle$ where $q \in Q$, $l_0, l_1 \in \mathbb{N}$. A rule (1) is applicable to a configuration $\langle q, l_0, l_1\rangle$ when

- $T = Z_i$, $l_i = 0$, $l_{\bar{i}} \neq 0$, and $k_i \geq 0$, or
- $T = Z_i$, $l_i = 0$, $l_{\bar{i}} = 0$, and $k_0, k_1 \geq 0$, or
- $T = NZ_i$, $l_i \neq 0$, and $l_{\bar{i}} \neq 0$, or
- $T = NZ_i$, $l_i \neq 0$, $l_{\bar{i}} = 0$, and $k_{\bar{i}} \geq 0$

for $i \in \{0, 1\}$ and $\bar{i} = i + 1 \mod 2$. Observe that the rules that perform subtraction can only be fired when the resulting counter is non-negative. We fix an automaton $\mathcal{A}$ for the rest of the section. Halting is defined inductively as follows. We say that the automaton *halts from a configuration* $\langle q, l_0, l_1\rangle$ when

- either $q \in F$ or
- there is a rule $q, T \mapsto q', k_0, k_1$ applicable to $\langle q, l_0, l_1\rangle$ such that $\mathcal{A}$ halts from $\langle q', l_0 + k_0, l_1 + k_1\rangle$.

We say that the automaton *halts* when it halts from the configuration $\langle q_I, 0, 0\rangle$.

In our construction we use the following vocabulary of type constants and constructors

- type constants: $\mathsf{loop}, \mathsf{start}, p, \mathsf{1}, \mathsf{0}$ and
- type constructors: P, R of arity 1.

We need first to provide representation for states. For this we define types $R^0(A) = A$ and $R^{i+1}(A) = R(R^i(A))$. Assume that the set of states is $Q = \{q_0, \ldots, q_n\}$. We can represent the state q_i by $A_{q_i} = R^i(\mathsf{0})$. We write $L_0^0 = \mathsf{0}, L_1^0 = \mathsf{1}, L_0^{i+1} = P(P^i(\mathsf{0})), L_1^{i+1} = P(P^i(\mathsf{1}))$. We can now define an ML type $A\langle q, l_0, l_1\rangle$ that represents the configuration $\langle q, l_0, l_1\rangle$ of the automaton

$$A\langle q, l_0, l_1\rangle = L_0^{l_0} \to L_1^{l_1} \to A_q \to p. \tag{2}$$

With this in mind, we can define formulas that represent particular kinds of automaton rules. Let us define first formulas that make it possible to test for zero and non-zero:

$$\begin{array}{llll} B_{Z,0}(D_1, D_2) & = L_0^0 \to D_1 \to D_2 \to p, & B_{Z,1}(D_1, D_2) & = D_1 \to L_1^0 \to D_2 \to p, \\ B_{NZ,0}(D_1, D_2, D_3) & = P(D_1) \to D_2 \to D_3 \to p, & B_{NZ,1}(D_1, D_2, D_3) & = D_1 \to P(D_2) \to D_3 \to p \end{array}$$

as well as the formula that updates counters

$$C(\alpha, \beta, \gamma) = \alpha \to \beta \to \gamma \to p.$$

We can now define operations $B + k$ for $k \in \mathbb{Z}$.

$$\begin{array}{l} B + 0 = B, \quad B + (k+1) = P(B) + k \text{ for } k \geq 0 \\ P(B) + (k-1) = B + k, \text{ for } k < 0 \\ B + k = B, \text{ for } k < 0 \text{ and } B \neq P(B') \end{array}$$

We define $B-k = B+(-k)$. Observe that for $n = -1$ we have $\gamma - n = P(\gamma)$ and $\gamma - n + n = \gamma$. The context $\Gamma_{\mathcal{A}}$ consists of the following six kinds of type schemes:

(1) $(A\langle q_I, 0, 0\rangle \to \mathsf{loop}) \to \mathsf{start}$

(2) for each rule of the form $q, Z_0 \mapsto q', m, n$:
$\forall\gamma. B_{Z,0}(\gamma - n, A_q) \to (C(L_0^0 + m, \gamma - n + n, A_{q'}) \to \mathsf{loop}) \to \mathsf{loop}$

(3) for each rule of the form $q, Z_1 \mapsto q', m, n$:
$\forall\gamma. B_{Z,1}(\gamma - m, A_q) \to (C(\gamma - m + m, L_1^0 + n, A_{q'}) \to \mathsf{loop}) \to \mathsf{loop}$

(4) for each rule of the form $q, NZ_0 \mapsto q', m, n$:
$\forall\gamma_0\gamma_1. B_{NZ,0}(\gamma_0, \gamma_1 - n, A_q) \to$
$(C(\gamma_0 + 1 + m, \gamma_1 - n + n, A_{q'}) \to \mathsf{loop}) \to \mathsf{loop}$

(5) for each rule of the form $q, NZ_1 \mapsto q', m, n$:
$\forall\gamma_0\gamma_1. B_{NZ,1}(\gamma_0 - m, \gamma_1, A_q) \to$
$(C(\gamma_0 - m + m, \gamma_1 + 1 + n, A_{q'}) \to \mathsf{loop}) \to \mathsf{loop}$

(6) $\forall\gamma_0\gamma_1.(\gamma_0 \to \gamma_1 \to A_q \to p) \to \mathsf{loop}$ for $q \in F$

where $k_0, k_1 \geq 0$. An important property of $\Gamma_{\mathcal{A}}$ is that all its type schemes have targets being type constants. Since the notation above is quite dense, we give an example on how it expands. Suppose we want to obtain the concrete formula for the rule $q_1, Z_0 \mapsto q_2, +1, -1$. Assume that q_1 is represented as $R(\mathsf{0})$ and q_2 as $R(R(\mathsf{0}))$. The rule falls under the point (2) above and gives rise to the formula

$$\forall\gamma.(\mathsf{0} \to P(\gamma) \to R(\mathsf{0}) \to p) \to ((P(\mathsf{0}) \to \gamma \to R(R(\mathsf{0})) \to p) \to \mathsf{loop}) \to \mathsf{loop}.$$

It is worth pointing out here that the mentioned above type schemes are variations on the double negation principle (they turn into real double negation when loop is understood as falsity) and in this way resemble the very natural terms presented in our example on pages 12:1–12:2.

We say that a context Σ is *faithful for the automaton $\mathcal{A}$* when

- it consists only of pairs in one of the form: $x : A\langle q, l_0, l_1\rangle$,
- if $x : A\langle q, l_0, l_1\rangle \in \Sigma$ then either $q = q_I, l_0 = 0, l_1 = 0$ or there is a rule $q', T \mapsto q, k_0, k_1$ in δ, and a pair $x : A\langle q', l_0', l_1'\rangle \in \Sigma$ such that the rule is applicable to $\langle q', l_0', l_1'\rangle$, $l_0 = l_0' + k_0$, and $l_1 = l_1' + k_1$.

▶ **Lemma 10.** *If Σ is faithful for the automaton $\mathcal{A}$ with $x : A\langle q, l_0, l_1\rangle \in \Sigma$ such that $\mathcal{A}$ halts from $\langle q, l_0, l_1\rangle$ then there is a term M such that $\Gamma_{\mathcal{A}}, \Sigma \vdash_{\mathrm{ML}} M : \mathsf{loop}$.*

Proof. The proof is by a straightforward induction over the notion of halting from configuration. ◀

▶ **Lemma 11.** *If Σ is faithful for the automaton $\mathcal{A}$ and $\Gamma_{\mathcal{A}}, \Sigma \vdash_{\mathrm{ML}} M : \mathsf{loop}$ for some term M then there is $x : A\langle q, l_0, l_1\rangle \in \Sigma$ such that $\mathcal{A}$ halts from $\langle q, l_0, l_1\rangle$.*

Proof. We may assume that M is in normal form. The proof is by induction on the size of the term M. There is no term of size 1 of type loop so the base case follows.

For the inductive step we observe that M is in normal form and its type is a constant so it must be of the form $xM_1 \ldots M_n$ for some $x : \tau \in \Gamma_{\mathcal{A}}, \Sigma$. As no type in Σ has target loop we obtain that $x : \tau \in \Gamma_{\mathcal{A}}$. We analyse the cases for possible kinds of schemes in $\Gamma_{\mathcal{A}}$. We present here only the most interesting cases (2) and (4).

In case (2), $x : \forall\gamma. B_{Z,0}(\gamma - n, A_q) \to (C(L_0^0 + m, \gamma - n + n, A_{q'}) \to \mathsf{loop}) \to \mathsf{loop}$ for a transition rule $q, Z_0 \mapsto q', m, n$ in δ (*). This means that $M = xM_1M_2$ where M_1 is of type $B_{Z,0}((\gamma - n)[\gamma := A_0], A_q)$, the term M_2 is of type $C(L_0^0 + m, (\gamma - n + n)[\gamma := A_0], A_{q'}) \to \mathsf{loop}$.

As $B_{Z,0}((\gamma - n)[\gamma := A_0], A_q) = L_0^0 \to A_2 \to A_q \to p$, the term $M_1 = \lambda x_1 x_2 x_3.M_1'$ where $\Gamma_{\mathcal{A}}, \Sigma, x_1 : L_0^0, x_2 : A_2, x_3 : A_q \vdash_{\mathrm{ML}} M_1' : p$ with $A_2 = (\gamma - n)[\gamma := A_0]$. Only elements of Σ have target p so $M_1' = yM_1''M_2''M_3''$ for some $y : A\langle q_y, l_0, l_1\rangle = L_0^{l_0} \to L_1^{l_1} \to A_{q_y} \to p$, M_1'' of type $L_0^{l_0}$, M_2'' of type $L_1^{l_1}$ and M_3'' of type A_{q_y}. The only type in the context that has target of the form $P(\cdots P(\mathsf{0})\cdots)$ is the type of x_1 so $M_1'' = x_1$. Similarly, the only type in the context that has target of the form $P(\cdots P(\mathsf{1})\cdots)$ is the type of x_2 so $M_2'' = x_2$. Again, the only type in the context that can possibly have the target of the form $R(\cdots R(\mathsf{0})\cdots)$ is the type of x_3, so $M_3'' = x_3$ and $A_2 = (\gamma - n)[\gamma := A_0] = R(\cdots R(\mathsf{0})\cdots)$. Consequently $y : A\langle q, 0, l_1\rangle \in \Sigma$ for some number $l_1 \geq 0$. Recall that $\gamma - n$ for $n = -1, 0, 1$ is either γ or $P(\gamma)$ so γ is instantiated either with $A_0 = P^{l_1}(\mathsf{1})$ or $A_0 = P^{l_1 - 1}(\mathsf{1})$ (for $n = -1$). We can now turn our attention to the term M_2. Since its type is $A_g \to \mathsf{loop} = C(L_0^0 + m, (\gamma - n + n)[\gamma := A_0], A_{q'}) \to \mathsf{loop}$, it has the form $\lambda x_1.M_2'$ where $\Gamma, \Sigma, x_1 : A_g \vdash_{\mathrm{ML}} M_2' : \mathsf{loop}$. An analysis of A_g shows that it is actually $A\langle q', 0 + m, l_1 + n\rangle$. As the size of M_2' is less than the size of M, we can apply the induction hypothesis. As a result, we obtain $A\langle q'', k_0, k_1\rangle \in \Sigma$ such that $\mathcal{A}$ halts from $\langle q'', k_0, k_1\rangle$. In case $\langle q'', k_0, k_1\rangle = \langle q', 0 + m, l_1 + n\rangle$, we obtain by the mentioned above rule (*) that $\mathcal{A}$ halts from $\langle q, 0, l_1\rangle$ where $A\langle q, 0, l_1\rangle \in \Sigma$. In case $\langle q'', k_0, k_1\rangle \neq \langle q', 0 + m, l_1 + n\rangle$, we obtain that already $\langle q'', k_0, k_1\rangle \in \Sigma$. In both cases the claim of the current lemma is proved.

In case (4), $x : B_{NZ,0}(\gamma_0, \gamma_1 - n, A_q) \to (C(\gamma_0 + 1 + m, \gamma_1 - n + n, A_{q'}) \to \mathsf{loop}) \to \mathsf{loop}$ for a transition rule $q, NZ_0 \mapsto q', m, n$ in δ (**). This means that $M = xM_1M_2$ where M_1 is of type

$$B_{NZ,0}(\gamma_0[\gamma_0 := A_0], (\gamma_1 - n)[\gamma_1 := A_1], A_q),$$

and the term M_2 is of type

$$C((\gamma_0 + 1 + m)[\gamma_0 := A_0], (\gamma_1 - n + n)[\gamma_1 := A_1], A_{q'}) \to \mathsf{loop}.$$

As $B_{NZ,0}(\gamma_0[\gamma_0 := A_0], (\gamma_1 - n)[\gamma_1 := A_1], A_q) = P(A_0) \to (\gamma_1 - n)[\gamma_1 := A_1] \to A_q \to p$, the term $M_1 = \lambda x_1 x_2 x_3.M_1'$ where $\Gamma_{\mathcal{A}}, \Sigma, x_1 : P(A_0), x_2 : (\gamma_1 - n)[\gamma_1 := A_1], x_3 : A_q \vdash_{\mathrm{ML}} M_1' : p$. Only elements of Σ have target p so $M_1' = yM_1''M_2''M_3''$ for some $y : A\langle q_y, l_0, l_1\rangle = L_0^{l_0} \to L_1^{l_1} \to A_{q_y} \to p$ with $l_0, l_1 \geq 0$, M_1'' of type $L_0^{l_0}$, M_2'' of type $L_1^{l_1}$ and M_3'' of type A_{q_y}. The only type in the context that has target of the form $P(\cdots P(\mathsf{0})\cdots)$ is the type of x_1 so $M_1'' = x_1$. Similarly, the only type in the context that has target of the form $P(\cdots P(\mathsf{1})\cdots)$ is the type of x_2 so $M_2'' = x_2$. Again, the only type in the context that can possibly have the target of the form $R(\cdots R(\mathsf{0})\cdots)$ is the type of x_3, so $M_3'' = x_3$. Consequently $y : A\langle q, l_0, l_1\rangle \in \Sigma$, and $l_0 > 0$. Recall that $\gamma_1 - n$ for $n = -1, 0, 1$ is either γ_1 or $P(\gamma_1)$ so γ_1 is instantiated either with $A_1 = P^{l_1}(\mathsf{1})$ or $A_1 = P^{l_1 - 1}(\mathsf{1})$ (for $n = -1$). We can now turn our attention to the term M_2. Since its type is

$$A_g \to \mathsf{loop} = C((\gamma_0 + 1 + m)[\gamma_0 := A_0], (\gamma_1 - n + n)[\gamma_1 := A_1], A_{q'}) \to \mathsf{loop},$$

it has the form $\lambda x_1.M_2'$ where

$$\Gamma, \Sigma, x_1 : A_g \vdash_{\mathrm{ML}} M_2' : \mathsf{loop}.$$

An analysis of A_g shows that it is actually $A\langle q', l_0 + m, l_1 + n\rangle$. As the size of M_2' is less than the size of M, we can apply the induction hypothesis. As a result, we obtain $A\langle q'', k_0, k_1\rangle \in \Sigma$ such that $\mathcal{A}$ halts from $\langle q'', k_0, k_1\rangle$. In case $\langle q'', k_0, k_1\rangle = \langle q', l_0 + m, l_1 + n\rangle$, we obtain by the mentioned above rule (**) that $\mathcal{A}$ halts from $\langle q, l_0, l_1\rangle$ where $A\langle q, l_0, l_1\rangle \in \Sigma$. In case $\langle q'', k_0, k_1\rangle \neq \langle q', l_0 + m, l_1 + n\rangle$, we obtain that already $A\langle q'', k_0, k_1\rangle \in \Sigma$. In both cases the claim of the current lemma is proved. ◀

We can now use these lemmas to prove the undecidability result.

▶ **Theorem 12.** *The program synthesis for* ML *with algebraic types is undecidable.*

Proof. We reduce the halting problem for two-counter automata to the problem of program synthesis. Given an automaton $\mathcal{A}$ we generate the instance $\Gamma_{\mathcal{A}}, \mathsf{start}$ of the program synthesis for ML with algebraic types.

In case there is a (normal form) term M such that $\Gamma_{\mathcal{A}} \vdash_{\mathrm{ML}} M : \mathsf{start}$ we proceed as follows. Since start is atomic, the term M must be of the form $xM_1 \cdots M_n$ for some $x \in \Gamma_{\mathcal{A}}$. The only variable of this kind is $x : (A\langle q_I, 0, 0\rangle \to \mathsf{loop}) \to \mathsf{start}$ and $M = x(\lambda y.M')$ where $\Gamma_{\mathcal{A}}, y : A\langle q_I, 0, 0\rangle \vdash_{\mathrm{ML}} M' : \mathsf{loop}$. We observe that $\Sigma = y : A\langle q_I, 0, 0\rangle$ is faithful for the automaton $\mathcal{A}$. We can apply now Lemma 11 and conclude that $\mathcal{A}$ halts from $\langle q_I, 0, 0\rangle$.

In case $\mathcal{A}$ halts from $\langle q_I, 0, 0\rangle$, we can apply Lemma 10 to the context $\Sigma = y : A\langle q_I, 0, 0\rangle$, which is faithful for the automaton $\mathcal{A}$. As a result, we obtain a term M' such that $\Gamma_{\mathcal{A}}, y : A\langle q_I, 0, 0\rangle \vdash_{\mathrm{ML}} M' : \mathsf{loop}$. This gives us that $\Gamma_{\mathcal{A}} \vdash_{\mathrm{ML}} x(\lambda y.M') : \mathsf{start}$ where $x : (A\langle q_I, 0, 0\rangle \to \mathsf{loop}) \to \mathsf{start} \in \Gamma_{\mathcal{A}}$.

This concludes the reduction and thus the program synthesis for ML with algebraic types is undecidable as the halting problem is. ◀

Algebraic types in functional programming languages use constants that construct values of algebraic types (e.g. `cons` for lists) and destruct them (e.g. fold-like iterators for lists). The construction above does not work when the context Γ above contains constructors and destructors for algebraic types. However, it can be easily corrected when we allow value constructors, but disallow destructors. The proof breaks when we say that the only way to obtain target type of the forms $P^i(\mathsf{0}), P^i(\mathsf{1}), R^i(\mathsf{0})$ is through the use of a variable introduced through one of our six type schemes. In the presence of value constructors we have another option to construct them using them. We can forbid this possibility by taking a slightly different encoding of counters, and states and take there $P^i(\mathsf{0}) \to q, P^i(\mathsf{1}) \to q, R^i(\mathsf{0}) \to q$ for some new type constant q.

It is still possible to have realistic programming scenarios within these constraints. Algebraic types are often hidden in abstract types that make available value constructors, but do not offer destructors. Instead they give the programmers an interface to operate on constructed values without the knowledge of their actual representation.

3.2 Program Synthesis with Restricted Instantiation

Since the problem in its full generality is undecidable we can try to find a reasonable special case, for which the problem becomes decidable. One of the critical features of the reduction in the previous section is the possibility to instantiate type schemes with types of arbitrary size (they are used to match the values of the counters during a run of a simulated automaton). Therefore, we propose to restrict the instantiation.

▶ **Problem 13** (Program Synthesis for ML_1). *Given a context* Γ *and type* A *of* ML_1 *check if there is a term* M *such that* $\Gamma \vdash_{\mathrm{ML}_1} M : A$.

In this paper we show that this problem is EXPSPACE-complete for ML with algebraic types. As the main device to prove this, we use the results for the provability problem in IFOL in the restricted class $\mathbf{N}$ of first-order formulas of Mints-hierarchy [17].

We can now define a transformation $\lfloor\cdot\rfloor$ that transforms a formula from $\mathbf{B}$ to an ML type scheme. For simplicity, we assume that the set of predicates and algebraic type constructors are the same, i.e. *Preds* = *TAlg*. We assume that there is an injective correspondence between variables in $Vars_{IFOL}$ and type variables in $TVars$. The result of this correspondence for a variable X is written α_X. On other formulas of $\mathbf{B}$ it gives

$$\lfloor P(X_1,\ldots,X_n)\rfloor = P(\alpha_{X_1},\ldots,\alpha_{X_n}),\quad \lfloor\varphi_1\to\varphi_2\rfloor = \lfloor\varphi_1\rfloor\to\lfloor\varphi_2\rfloor,\quad \lfloor\forall X\,\varphi\rfloor = \forall\alpha_X.\lfloor\varphi\rfloor.$$

This definition naturally extends to contexts. We can also define its extension to formulas of $\mathbf{N}$. For a formula $\varphi = \varphi_1\to\cdots\to\varphi_n\to P(X_1,\ldots,X_n)$ we define (with a slight abuse of the target language) the type

$$\lfloor\varphi\rfloor = \lfloor\varphi_1\rfloor\to\cdots\to\lfloor\varphi_n\rfloor\to\lfloor P(X_1,\ldots,X_n)\rfloor.$$

(Note that this is actually a System F type the inhabitation of which is equivalent to the inhabitation of the type $\lfloor P(X_1,\ldots,X_n)\rfloor$ in the initial context extended with $x_1 : \lfloor\varphi_1\rfloor,\ldots,\lfloor\varphi_n\rfloor$.) Committing another small abuse of notation, we define $\lfloor\cdot\rfloor$ on proof terms.

$$\lfloor x\rfloor = x,\ \lfloor\lambda x:\varphi.M\rfloor = \lambda x.\lfloor M\rfloor,\ \lfloor M_1M_2\rfloor = \lfloor M_1\rfloor\lfloor M_2\rfloor,\ \lfloor\lambda X\,M\rfloor = M,\ \lfloor MX\rfloor = \lfloor M\rfloor.$$

Note that this translation may be viewed as a formula erasure mapping.

Here are the basic properties of the transformation.

▶ **Lemma 14.**

1. *For each formula ϕ and variables $X,Y\in Vars_{IFOL}$ $\lfloor\phi[X:=Y]\rfloor = \lfloor\phi\rfloor[\alpha_X:=\alpha_Y]$.*
2. *For each proof term M in normal form the term $\lfloor M\rfloor$ is also in normal form.*

Proof. Straightforward induction over the formula ϕ in (1) and term M in (2). ◀

▶ **Lemma 15.** *For each Γ with formulae from $\mathbf{\Pi}_1\cap\mathbf{B}$ only and $\varphi\in\mathbf{O}$ it holds that for each proof term M in normal form $\Gamma\vdash_{\mathrm{IFOL}} M:\varphi$ if and only if $\lfloor\Gamma\rfloor\vdash_{\mathrm{ML}_1}\lfloor M\rfloor:\lfloor\varphi\rfloor$.*

Proof.

(⇒) The proof is by induction over the derivation for $\Gamma\vdash_{\mathrm{IFOL}} M:\varphi$ with cases depending on the last rule used. The interesting case is when the last rule is $(\forall E)$ then $M = xX_1\cdots X_m$ and the judgement $\Gamma\vdash xX_1\cdots X_m:\varphi$ as well as $\Gamma\vdash x:\forall Y_1\ldots Y_m.\varphi[X_1:=Y_1,\ldots X_m:=Y_m]$ are derivable in intuitionistic first-order logic. This means that $x:\forall Y_1\ldots Y_m.\varphi[X_1:=Y_1,\ldots X_m:=Y_m]\in\Gamma$. Observe that $\lfloor\forall Y_1\ldots Y_m.\varphi[X_1:=Y_1,\ldots,X_m:=Y_m]\rfloor = \forall\alpha_{Y_1}\ldots\alpha_{Y_m}.\lfloor\varphi[X_1:=Y_1,\ldots,X_m:=Y_m]\rfloor$ and consequently that

$$x:\forall\alpha_{Y_1}\ldots\alpha_{Y_m}.\lfloor\varphi[X_1:=Y_1,\ldots,X_m:=Y_m]\rfloor\in\lfloor\Gamma\rfloor.$$

We can now use the (var) rule of ML_1 with the instantiation $S = [\alpha_{Y_1}:=\alpha_{X_1},\ldots,\alpha_{Y_m}:=\alpha_{X_m}]$ since this instantiation can be used to obtain

$$S(\lfloor\varphi[X_1{:=}Y_1,\ldots,X_m{:=}Y_m]\rfloor)\ \mathsf{inst}_1\ \forall\alpha_{Y_1}\ldots\alpha_{Y_m}.\lfloor\varphi[X_1{:=}Y_1,\ldots,X_m{:=}Y_m]\rfloor.$$

As a result, we obtain $\lfloor\Gamma\rfloor\vdash_{\mathrm{ML}_1} x:S(\lfloor\varphi[X_1:=Y_1,\ldots,X_m:=Y_m]\rfloor)$, which is actually $\lfloor\Gamma\rfloor\vdash_{\mathrm{ML}_1} x:\lfloor\varphi\rfloor$ as $\lfloor\varphi\rfloor = S(\lfloor\varphi[X_1{:=}Y_1,\ldots,X_m{:=}Y_m]\rfloor)$.

(⇐) The proof is by induction over derivation for $\lfloor\Gamma\rfloor\vdash_{\mathrm{ML}_1}\lfloor M\rfloor:\lfloor\varphi\rfloor$. The interesting case is when the last rule is (var) then our judgement has the form $\lfloor\Gamma\rfloor, x:\lfloor\psi\rfloor\vdash x:\lfloor\varphi\rfloor$ for some ψ, and it holds that $\lfloor\psi\rfloor\ \mathsf{inst}_1\ \lfloor\varphi\rfloor$. Since $\psi\in\mathbf{\Pi}_1\cap\mathbf{B}$ and $\varphi\in\mathbf{O}$, the formula ψ is $\forall X_1\ldots X_n.\psi_0$, where ψ_0 is quantifier-free and φ is quantifier-free. This implies that $\lfloor\psi\rfloor = \forall\alpha_{X_1}\ldots\alpha_{X_n}.\lfloor\psi_0\rfloor$. Moreover, relation inst_1 means that $\lfloor\psi_0\rfloor[\alpha_{X_1}:=\beta_1,\ldots,\alpha_{X_n}:=\beta_n] = \lfloor\varphi\rfloor$ where $\beta_1,\ldots,\beta_n$ are type variables. For those of the variables that occur in $\mathrm{FTV}(\lfloor\varphi\rfloor)$ we may assume that are of the form α_Y. For other ones we can also assume they have analogous form. Therefore, we have $\lfloor\psi_0\rfloor[\alpha_{X_1} := \alpha_{Y_1}, \ldots, \alpha_{X_n} := \alpha_{Y_n}] = \lfloor\varphi\rfloor$ for some $Y_1, \ldots, Y_n \in \mathit{Vars}_{\mathit{IFOL}}$. We can now use the (var) rule of intuitionistic first-order logic to obtain $\Gamma, x : \psi \vdash x : \psi$ and then subsequently n times the $(\forall E)$ rule with substitutions of the form $[X_i := Y_i]$ to obtain $\Gamma, x : \psi \vdash xY_1 \cdots Y_n : \psi_0[X_1 := Y_1, \ldots, X_n := Y_n]$, which is the required judgement as $\lfloor\psi_0[X_1 := Y_1, \ldots, X_n := Y_n]\rfloor = \lfloor\psi_0\rfloor[\alpha_{X_1} := \alpha_{Y_1}, \ldots, \alpha_{X_n} := \alpha_{Y_n}]$ by Lemma 14 and $\lfloor xY_1 \cdots Y_n\rfloor = x$ by definition of $\lfloor\cdot\rfloor$. ◀

The lemma above helps in giving the proof of EXPSPACE-hardness. One might be tempted to give a proof that the problem is in EXPSPACE also through a translation argument. However, such reasoning would be complicated as the arguments of type constructors need not be type variables. Therefore, we present a direct proof.

First, we need to know how type substitutions affect inferences in ML.

▶ **Lemma 16.** *If* $\Gamma \vdash N : A$ *in either* ML *or* ML_1 *then* $\Gamma[\vec{\alpha} := \vec{\beta}] \vdash N : A[\vec{\alpha} := \vec{\beta}]$ *in* ML *or* ML_1 *respectively.*

Proof. Straightforward induction over the term N. ◀

▶ **Lemma 17.** *Let* $\mathcal{C}$ *be the set of type constants that occur in* Γ, A*. If* N *is in normal form and* $\Gamma \vdash N : A$ *in* ML_1 *where* $\mathcal{C} \cup \mathrm{FTV}(\Gamma, A) \subseteq \mathcal{V}$ *for some non-empty set* $\mathcal{V}$ *then all type instantiations in the derivation can be restricted to use atoms in* $\mathcal{V}$*.*

Proof. The proof is by induction over the term N.

In case N is a variable the proof is obvious. In case $N = \lambda x.N'$, the type A is $A_1 \to A_2$ and the inference must end with the $(\to I)$ rule. This reduces the problem to the one for the judgement $\Gamma, x : A_1 \vdash N' : A_2$, for which we can apply the induction hypothesis and then obtain our conclusion.

In case $N = xN_1 \ldots N_k$, the inference must start with the (var) rule for $x : \sigma \in \Gamma$ where $\sigma = \forall\vec{\gamma}.A_1 \to \cdots \to A_k \to B$ and $\Gamma \vdash N_i : A_i[\vec{\gamma} := \vec{A}]$ where $\vec{A}$ are atomic. By Lemma 16 we obtain $\Gamma \vdash N_i : A_i[\vec{\gamma} := \vec{A}']$ for $i = 1, \ldots, k$ where $\vec{A}'$ differs from $\vec{A}$ only on positions that are outside of $\mathcal{V}$ and has a fixed element α_0 of $\mathcal{V}$ there. Therefore the atomic instantiation with $\vec{A}'$ can be used in the initial (var) rule instead of $\vec{A}$, which gives the required conclusion after application of the induction hypothesis to arguments N_i for $i = 1, \ldots, k$. ◀

▶ **Lemma 18.** *The program synthesis problem for* ML_1 *is in EXPSPACE.*

Proof. Given a context Γ and type A we use a simple generalisation of the Ben-Yelles algorithm [20]. Lemma 17 implies that type inference for a normal program N of type A above can be restricted to use only type atoms from the set $\mathcal{A} = \mathcal{C} \cup \mathrm{FTV}(A, \Gamma) \cup \{\alpha_0\}$, where $\mathcal{C}$ is the set of type constants in A, Γ. Note that the type variable α_0 guarantees that $\mathcal{A}$ is not empty. Therefore the algorithm needs only to consider judgements $\Gamma' \vdash M : B$ where all type atoms are in $\mathcal{A}$. It should be clear that the number of different types in Γ' is at most exponential in the size n of A, Γ. (A type scheme that generalises m variables has at most m^n instances.) Using the same argument as for simply typed lambda calculus in the Ben-Yelles algorithm we obtain an alternating exponential time exponential algorithm. ◀

▶ **Theorem 19.** *The program synthesis problem for* ML_1 *is EXPSPACE-complete.*

Proof. We can now exploit Theorem 5 in the context of the program synthesis problem. The EXPSPACE-hardness is the result of this theorem combined with the reduction presented in Lemma 15. The fact that the problem is in EXPSPACE is again the the result of the theorem combined this time with the construction presented in Lemma 18. ◀

4 Conclusions and Further Work

We presented an initial study on program synthesis for functional programs with libraries. The goal of the constructions presented here is not only to demonstrate undecidability or a particular complexity. They also allow us to better understand intricate difficulties of program synthesis. In particular, the undecidability proof works because the depth of instantiation is not bounded. This makes it reasonable to restrict it in practical procedures or heuristics for program synthesis. We make restriction of this kind in our analysis for ML with instantiations restricted to atomic types. The EXPSPACE-hardness proof there relies on the translation from the core of the fragment Σ_1 of intuitionistic first-order logic. This translation is direct so the reason for the high complexity in the logic is the same as in ML, namely, the number of quantifiers in front of a type scheme occurs in the exponent while the number of atoms determines the base of exponentiation. This is also a hint on the heuristics for program synthesis so that they should give ways to restrict type schemes in this fashion.

The discussion of this paper goes beyond the solution of Djinn, which handles polymorphic functions, but does not allow for any instantiation. One improvement of the current work over Djinn is that we show that handling of instantiation may lead to undecidability, but also we show how to handle some interesting instantiation cases.

There are still some interesting questions that are left open here. First, it is not clear what is the complexity of program synthesis with context for ML programs with no algebraic types. Second, the complexity of program synthesis for the case where the context contains only types of the algebraic type constructors and destructors may be different than the complexities obtained in this work. It is also appealing to investigate the impact of the size of the instantiation on the complexity of the problem in the spirit of the bounded combinatory logic studied by Düdder et al [7]. These theoretical questions can be complemented by investigations in more practical directions. For example, which syntactic forms of types give rise to small number of inhabitants? What are the situations when the resulting terms are small enough to be further examined by programmers in reasonable time?

References

1 Lennart Augustsson, 2005. URL: `https://github.com/augustss/djinn`.

2 Coq Development Team. *The Coq Proof Assistant Reference Manual V8.4*, March 2012. URL: `http://coq.inria.fr/distrib/V8.4/refman/`.

3 Thierry Coquand and Gerard Huet. The calculus of constructions. *Information and Computation*, pages 95–120, 1988.

4 Luis Damas and Robin Milner. Principal type-schemes for functional programs. In *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 207–212, New York, NY, USA, 1982. ACM.

5 Edsger Wybe Dijkstra. *A Discipline of Programming.* Prentice Hall, 1976.

6 Catherine Dubois. Proving ML type soundness within Coq. In Mark Aagaard and John Harrison, editors, *Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000 Portland, OR, USA, August 14–18, 2000 Proceedings*, pages 126–144, Berlin, Heidelberg, 2000. Springer.

7 Boris Düdder, Moritz Martens, Jakob Rehof, and Pawel Urzyczyn. Bounded Combinatory Logic. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) – 26th International Workshop/21st Annual Conference of the EACSL*, volume 16 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 243–258, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

8 J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmetique d'ordre supérieur.* PhD thesis, Université Paris VII, 1972.

9 Roger Hindley. The principal type-scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1969.

10 Pierre Letouzey. Coq Extraction, an Overview. In C. Dimitracopoulos A. Beckmann and B. Löwe, editors, *Logic and Theory of Algorithms, Fourth Conference on Computability in Europe, CiE 2008*, volume 5028 of *Lecture Notes in Computer Science*. Springer-Verlag, 2008.

11 Samuel Linial and E. L. Post. Recursive unsolvability of the deducibility, Tarski's completeness, and independence of axioms problems of propositional calculus. *Bulletin of the American Mathematical Society*, vol. 55, p. 50, 1949. Abstract.

12 Per Martin-Löf. Constructive mathematics and computer programming. In *Proceedings of the Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175. North-Holland, 1982.

13 Marvin L. Minsky. Recursive unsolvability of Post's problem of "Tag" and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, 1961.

14 T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL. A Proof Assistant for Higher-Order Logic*. Springer, 2002.

15 Christine Paulin-Mohring. Fω's programs from proofs in the calculus of constructions. In *Sixteenth Annual ACM Symposium on Principles of Programming Languages*. ACM Press, 1989.

16 J. C. Reynolds. Towards a theory of type structure. In Ehring et al., editor, *Programming Symposium, Proceedings Colloque Sur La Programmation*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer, 1974.

17 Aleksy Schubert, Paweł Urzyczyn, and Konrad Zdanowski. On the Mints hierarchy in first-order intuitionistic logic. In A. Pitts, editor, *Foundations of Software Science and Computation Structures 2015*, volume 9034 of *Lecture Notes in Computer Science*, pages 451–465. Springer, 2015. doi:10.1007/978-3-662-46678-0_29.

18 Helmut Schwichtenberg. The MINLOG system. URL: http://www.mathematik.uni-muenchen.de/~logik/minlog/.

19 W. E. Singletary. Many-one degrees associated with partial propositional calculi. *Notre Dame J. Formal Logic*, 15(2):335–343, 04 1974.

20 M.H. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism*, volume 149. Elsevier, 2006.

21 Richard Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9:67–72, 1979.

22 Nicholas Wirth. *Systematic Programming: An Introduction.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1973.

Classical Extraction in Continuation Models[*]

Valentin Blot

Department of Computer Science, University of Bath, Bath, United Kingdom
v.blot@bath.ac.uk

Abstract

We use the control features of continuation models to interpret proofs in first-order classical theories. This interpretation is suitable for extracting algorithms from proofs of Π_2^0 formulas. It is fundamentally different from the usual direct interpretation, which is shown to be equivalent to Friedman's trick. The main difference is that atomic formulas and natural numbers are interpreted as distinct objects. Nevertheless, the control features inherent to the continuation models permit extraction using a special "channel" on which the extracted value is transmitted at toplevel without unfolding the recursive calls. We prove that the technique fails in Scott domains, but succeeds in the refined setting of Laird's bistable bicpos, as well as in game semantics.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages: Denotational semantics, F.4.1 Mathematical Logic

Keywords and phrases Extraction, Classical Logic, Control Operators, Game Semantics

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.13

1 Introduction

Game semantics appeared simultaneously in [13, 21, 2] and provides precise models for PCF. These seminal works started a new line of research which led among other things to models of languages with higher-order references [1] and control operators [18]. These interesting features were obtained by removing some requirements of the original model: innocence for references, and well-bracketing for control operators. This interesting property means that the model is at first a model of a language with side effects, that we can then constrain to eliminate non-functional behaviors. In this work, we are interested into giving computational content to formulas through realizability, so it is an interesting feature to have an expressive language in which we can write simple realizers for complicated formulas.

Realizability is a way to relate programs and formulas invented by Kleene [14]. To each formula A in a given language $\mathcal{L}$ we associate a set $|A|$ of realizers in a given programming language $\mathcal{P}$. The realizability interpretation $|\_|$ is usually defined by induction on the formula, for example $M \in |A \Rightarrow B|$ is typically defined as: for all $N \in |A|$, $M\,N \in |B|$, where $M\,N$ is the application of the argument N to the program M in $\mathcal{P}$. The property of adequacy is obtained by the definition of a theory for $\mathcal{L}$ which is correct with respect to the realizability interpretation. Adequacy allows the mapping of any proof π of a formula A in the chosen theory to some $[\pi] \in |A|$. Finally, if the realizability interpretation is sufficiently well behaved, then a realizer M of a formula $\forall x\,\exists y\,A\,\{x, y\}$ is such that for any element a in the model, $A\,\{a, M\,a\}$ holds. This property combined with adequacy gives extraction: from a proof of a formula A one can obtain a program M such that $A\,\{a, M\,a\}$ holds for any element a.

[*] Research supported by the UK EPSRC grant EP/K037633/1.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 13; pp. 13:1–13:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the context of classical logic, there are mainly two extraction methods. The first one relies on Gödel's negative translation which maps a proof of $\forall x\, \exists y\, P\{x, y\}$ (where P is an atomic predicate) in classical arithmetic to a proof of $\forall x\, \neg\forall y\, \neg P\{x, y\}$ in intuitionistic arithmetic. Then Friedman's trick [8] replaces $\bot$ with $\exists y\, P\{x, y\}$, to obtain an intuitionistic proof of $\forall x\, \exists y\, P\{x, y\}$. This turns any extraction method in intuitionistic arithmetic into an extraction method for Π^0_2 formulas in classical arithmetic. Refinements of this technique have been studied in [5], and bar recursive interpretations of the negative translation of the axiom of choice in this setting have been given in [4, 6].

The second method uses control operators [9] like scheme's call/cc. If $\mathcal{P}$ has control features, then adequacy can hold for a classical theory. Extraction with this method is obtained by taking a non-empty set of realizers of $\bot$. The first realizability models for classical logic with control operators were built by Krivine [16] and use an untyped λ-calculus extended with call/cc as programming language, and second order Peano arithmetic as logical theory. They have also been later extended with the axiom of dependent choice, by the addition of particular instructions to the language of realizers [15]. Krivine's realizability has also been related to Friedman's A-translation in [22, 20], and several extraction methods in this setting have been studied [20, 24].

In this work we define a variant of the second method and introduce control features in $\mathcal{P}$ by working in simply-typed $\lambda\mu$-calculus [23]. Precisely, we take $\mathcal{P}$ to be a model of $\lambda\mu$-calculus, that is, a category of continuations [12]. Working in a model rather than in a language can be interesting when one wants features that fit more naturally in a model than in the syntax. An example is the realization of the axiom of choice using the bar recursion operator [7], which requires the existence of all the first-class functions in $\mathcal{P}$. Taking the game semantics model for $\mathcal{P}$ also gives access to references in the realizers.

In Section 2 we first fix the logical framework, that is first-order logic. We then describe the mapping of intuitionistic (resp. classical) proofs to λ-calculus (resp. $\lambda\mu$-calculus) and the realizability interpretation. We describe Friedman's trick which turns extraction for intuitionistic theories into extraction for classical theories, then we describe how control operators can be used to interpret directly classical logic, and finally we explain why the two methods are equivalent when working in a model. In Section 3 we present our method of extraction for classical logic in a category of continuations and explain how it provides a simpler interpretation. Finally we explain why this technique fails in models based on Scott domains, but works in the model of unbracketed Hyland-Ong-style game semantics.

2 Friedman's trick and direct interpretation

In this section, we fix the logical system we will be working with. Then we present on one hand the indirect interpretation through negative translation and Friedman's trick and on the other hand the direct interpretation with control operators. Finally we explain why these two interpretations are the same when we work in a model.

2.1 The logical system

The logical systems under consideration in this work are first-order minimal logic, $\mathfrak{M}$, and first-order classical logic, $\mathfrak{C}$. $\mathfrak{C}$ is simply an extension of $\mathfrak{M}$ with double-negation elimination. The common syntax of first-order terms and formulas of $\mathfrak{M}$ and $\mathfrak{C}$ is the following:

$$t, u \ ::= \ x \mid \mathsf{f}\left(\vec{t}\right) \qquad A, B \ ::= \ P\left(\vec{t}\right) \mid \bot \mid A \Rightarrow B \mid A \wedge B \mid \forall x\, A \mid A \vee B \mid \exists x\, A$$

$$\frac{}{\Gamma, p : A \vdash p : A} \qquad \frac{\Gamma, p : A \vdash M : B}{\Gamma \vdash \lambda p.M : A \Rightarrow B} \qquad \frac{\Gamma \vdash M : B \Rightarrow A \qquad \Gamma \vdash N : B}{\Gamma \vdash MN : A}$$

$$\frac{\Gamma \vdash M_1 : A_1 \qquad \Gamma \vdash M_2 : A_2}{\Gamma \vdash \langle M_1, M_2 \rangle : A_1 \wedge A_2} \qquad \frac{\Gamma \vdash M : A_1 \wedge A_2}{\Gamma \vdash \pi_i \, M : A_i}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \Lambda x.M : \forall x\, A} \; (x \notin \mathrm{FV}(\Gamma)) \qquad \frac{\Gamma \vdash M : \forall x\, A}{\Gamma \vdash M \lfloor t \rfloor : A\{t/x\}}$$

$$\frac{\Gamma \vdash M : A_i}{\Gamma \vdash \mathtt{in}_i\, M : A_1 \vee A_2} \qquad \frac{\Gamma \vdash M : A_1 \vee A_2 \qquad \Gamma, p : A_1 \vdash N_1 : B \qquad \Gamma, p : A_2 \vdash N_2 : B}{\Gamma \vdash \mathsf{case}\, M\, [\mathtt{in}_1\, p \mapsto N_1, \mathtt{in}_2\, p \mapsto N_2] : B}$$

$$\frac{\Gamma \vdash M : A\{t/x\}}{\Gamma \vdash \mathsf{ex}\, [t, M] : \exists x\, A} \qquad \frac{\Gamma \vdash M : \exists x\, A \qquad \Gamma, p : A \vdash N : B}{\Gamma \vdash \mathsf{dest}\, M\, \mathsf{as}\, \mathsf{ex}\, [x, p]\ \mathsf{in}\, N : B} \; (x \notin \mathrm{FV}(\Gamma, B))$$

Figure 1 Rules for $\mathfrak{M}$.

where f (resp. P) ranges over a set of function (resp. predicate) symbols given with their arity. Quantification has precedence over other connectives and negation is encoded as usual: $\neg A \triangleq A \Rightarrow \bot$. The proofs and proof terms of $\mathfrak{M}$ are defined by the rules of Figure 1, where Γ stands for a sequence of pairs $p : A$ where p is a proof variable, for which we allow implicit re-ordering.

For $\mathfrak{C}$, we simply add the rules:

$$\frac{}{\Gamma \vdash \mathsf{dne} : \neg\neg A \Rightarrow A}$$

Provability in $\mathfrak{M}$ (resp. $\mathfrak{C}$) will be denoted $\vdash_m$ (resp. $\vdash_c$). We will sometimes write $\Gamma \vdash_m A$ (resp. $\Gamma \vdash_c A$) if we're not interested in the proof term, and we may write $X \vdash_m A$ (resp. $X \vdash_c A$) for some set X, which means $\Gamma \vdash_m A$ (resp. $\Gamma \vdash_c A$) for some finite sequence Γ of formulas of X. Since contraction is derivable and weakening is admissible, we will use these implicitly.

2.2 Mapping proofs to terms

We will use two programming languages that share a common ground to interpret $\mathfrak{M}$ and $\mathfrak{C}$. This common ground is simply-typed λ-calculus with products and unit types, and one base type ι. The set of variables of this λ-calculus is the union of first-order variables $x, y, \ldots$ and proof variables $p, q, \ldots$, this union being ranged over with $e, f, \ldots$. There is also one constant f of type $\iota \to \ldots \to \iota \to \iota$ ($n+1$ times) for each first-order constant f of arity n. The syntax of types and terms of this common ground is as follows:

$$T, U \;::=\; \iota \mid T \to U \mid 1 \mid T \times U \qquad M, N \;::=\; \mathsf{f} \mid \lambda e.M \mid M\,N \mid \langle\rangle \mid \langle M, N \rangle \mid \pi_1\, M \mid \pi_2\, M$$

From this common ground, the first language we consider is λ^+, in which we will interpret $\mathfrak{M}$. λ^+ is obtained by adding sum types to the common ground:

$$T, U \;::=\; \ldots \mid T + U \qquad M, N \;::=\; \ldots \mid \mathsf{in}_1\, M \mid \mathsf{in}_2\, M \mid \mathsf{case}\, M\, \{\mathsf{in}_1\, e \mapsto N_1 \mid \mathsf{in}_2\, e \mapsto N_2\}$$

The second language is $\lambda\mu$, in which we will interpret $\mathfrak{C}$, and which is obtained by adding to the common ground an empty type, control features and μ-variables:

$$T, U \;::=\; \ldots \mid \diamond \qquad M, N \;::=\; \ldots \mid \mu\alpha.M \mid [\alpha]\, M$$

$$(A \Rightarrow B)^* \triangleq A^* \to B^* \qquad (A \wedge B)^* \triangleq A^* \times B^* \qquad (\forall x\, A)^* \triangleq \iota \to A^*$$

$$\bot^{*m} \triangleq 1 \qquad P\left(\vec{t}\right)^{*m} \triangleq 1 \qquad (A \vee B)^{*m} \triangleq A^{*m} + B^{*m} \qquad (\exists x\, A)^{*m} \triangleq \iota \times A^{*m}$$

$$\bot^{*c} \triangleq \diamond \qquad P\left(\vec{t}\right)^{*c} \triangleq \diamond \to \diamond \qquad (A \vee B)^{*c} \triangleq (\neg\,(\neg\, A \wedge \neg\, B))^{*c} \qquad (\exists x\, A)^{*c} \triangleq (\neg\,\forall x\, \neg\, A)^{*c}$$

Figure 2 Mapping formulas to types.

$$p^* \triangleq p \qquad (\lambda p.M)^* \triangleq \lambda p.M^* \qquad (M\,N)^* \triangleq M^*\,N^* \qquad (\Lambda x.M)^* \triangleq \lambda x.M^*$$

$$(M\ \lfloor t \rfloor)^* \triangleq M^*\,t^* \qquad \langle M, N\rangle^* \triangleq \langle M^*, N^*\rangle \qquad (\pi_i\, M)^* \triangleq \pi_i\, M^* \qquad \mathsf{dne}^{*c} \triangleq \lambda p.\mu\alpha.p\,(\lambda q.\,[\alpha]\,q)$$

$$(\mathtt{in}_i\, M)^{*m} \triangleq \mathsf{in}_i\, M^{*m} \qquad (\mathsf{ex}\,[t, M])^{*m} \triangleq \langle t^*, M^{*m}\rangle$$

$$(\mathtt{in}_i\, M)^{*c} \triangleq \lambda p.\pi_i\, p\, M^{*c} \qquad (\mathsf{ex}\,[t, M])^{*c} \triangleq \lambda p.p\, t^*\, M^{*c}$$

$$(\mathtt{case}\, M\,[\mathtt{in}_1\, p \mapsto N_1, \mathtt{in}_2\, p \mapsto N_2])^{*m} \triangleq \mathtt{case}\, M^{*m}\,[\mathtt{in}_1\, p \mapsto {N_1}^{*m}, \mathtt{in}_2\, p \mapsto {N_2}^{*m}]$$

$$(\mathtt{case}\, M\,[\mathtt{in}_1\, p \mapsto N_1, \mathtt{in}_2\, p \mapsto N_2])^{*c} \triangleq \mathsf{dne}^{*c}\,(\lambda q.M^{*c}\,\langle \lambda p.q\, {N_1}^{*c}, \lambda p.q\, {N_2}^{*c}\rangle)$$

$$(\mathtt{dest}\, M\, \mathsf{as}\, \mathsf{ex}\,[x, p]\ \mathtt{in}\, N)^{*m} \triangleq (\lambda xp.N^{*m})\,(\pi_1\, M^{*m})\,(\pi_2\, M^{*m})$$

$$(\mathtt{dest}\, M\, \mathsf{as}\, \mathsf{ex}\,[x, p]\ \mathtt{in}\, N)^{*c} \triangleq \mathsf{dne}^{*c}\,(\lambda q.M^{*c}\,(\lambda xp.q\, N^{*c}))$$

Figure 3 Mapping proof terms to $\lambda^+/\lambda\mu$-terms.

We work in a call-by-name setting and refer to [12, 25] for the typing rules and equational theory of $\lambda\mu$.

Note that terms of λ^+ and $\lambda\mu$ are not the same as the proof terms of Figure 1. In particular, proof terms have no operational or denotational semantics, but provide a convenient way to manipulate proofs. Proofs terms will now be mapped to terms of λ^+ or $\lambda\mu$. First, we map each formula A to a type A^* of either λ^+ or $\lambda\mu$. When the interpretation is different in $\mathfrak{M}$ and $\mathfrak{C}$ we will write A^{*m} or A^{*c}. The mapping is defined in Figure 2.

In the case of $\mathfrak{C}$, since positive connectives are known to raise issues on the computational level when combined with classical logic [11] (we also give a concrete example at the end of Section 2.3) we use a negative encoding of connectives $\vee$ and $\exists$. First-order terms (which are common to $\mathfrak{M}$ and $\mathfrak{C}$) are mapped to terms of $\lambda^+ \cap \lambda\mu$ by $x^* \triangleq x$ and $(\mathsf{f}\,(t_1, \ldots, t_n))^* \triangleq \mathsf{f}\, {t_1}^* \ldots {t_n}^*$, and proof terms of $\mathfrak{M}$ (resp. $\mathfrak{C}$) are mapped to terms of λ^+ (resp. $\lambda\mu$) as in Figure 3. The elimination of the encoded connectives in $\mathfrak{C}$ is made possible by classical logic and control features of $\lambda\mu$.

A proof term $p_1 : A_1, \ldots, p_n : A_n \vDash_{\mathfrak{m}} M : A$ (resp. $p_1 : A_1, \ldots, p_n : A_n \vDash_{\mathfrak{c}} M : A$) with $\mathrm{FV}\,(A_1, \ldots, A_n, A) = \{x_1, \ldots, x_m\}$ is therefore mapped to a term $M^{*m} : A^{*m}$ (resp. $M^{*c} : A^{*c}$) of λ^+ (resp. $\lambda\mu$) with free variables among $p_1 : {A_1}^*, \ldots, p_n : {A_n}^*, x_1 : \iota, \ldots, x_m : \iota$, and no free μ-variable (in the case of $\mathfrak{C}$).

$$\left|P\left(\vec{t}\right)\right|_m \triangleq \begin{cases} \{\langle\rangle\} & \text{if } \vec{t}^* \in \{\!|P|\!\} \\ \emptyset & \text{otherwise} \end{cases} \qquad |A_1 \wedge A_2| \triangleq \{\phi |\ \pi_1\, \phi \in |A_1| \text{ and } \pi_2\, \phi \in |A_2|\}$$

$$\left|P\left(\vec{t}\right)\right|_c \triangleq \begin{cases} |\bot \Rightarrow \bot|_c & \text{if } \vec{t}^* \in \{\!|P|\!\} \\ \{\phi |\ \text{if } \psi \in \mathcal{C}\,(1, \diamond) \text{ then } \phi\, \psi \in |\bot|_c\} & \text{otherwise} \end{cases}$$

$$|A \Rightarrow B| \triangleq \{\phi |\ \text{if } \psi \in |A| \text{ then } \phi\, \psi \in |B|\} \qquad |\forall x\, A| \triangleq \{\phi |\ \text{if } \psi \in \langle\!|\iota|\!\rangle \text{ then } \phi\, \psi \in |A\,\{\psi/x\}|\}$$

$$|A_1 \vee A_2|_m \triangleq \{\mathsf{in}_1\, \phi |\ \phi \in |A_1|_m\} \cup \{\mathsf{in}_2\, \phi |\ \phi \in |A_2|_m\} \qquad |A_1 \vee A_2|_c \triangleq |\neg\,(\neg\, A_1 \wedge \neg\, A_2)|_c$$

$$|\exists x\, A|_m \triangleq \{\phi |\ \pi_1\, \phi \in \langle\!|\iota|\!\rangle \text{ and } \pi_2\, \phi \in |A\,\{\pi_1\, \phi/x\}|_m\} \qquad |\exists x\, A|_c \triangleq |\neg\, \forall x\, \neg\, A|_c$$

Figure 4 The realizability interpretation.

2.3 Typed realizability

We now describe the realizability semantics of $\mathfrak{M}$ and $\mathfrak{C}$. Let $\mathcal{C}$ be a cartesian closed category. We also suppose that $\mathcal{C}$ has coproducts in the case of $\mathfrak{M}$, or that it is a category of continuations (see [12, 25] and Section 2.6) in the case of $\mathfrak{C}$. Every type of λ^+ (for $\mathfrak{M}$) or $\lambda\mu$ (for $\mathfrak{C}$) is interpreted as an object of $\mathcal{C}$ in the standard way, from a chosen interpretation of ι. We will therefore consider types as objects of $\mathcal{C}$. Similarly, we suppose given a morphism for each constant f of $\lambda^+ \cap \lambda\mu$ (remember from Section 2.2 that there is one such constant for each first-order constant f), so that every term of λ^+ or $\lambda\mu$ is interpreted as a morphism in the homset corresponding to its context. We will therefore also consider terms as morphisms in $\mathcal{C}$, and we will use the syntax of λ^+ and $\lambda\mu$ to manipulate morphisms with domain 1. In order to define the realizability values of the formulas, we first fix a set $\langle\!|\iota|\!\rangle \subseteq \mathcal{C}\,(1, \iota)$ such that for every closed first-order term t, $t^* \in \langle\!|\iota|\!\rangle$. This set represents the elements of the model on which we quantify. Typically, if $\mathcal{C}\,(1, \iota)$ is a domain of natural numbers with a bottom element, $\langle\!|\iota|\!\rangle$ would be the set of natural numbers (the domain minus the bottom element). We also fix for every predicate P of arity n a set $\{\!|P|\!\} \subseteq \langle\!|\iota|\!\rangle^n$ of n-tuples satisfying the predicate. The set of realizers of a closed formula with parameters in $\langle\!|\iota|\!\rangle$ is then a set of morphisms $|A| \subseteq \mathcal{C}\,(1, A^*)$. When the interpretation is different in $\mathfrak{M}$ and $\mathfrak{C}$ we will again write $|A|_m$ or $|A|_c$. We take $|\bot|_m = \emptyset$, but $|\bot|_c$ is a parameter of the realizability interpretation. The interpretation is defined in Figure 4.

It is worth noting here that taking $|\bot|_c = \emptyset$ gives a degenerated model, in which $|A|_c$ is either empty or the full homset $\mathcal{C}\,(1, A^*)$. Indeed, suppose that $|A|_c \neq \emptyset$, and let $\phi \in \mathcal{C}\,(1, A^*)$. First, since $|A|_c \neq \emptyset$ and $|\bot|_c = \emptyset$, we have $|\neg\, A|_c = \emptyset$, and therefore $|\neg\,\neg\, A|_c = \mathcal{C}\left(1, (\neg\,\neg\, A)^*\right)$. But since $\lambda p.p\, \phi \in \mathcal{C}\left(1, (\neg\,\neg\, A)^*\right)$, we get $\lambda p.p\, \phi \in |\neg\,\neg\, A|_c$, and then $\mathsf{dne}^*\,(\lambda p.p\, \phi) = \phi \in |A|_c$ by adequacy (Lemma 1 below). Therefore, if we want to get computational content from classical proofs, we need to consider $|\bot|_c \neq \emptyset$.

We can already state an adequacy lemma for $\mathfrak{M}$ and $\mathfrak{C}$:

▶ **Lemma 1** (Adequacy). *Suppose $\vec{p} : \vec{A} \vdash M : B$ is a proof in $\mathfrak{M}$ or $\mathfrak{C}$ and write $FV\left(\vec{A}, B\right) = \vec{x} = \{x_1, \ldots, x_n\}$. Then for any $\vec{\phi}$ in $\langle\!|\iota|\!\rangle^n$ and any realizers $\vec{\psi} \in \left|\vec{A}\left\{\vec{\phi}/\vec{x}\right\}\right|$, we have $M^*\left\{\vec{\phi}/\vec{x}, \vec{\psi}/\vec{p}\right\} \in \left|B\left\{\vec{\phi}/\vec{x}\right\}\right|$.*

Proof. By induction on M. An interesting case is that of $\mathsf{dne} : \neg\,\neg\, A \Rightarrow A$ in the case of $\mathfrak{C}$ (we will write dne_A in this proof), which is proved by induction on A. If A is atomic, then it is a case analysis. In the other cases, it follows from the observation that in $\mathcal{C}$:

$$\mathsf{dne}_{A\Rightarrow B}{}^{*c} = \lambda pq.\mathsf{dne}_{B}{}^{*c}\left(\lambda r.p\left(\lambda s.r\left(s\,q\right)\right)\right) \qquad \mathsf{dne}_{\forall x\,A}{}^{*c} = \lambda px.\mathsf{dne}_{A}{}^{*c}\left(\lambda q.p\left(\lambda r.q\left(r\,x\right)\right)\right)$$

$$\mathsf{dne}_{A\wedge B}{}^{*c} = \lambda p.\left\langle \mathsf{dne}_{A}{}^{*c}\left(\lambda q.p\left(\lambda r.q\left(\pi_1\,r\right)\right)\right), \mathsf{dne}_{A}{}^{*c}\left(\lambda q.p\left(\lambda r.q\left(\pi_2\,r\right)\right)\right)\right\rangle \quad ◀$$

We can now explain by reformulating the ideas of [11] why we encoded $\exists x\,A$ in $\mathfrak{C}$ instead of interpreting it as in $\mathfrak{M}$. Suppose there is a binary predicate $P\left(t,u\right)$ and some $\phi,\psi_1,\psi_2 \in \langle\!|\iota|\!\rangle$ such that $\left(\phi,\psi_1\right) \notin \{\!|P|\!\}$ and $\left(\phi,\psi_2\right) \in \{\!|P|\!\}$ (for example, P may be equality and $\psi_1 \neq \phi = \psi_2$). Consider the proof:

$$r : P\left(x,y_2\right) \vdash_c \mathsf{dne}\left(\lambda p.p\,\mathsf{ex}\left[y_1,\mathsf{dne}\left(\lambda q.p\,\mathsf{ex}\left[y_2,r\right]\right)\right]\right) : \exists y\,P\left(x,y\right)$$

If $\left(\mathsf{ex}\left[\_,\_\right]\right)^{*c}$ was a pair like in $\mathfrak{M}$, then this proof would be translated to a term of $\lambda\mu$ which is equal in $\mathcal{C}$ to $\mu\alpha.\left[\alpha\right]\left\langle y_1,\mu\beta.\left[\alpha\right]\left\langle y_2,r\right\rangle\right\rangle$. Adequacy would imply that for any $\xi \in \left|P\left(\phi,\psi_2\right)\right|_c$:

$$\zeta = \mu\alpha.\left[\alpha\right]\left\langle\psi_1,\mu\beta.\left[\alpha\right]\left\langle\psi_2,\xi\right\rangle\right\rangle \in \left|\exists y\,P\left(\phi,y\right)\right|_c$$

so $\pi_2\,\zeta \in \left|P\left(\phi,\pi_1\,\zeta\right)\right|_c$ (if $\left|\exists\,y\,\_\right|_c$ was as in $\mathfrak{M}$). But since $\pi_1\,\zeta = \psi_1$ and $\pi_2\,\zeta = \xi$, this would mean that $\xi \in \left|P\left(\phi,\psi_1\right)\right|_c$. The other inclusion being easy, we would get $\left|P\left(\phi,\psi_1\right)\right|_c = \left|P\left(\phi,\psi_2\right)\right|_c$, that is, $\left|P\left(\phi,\psi\right)\right|_c$ would not depend on whether $\left(\phi,\psi\right) \in \{\!|P|\!\}$ or not and the model would be degenerated.

2.4 Extraction for minimal logic

We fix now a theory $\mathcal{A}x$, that is, a set of closed formulas, or axioms. We also suppose that for each $A \in \mathcal{A}x$ we have some realizer $\zeta_A \in \left|A\right|_m$. Extraction is an immediate consequence of adequacy:

▶ **Theorem 2** (Extraction). *Let $\vec{p_A} : \vec{A} \vdash_m M : \forall x\,\exists y\,B$ be a proof, where $\vec{A} \subseteq \mathcal{A}x$ and $FV(B) \subseteq \{x;y\}$. From that proof we can extract some $\phi \in \mathcal{C}\left(1,\iota \to \iota \times B^{*m}\right)$ such that for any $\psi \in \langle\!|\iota|\!\rangle$, $\pi_1\left(\phi\,\psi\right) \in \langle\!|\iota|\!\rangle$ and:*

$$\pi_2\left(\phi\,\psi\right) \in \left|B\left\{\psi/x,\pi_1\left(\phi\,\psi\right)/y\right\}\right|_m$$

Proof. Immediate from adequacy, with $\phi = M^{*m}\left\{\vec{\zeta_A}/\vec{p_A}\right\}$. ◀

Let's look at the particular example of $\mathcal{A}x$ being the set of axioms of arithmetic. Suppose we have realizers of these axioms in $\mathcal{C}$ (which usually means that $\mathcal{C}$ is a model of Gödel's system T) and $\langle\!|\iota|\!\rangle$ is isomorphic to $\mathbb{N}$ with the constants interpreted accordingly. The extraction result tells us that from a proof of $\forall x\,\exists y\left(t = \mathsf{0}\right)$ in $\mathcal{A}x$ we can extract an element:

$$\phi' = \lambda x.\pi_1\left(\phi\,x\right) \in \mathcal{C}\left(1,\iota \to \iota\right)$$

such that for any $n \in \mathbb{N}$, $\left|t\left\{n/x,\phi'\,n/y\right\} = \mathsf{0}\right|_m \neq \emptyset$, and so $\left(n,\phi'\,n\right) \in \{\!|{=}|\!\}$ by definition of the realizability interpretation in $\mathfrak{M}$. If moreover $\{\!|{=}|\!\}$ is equality on $\langle\!|\iota|\!\rangle \simeq \mathbb{N}$ then it tells us that $t^*\left\{n/x,\phi'\,n/y\right\} = 0$ in $\mathbb{N}$.

$$p^R \triangleq p \qquad (\lambda p.M)^R \triangleq \lambda q.\left(\lambda p.M^R\right)(\pi_1\, q)\,(\pi_2\, q) \qquad (M\;N)^R \triangleq \lambda p.M^R\left\langle N^R, p\right\rangle$$

$$\langle M_1, M_2\rangle^R \triangleq \lambda p.\mathtt{case}\; p\left[\mathtt{in}_1\, q \mapsto M_1^R\, q, \mathtt{in}_2\, q \mapsto M_2^R\, q\right] \qquad (\pi_i\; M)^R \triangleq \lambda p.M^R\,(\mathtt{in}_i\, p)$$

$$(\Lambda x.M)^R \triangleq \lambda p.\mathtt{dest}\; p\;\mathtt{as}\;\mathtt{ex}\,[x, q]\;\;\mathtt{in}\; M^R\, q \qquad (M\;\;\lfloor t\rfloor)^R \triangleq \lambda p.M^R\mathtt{ex}\,[t, p]$$

$$(\mathtt{in}_i\; M)^R \triangleq \lambda p.\pi_i\, p\, M^R \qquad (\mathtt{ex}\,[t, M])^R \triangleq \lambda p.p\;\mathtt{ex}\left[t, M^R\right]$$

$$(\mathtt{case}\; M\,[\mathtt{in}_1\, p \mapsto N_1, \mathtt{in}_2\, p \mapsto N_2])^R \triangleq \lambda q.M^R\left\langle \lambda p.N_1^R\, q, \lambda p.N_2^R\, q\right\rangle$$

$$(\mathtt{dest}\; M\;\mathtt{as}\;\mathtt{ex}\,[x, p]\;\;\mathtt{in}\; N)^R \triangleq \lambda q.M^R\left(\lambda r.\mathtt{dest}\; r\;\mathtt{as}\;\mathtt{ex}\,[x, p]\;\;\mathtt{in}\; N^R\, q\right)$$

$$\mathsf{dne}^R \triangleq \lambda p.\pi_1\, p\,\langle \lambda q.\pi_1\, q\,(\pi_2\, p)\,, \lambda r.r\rangle$$

Figure 5 Lafont-Reus-Streicher translation of proofs from $\mathfrak{C}$ to $\mathfrak{M}$.

2.5 Negative translation and Friedman's trick

In this section we explain the indirect interpretation of classical theories via negative translation and Friedman's trick. Negative translation has been defined by Gödel to study the relationship between intuitionistic and classical provability. A formula A is translated to $A^{\neg}$ by inductively replacing every positive formula by a (classically) equivalent negative one. Negative translation turns classical provability into intuitionistic provability, in the sense that if $A_1, \ldots, A_n \vdash_{c} B$, then $A_1^{\neg}, \ldots, A_n^{\neg} \vdash_{m} B^{\neg}$.

Adapting Friedman's original translation [8], we can parameterize negative translation by an arbitrary formula R playing the role of $\bot$. R will be instantiated later on with a carefully chosen formula. This is known as Friedman's trick and can be used to prove conservativity of classical arithmetic over intuitionistic arithmetic for Π_2^0 formulas. Here we use a slightly different version: Lafont-Reus-Streicher (LRS) translation [17]. This version makes explicit the connection with the direct interpretation that we will present in Section 2.6.

For each formula A we define an intermediate translation $A^{\overline{R}}$ in order to then define $A^R \equiv A^{\overline{R}} \Rightarrow R$. $A^{\overline{R}}$ and A^R are parameterized by the arbitrary formula R:

$$\left(P\left(\vec{t}\right)\right)^{\overline{R}} \triangleq P\left(\vec{t}\right) \Rightarrow R \qquad \bot^{\overline{R}} \triangleq \bot \Rightarrow \bot \qquad (\forall x\; A)^{\overline{R}} \triangleq \exists x A^{\overline{R}} \qquad (\exists x\; A)^{\overline{R}} \triangleq \exists x A^R \Rightarrow R$$

$$(A \Rightarrow B)^{\overline{R}} \triangleq A^R \wedge B^{\overline{R}} \qquad (A \wedge B)^{\overline{R}} \triangleq A^{\overline{R}} \vee B^{\overline{R}} \qquad (A \vee B)^{\overline{R}} \triangleq \left(A^R \Rightarrow R\right) \wedge \left(B^R \Rightarrow R\right)$$

This translation turns provability in classical logic into provability in minimal logic:

▶ **Lemma 3.** *If $\Gamma \vdash_{c} A$ then $\Gamma^R \vdash_{m} A^R$ (modulo α-conversion to avoid capture of the free variables of R).*

Proof. The translation $M \mapsto M^R$ on proof terms is given in Figure 5 ◀

We now show how we can transpose the extraction Theorem 2 into an extraction theorem for a classical theory through LRS translation. We fix $\mathcal{C}$ to be a cartesian closed category with coproducts as in Section 2.3, with a chosen object ι, a set $\langle\!|\iota|\!\rangle \subseteq \mathcal{C}(1, \iota)$, interpretations of the constants f and interpretations of the predicates $\{\!|P|\!\} \subseteq \langle\!|\iota|\!\rangle^n$. We also fix a theory $\mathcal{A}x$, and we suppose that for each $A \in \mathcal{A}x$ we have some realizer $\zeta_A \in |A|_m$. Extraction from classical proofs of Π_0^2 formulas is obtained through Friedman's trick combined with the extraction Theorem 2:

▶ **Theorem 4** (Extraction). *Suppose that for every $A \in \mathcal{A}x$ we have $\mathcal{A}x \vdash_{m} A^R$. From a proof of $\mathcal{A}x \vdash_{c} \forall x \, \exists y \, P\left(\vec{t}\right)$ where $FV\left(\vec{t}\right) \subseteq \{x; y\}$, we can extract a morphism $\phi \in \mathcal{C}\left(1, \iota \to \iota\right)$ such that for any $\psi \in \langle\!|\iota|\!\rangle$, $\phi\,\psi \in \langle\!|\iota|\!\rangle$ and:*

$$\vec{t}^{*}\left\{\psi/x, \phi\,\psi/y\right\} \in \{\!|P|\!\}$$

Proof. Elimination of the $\forall$ quantification gives $\mathcal{A}x \vdash_{c} \exists y \, P\left(\vec{t}\right)$, and LRS translation combined with the proofs $\mathcal{A}x \vdash_{m} A^R$ gives some proof term:

$$p_{\vec{A}} : \vec{A} \vdash_{m} M : \left(\exists y \, P\left(\vec{t}\right)\right)^R \equiv \left(\exists y \left(\left(P\left(\vec{t}\right) \Rightarrow R\right) \Rightarrow R\right) \Rightarrow R\right) \Rightarrow R$$

for some $\vec{A} \subseteq \mathcal{A}x$. We apply now Friedman's trick: take $R \equiv \exists \, y \, P\left(\vec{t}\right)$ (its only free variable is x so there is no capture of variables). Then we have:

$$p_{\vec{A}} : \vec{A} \vdash_{m} \Lambda x . M \left(\lambda p . \mathtt{dest}\, p \,\mathtt{as}\, \mathtt{ex}\left[y, q\right] \; \mathtt{in}\, q \left(\lambda r . \mathtt{ex}\left[y, r\right]\right)\right) : \forall x \, \exists \, y \, P\left(\vec{t}\right)$$

to which we apply Theorem 2 and get some ϕ_0 such that for any $\psi \in \langle\!|\iota|\!\rangle$:

$$\pi_2\left(\phi_0\,\psi\right) \in \left|P\left(\vec{t}\{\psi/x, \pi_1\left(\phi_0\,\psi\right)/y\}\right)\right|_m$$

so $\left|P\left(\vec{t}\{\psi/x, \pi_1\left(\phi_0\,\psi\right)/y\}\right)\right|_m \neq \emptyset$, and $\vec{t}^{*}\left\{\psi/x, \phi\,\psi/y\right\} \in \{\!|P|\!\}$ with $\phi = \lambda x . \pi_1\left(\phi_0\,x\right)$. ◀

We discuss now about the assumption that for every $A \in \mathcal{A}x$ we have $\mathcal{A}x \vdash_{m} A^R$.

In the case of arithmetic, since equality is decidable in minimal logic, all the axioms $A \in \mathcal{A}x$ but induction are such that $\mathcal{A}x \vdash_{m} A^R$. For induction it is even simpler since its translation is itself an instance of induction. Therefore, the extraction in minimal arithmetic presented in Section 2.4 can be turned into extraction for classical arithmetic.

This assumption however doesn't hold for every theory. For example, consider the axiom of dependent choice DC (that we can formulate in a multi-sorted version of first-order logic). DC fails to prove DC^R in minimal logic. However, DC^R is a consequence of $DNS + DC$ in minimal logic, where DNS is the double-negation shift:

$$\forall x \left(\left(A \Rightarrow R\right) \Rightarrow R\right) \Rightarrow \left(\forall x \, A \Rightarrow R\right) \Rightarrow R$$

where the sort of x is that of natural numbers. In a single-sorted setting, one can even show that DNS proves $A \Rightarrow A^R$ for any formula A in minimal logic. Historically, this technique has been used to give computational content to the axiom of choice in a classical setting, interpreting DNS intuitionistically with Spector's operator of bar recursion [4, 6].

2.6 Direct interpretation

Since Griffin's discovery [9], we can directly interpret proofs of classical logic in functional programming languages with control operators, as was done in Section 2.2. In order to interpret $\lambda\mu$ (in which classical proofs are mapped), we fix a category of continuations $\mathcal{C} = \diamond^{\mathcal{D}}$ (see [12, 25]). This means that $\mathcal{D}$ is a distributive category, $\diamond$ is an object of $\mathcal{D}$ such that all exponents $\diamond^X$ exist in $\mathcal{D}$, and $\diamond^{\mathcal{D}}$ is the full subcategory of $\mathcal{D}$ consisting of the objects $\diamond^X$. As suggested by the notation, $\diamond$ is the object interpreting the type $\diamond$ of $\lambda\mu$. In order to perform extraction we suppose $\iota = \diamond$ in $\mathcal{C}$. We fix a theory $\mathcal{A}x$ and we suppose that for each $A \in \mathcal{A}x$ we have some realizer $\zeta_A \in \left|A\right|_c$. Extraction requires a clever choice of the parameter $\left|\bot\right|_c$ of the realizability interpretation:

$$\begin{tikzcd}
\mathfrak{C} \arrow[r, "\_^{*c}"] \arrow[dd, "\_^{R}"'] & \lambda\mu \arrow[dr] & & \\
 & & \mathcal{C} = \diamond^{\mathcal{D}} & R^{*m} \simeq \diamond \\
\mathfrak{M} \arrow[r, "\_^{*m}"'] & \lambda^{+} \arrow[ur] & &
\end{tikzcd}$$

Figure 6 Correspondence of direct and indirect realizability interpretations.

▶ **Theorem 5** (Extraction). *From a proof of* $\mathcal{A}x \vdash_{\mathfrak{C}} \forall x\, \exists y\, P\left(\vec{t}\right)$ *where* $FV\left(\vec{t}\right) \subseteq \{x; y\}$, *we can extract a morphism* $\phi \in \mathcal{C}\left(1, \iota \to \iota\right)$ *such that for any* $\psi \in \langle\!|\iota|\!\rangle$, $\phi\, \psi \in \langle\!|\iota|\!\rangle$ *and:*

$$\vec{t}^{*}\, \{\psi/x, \phi\, \psi/y\} \in \{\!|P|\!\}$$

Proof. Write $\vec{p_A} : \vec{A} \vdash_{\mathfrak{C}} M : \forall x\, \exists y\, P\left(\vec{t}\right)$ where $\vec{A} \subseteq \mathcal{A}x$. By adequacy we have for any $\psi \in \langle\!|\iota|\!\rangle$:

$$M^{*c}\left\{\vec{\zeta_A}/\vec{p_A}\right\}\psi \in \left|\exists y\, P\left(\vec{t}\{\psi/x\}\right)\right|_c = \left|\neg\, \forall y\, \neg\, P\left(\vec{t}\{\psi/x\}\right)\right|_c$$

Remember now that $|\bot|_c \subseteq \mathcal{C}\left(1, \diamond\right) = \mathcal{C}\left(1, \iota\right)$ is a parameter that we can choose freely. Take now:

$$|\bot|_c = \left\{\zeta \in \langle\!|\iota|\!\rangle \,\middle|\, \vec{t}^{*}\, \{\psi/x, \zeta/y\} \in \{\!|P|\!\}\right\}$$

By a simple disjunction of cases we prove that $\lambda yp.p\, y \in \left|\forall y\, \neg\, P\left(\vec{t}\{\psi/x\}\right)\right|_c$ for this choice of $|\bot|_c$. Note that this term is well-typed precisely because $\iota = \diamond$. The following morphism then has the required property:

$$\phi = \lambda x.M^{*c}\left\{\vec{\zeta_A}/\vec{p_A}\right\} x\, (\lambda yp.p\, y)\ .$$ ◀

2.7 Correspondence of the two methods

The indirect and direct methods presented in the previous two sections share some similarities, which we shall now make explicit. More precisely, we will prove that the diagram of Figure 6 commutes.

Fix a category of continuations $\mathcal{C} = \diamond^{\mathcal{D}}$. First, observe that the LRS translation of Section 2.5 is such that for any formula A, A^{R} belongs to a restricted syntax of formulas where implications are all of the form $B \Rightarrow R$. Moreover, A^{R} doesn't contain $\forall$, therefore, the simple type $\left(A^{R}\right)^{*m}$ is itself in a restricted syntax where arrow types are all of the form $T \to R^{*}$. This means that $\mathcal{D}$ has enough structure to interpret the LRS translations of $\mathfrak{C}$-proofs if we suppose $R^{*m} \simeq \diamond$ (remember that $\mathcal{D}$ has all exponentials $\diamond^{X}$). Finally, since $\left(A^{R}\right)^{*m} = \left(\overline{A^{R}}\right)^{*m} \to R^{*m}$, LRS translations of $\mathfrak{C}$-proofs can be interpreted in the full subcategory $\mathcal{C} = \diamond^{\mathcal{D}}$. We claim now that this interpretation is the same as the direct interpretation of Section 2.6.

The following lemma, proved by induction on formulas and proofs, connects the two interpretations in $\mathcal{C}$ at the level of formulas, proof terms and realizers:

▶ **Lemma 6.** *If* $R^{*m} \simeq \diamond$, *then:*

- *for any formula* A, $\left(A^{R}\right)^{*m} \simeq A^{*c}$ *in* $\mathcal{C}$
- *for any* $\mathfrak{C}$*-proof term* M, $\left(M^{R}\right)^{*m}$ *and* M^{*c} *are equal in* $\mathcal{C}$, *up to the previous isomorphism*

- *if $|R|_m$ and $|\bot|_c$ are equal up to the isomorphism $R^{*m} \simeq \diamond$, then for any formula A, $\left|A^R\right|_m$ is equal to $|A|_c$, up to the isomorphism $\left(A^R\right)^{*m} \simeq A^{*c}$*

To conclude this section, we show that in the extraction Theorems 4 and 5, we indeed have $R^{*m} \simeq \diamond$ and $|R|_m$ equal to $|\bot|_c$ up to this isomorphism. On one hand, in the extraction Theorem 4, we fixed $R \equiv \exists y\, P\left(\vec{t}\right)$, and therefore:

$$R^{*m} = \left(\exists y\, P\left(\vec{t}\right)\right)^{*m} = \iota \times 1 \simeq \iota$$

And on the other hand, in the extraction Theorem 5 we supposed that $\iota = \diamond$. Therefore we have indeed $R^{*m} \simeq \diamond$. Also, in Theorem 4 we have for any $\psi \in \llparenthesis \iota \rrparenthesis$:

$$|R\{\psi/x\}|_m = \left|\exists y\, P\left(\vec{t}\{\psi/x\}\right)\right|_m = \left\{\phi \,\middle|\, \pi_1\,\phi \in \llparenthesis \iota \rrparenthesis \text{ and } \pi_2\,\phi \in \left|P\left(\vec{t}\{\psi/x, \pi_1\,\phi/y\}\right)\right|_m\right\}$$

but since $\pi_2\,\phi \in \mathcal{C}(1,1)$ which is a singleton, $|R\{\psi/x\}|_m$ is isomorphic to:

$$\left\{\zeta \in \llparenthesis \iota \rrparenthesis \,\middle|\, \left|P\left(\vec{t}\{\psi/x, \zeta/y\}\right)\right|_m \neq \emptyset\right\} = \left\{\zeta \in \llparenthesis \iota \rrparenthesis \,\middle|\, \vec{t}^{*}\{\psi/x, \zeta/y\} \in \{\!|P|\!\}\right\}$$

but this last set is exactly the one chosen for $|\bot|_c$ in Theorem 5. Finally, the requirement of Theorem 4 that for every $A \in \mathcal{A}x$ we have $\mathcal{A}x \Vdash_m A^R$ provides through adequacy a method to turn a set of realizers $\{\zeta_B \in |B|_m \,|\, B \in \mathcal{A}x\}$ into a realizer $\xi_A \in |A|_c \simeq \left|A^R\right|_m$.

Before presenting another method for direct extraction, we discuss the reason for the correspondence between the two methods. This correspondence comes from the fact that we chose $\diamond = \iota$ in the direct interpretation so we could take elements of ι as realizers of $\bot$. This choice was motivated by the fact that taking $|\bot|_c = \emptyset$ gives a degenerated model. However, this is an unnatural interpretation, since the object $\diamond$ should represent an empty type. We will see that even though this natural interpretation is not possible in Scott domains, it is possible in bistable bicpos and game semantics.

3 Another direct interpretation

In this section we present another direct realizability interpretation of $\mathfrak{C}$ in a category of continuations $\mathcal{C} = \diamond^{\mathcal{D}}$. The interpretation of $\mathfrak{C}$-proofs as terms of $\lambda\mu$ is almost identical as in Section 2.2, the only difference being that a proof term $\vec{p} : \vec{A} \vdash_c M : B$ with $\mathrm{FV}\left(\vec{A}, B\right) = \{\vec{x}\}$ is now mapped to a term $M^{*c} : B^{*c}$ of $\lambda\mu$ with free λ-variables among $\vec{p} : \vec{A}^{*c}$, $\vec{x} : \vec{\iota}$, and a special free μ-variable $\kappa : \iota$, which doesn't appear in M^{*c} but may appear in an arbitrary realizer. This free μ-variable will only be used for extraction, as a channel to transmit the extracted value. In categories of continuations, a term $M : T$ of $\lambda\mu$ with λ-context Γ and μ-context Δ is interpreted as a morphism in $\mathcal{C}(\Gamma, T \parr \Delta)$ where $\parr$ is the pretensor defined by $\diamond^X \parr \diamond^Y \triangleq \diamond^{X \times Y}$, see [25]. Therefore, we adapt the realizability semantics, with the realizability value of a formula A being now $|A|_c \subseteq \mathcal{C}(1, A^* \parr \iota)$. The parameter $|\bot|_c$ is now a subset of $\mathcal{C}(1, \diamond \parr \iota)$, which is isomorphic to the homset $\mathcal{C}(1,\iota)$ used in the previous direct interpretation. The difference is that now we can choose $\diamond \neq \iota$ and take $\diamond$ to be a "truly" empty object. The new realizability value for atomic predicates is:

$$\left|P\left(\vec{t}\right)\right|_c \triangleq \begin{cases} |\bot \Rightarrow \bot|_c & \text{if } \vec{t}^{*} \in \{\!|P|\!\} \\ \{\phi \,|\, \text{if } \psi \in \mathcal{C}(1, \diamond \parr \iota) \text{ then } \phi\,\psi \in |\bot|_c\} & \text{otherwise} \end{cases}$$

and the other definitions go through easily, the "$\parr \iota$" part being carried over transparently as a "semantic" free μ-variable. For the definition of $|\forall x\, A|_c$, the morphism $\psi \in \llparenthesis \iota \rrparenthesis \subseteq \mathcal{C}(1,\iota)$ is viewed as a morphism in $\mathcal{C}(1, \iota \parr \iota)$ by adding the semantic μ-variable κ with weakening, that is, post-composing ψ with $w^{l}_{\iota,\iota}$ (with the notations of [25], r emark 2.6). Adequacy still holds and the extraction theorem is very similar to the previous one:

▶ **Theorem 7** (Extraction). *From a proof of $\mathcal{A}x \vdash_{c} \forall x\, \exists y\, P\left(\vec{t}\right)$ where $FV\left(\vec{t}\right) \subseteq \{x; y\}$, we can extract a morphism $\phi \in \mathcal{C}\left(1, \iota \to \iota\right)$ such that for any $\psi \in \langle\!|\iota|\!\rangle$, $\phi\,\psi \in \langle\!|\iota|\!\rangle$ and:*

$$\vec{t}^{*}\left\{\psi/x, \phi\,\psi/y\right\} \in \{\!|P|\!\}$$

Proof. Write $\vec{p_A} : \vec{A} \vdash_{c} M : \forall x\, \exists y\, P\left(\vec{t}\right)$ where $\vec{A} \subseteq \mathcal{A}x$. Adequacy gives for any $\psi \in \langle\!|\iota|\!\rangle$:

$$M^{*c}\left\{\vec{\zeta_A}/\vec{p_A}\right\}\psi \in \left|\exists y\, P\left(\vec{t}\{\psi/x\}\right)\right|_{c} = \left|\neg\forall y\, \neg P\left(\vec{t}\{\psi/x\}\right)\right|_{c}$$

We fix now the parameter $|\bot|_{c} \subseteq \mathcal{C}\left(1, \diamond \parr \iota\right)$:

$$|\bot|_{c} = \left\{\zeta \in \mathcal{C}\left(1, \diamond \parr \iota\right) \,\middle|\, \mu\kappa.\zeta \in \langle\!|\iota|\!\rangle \text{ and } \vec{t}^{*}\left\{\psi/x, \mu\kappa.\zeta/y\right\} \in \{\!|P|\!\}\right\}$$

Here, the morphism $\zeta \in \mathcal{C}\left(1, \diamond \parr \iota\right)$ is viewed as a term of $\lambda\mu$ of type $\diamond$ with the special free μ-variable κ of type ι, and therefore $\mu\kappa.\zeta \in \mathcal{C}\left(1, \iota\right)$. We prove that $\lambda y p.p\left([\kappa]\, y\right) \in \left|\forall y\, \neg P\left(\vec{t}\{\psi/x\}\right)\right|_{c}$ by taking $\varphi \in \langle\!|\iota|\!\rangle$ and $\xi \in \left|P\left(\vec{t}\{\psi/x, \varphi/y\}\right)\right|_{c}$ and showing $\xi\left([\kappa]\,\varphi\right) \in |\bot|_{c}$. We distinguish two cases. If $\vec{t}^{*}\left\{\psi/x, \varphi/y\right\} \in \{\!|P|\!\}$, then $\xi \in |\bot \Rightarrow \bot|_{c}$ and we are left to prove $[\kappa]\,\varphi \in |\bot|_{c}$, which is true since $\mu\kappa.\,[\kappa]\,\varphi = \varphi$ (because the semantic μ-variable κ doesn't appear in φ which comes from weakening), $\varphi \in \langle\!|\iota|\!\rangle$ and $\vec{t}^{*}\left\{\psi/x, \varphi/y\right\} \in \{\!|P|\!\}$. In the other case $\vec{t}^{*}\left\{\psi/x, \varphi/y\right\} \notin \{\!|P|\!\}$, and $\xi\left([\kappa]\,\varphi\right) \in |\bot|_{c}$ by definition of $\left|P\left(\vec{t}\{\psi/x, \varphi/y\}\right)\right|_{c}$, since $[\kappa]\,\varphi \in \mathcal{C}\left(1, \diamond \parr \iota\right)$. Therefore we get:

$$\phi_0 = M^{*}\left\{\vec{\zeta_A}/\vec{p_A}\right\}\psi\left(\lambda y p.p\left([\kappa]\, y\right)\right) \in |\bot|_{c}$$

so $\mu\kappa.\phi_0 \in \langle\!|\iota|\!\rangle$ and $\vec{t}^{*}\left\{\psi/x, \mu\kappa.\phi_0/y\right\} \in \{\!|P|\!\}$ by definition of $|\bot|_{c}$. Finally, the following morphism has the required property:

$$\phi = \lambda x.\mu\kappa.M^{*}\left\{\vec{\zeta_A}/\vec{p_A}\right\} x\left(\lambda y p.p\left([\kappa]\, y\right)\right)\ . \qquad ◀$$

The computational behavior of the extracted term is different from that of the extracted term of Section 2.6, because as soon as the argument $\left(\lambda y p.p\left([\kappa]\, y\right)\right)$ is called inside M^{*c}, y receives a value which is directly transmitted over channel κ and redirected to toplevel. Conversely, in the previous direct interpretation, once the argument $\left(\lambda y p.p\, y\right)$ was called, the value given to y had to go through all the call stack before returning to toplevel. This interpretation corresponds to the meaning of the control features of $\lambda\mu$-calculus.

This new direct interpretation's improvement relies heavily on the fact that we do not require $\diamond = \iota$ anymore, and we can choose $\diamond$ to be a "truly" empty object. We will see in the next sections that the ability to choose $\diamond \neq \iota$ is very dependent on the particular model that we choose.

3.1 Failure in Scott domains

In this section, we explain why in Scott domains, we have no choice but to take $\diamond = \iota$ if we want ι to be in the category of continuations. Recall that a Scott domain is a partial order with a least element, least upper bounds of directed subsets, least upper bounds of non-empty upper-bounded subsets, and which is algebraic (see e.g. [3] for the definitions and basic properties). The standard domain interpretation of a base type ι with set-theoretic interpretation $[\![\iota]\!]$ is $[\![\iota]\!]_{\bot} = \left([\![\iota]\!] \cup \{\bot\}, \leq\right)$ with $x \leq y$ if and only if $x = y$ or $x = \bot$. First, we should ask ourselves what should $\mathcal{D}$ be if $\diamond^{\mathcal{D}}$ is a category of continuations of domains. The category of Scott domains is cartesian closed and has fixpoints: for any morphism $\phi : X \to X$ there is a morphism $\psi : \mathbf{1} \to X$ such that $\psi; \phi = \psi$. It is well-known that a bicartesian closed category with fixpoints has to be trivial (since in that case $\mathbf{1} \simeq \mathbf{0}$). Therefore, since $\mathcal{D}$ should have coproducts, we should relax one of the conditions of Scott domains. The most natural choice is to drop the requirement of existence of a least element and define $\mathcal{D}$ as the category of unpointed Scott domains. In that case, the set-theoretic disjoint union provide $\mathcal{D}$ with a bicartesian closed structure, at the expense of not having fixpoints. We can now state the following failure lemma:

▶ **Lemma 8.** *If $\mathcal{D}$ is a category of unpointed Scott domains, if $\mathcal{C} = \diamond^{\mathcal{D}}$ is a category of Scott domains, if $[\![\iota]\!] \neq \emptyset$ and if $[\![\iota]\!]_\perp$ is an object of $\mathcal{C}$, then $\diamond \simeq [\![\iota]\!]_\perp$.*

Proof. Suppose $[\![\iota]\!]_\perp$ is an object of $\mathcal{C} = \diamond^{\mathcal{D}}$. Then there is some unpointed domain X in $\mathcal{D}$ such that $[\![\iota]\!]_\perp$ is the domain $\diamond^X$ of functions from X to $\diamond$. If X is empty then $\diamond^X$ has only one element, which is impossible since $[\![\iota]\!] \neq \emptyset$. If X has only one element, then $X \simeq \mathbf{1}$ and $\diamond \simeq [\![\iota]\!]_\perp$, which is the conclusion of the lemma. Suppose now that X has at least two elements $a \neq b$. We will derive a contradiction. Since X is non-empty and $\diamond^X = [\![\iota]\!]_\perp$ is pointed, $\diamond$ is also pointed and we write $\perp_\diamond$ for its least element. If $\diamond = \{\perp_\diamond\}$ then $\diamond^X$ has only one element, which is impossible since $[\![\iota]\!] \neq \emptyset$. Therefore there must be some $c \neq \perp_\diamond$ in $\diamond$. Since X is algebraic, we can suppose without loss of generality that a and b are compact (a non-compact element dominates infinitely many compact elements), and since $a \neq b$, we can also suppose without loss of generality that $a \not\geq b$. Define now monotone continuous functions f, g and h from X to $\diamond$ by:

$$f(x) = \perp_\diamond \qquad g(x) = \begin{cases} c \text{ if } x \geq b \\ \perp_\diamond \text{ otherwise} \end{cases} \qquad h(x) = c$$

From the above assumptions we have $f < g < h$ ($a \not\geq b$ ensures $g \neq h$), but $[\![\iota]\!]_\perp$ has no chain of length > 2, so $\diamond^X \not\simeq [\![\iota]\!]_\perp$. ◀

3.2 Bistable bicpos

In this section we prove that the category of bistable bicpos [19] is a category of continuations $\diamond^{\mathcal{D}}$ in which the natural interpretation of ι is in general different from $\diamond$. First we recall the definition of bistable biorders and bistable functions:

▶ **Definition 9** (Bistable Biorder)**.** A bistable biorder is a partial order $(X, \leq)$ together with an equivalence relation $\updownarrow$ on X such that for each $\updownarrow$-equivalence class E, $(E, \leq_{|E})$ is a distributive lattice and the inclusion $E \subseteq X$ preserves meets and joins.

▶ **Definition 10** (Bistable Function)**.** A bistable function from $(X, \leq_X, \updownarrow_X)$ to $(Y, \leq_Y, \updownarrow_Y)$ is a monotone function $f : X \to Y$ such that for any $x, y \in X$, $x \updownarrow_X y$ implies:

$$f(x) \updownarrow_Y f(y) \qquad f(x \wedge y) = f(x) \wedge f(y) \qquad f(x \vee y) = f(x) \vee f(y)$$

The set Y^X of bistable functions from $(X, \leq_X, \updownarrow_X)$ to $(Y, \leq_Y, \updownarrow_Y)$ is itself a bistable biorder:

▶ **Lemma 11.** *If f, g are bistable functions from $(X, \leq_X, \updownarrow_X)$ to $(Y, \leq_Y, \updownarrow_Y)$, define:*

$$f \leq_{Y^X} g \equiv \forall x \in X, f(x) \leq_Y g(x)$$

$$f \updownarrow_{Y^X} g \equiv \begin{cases} \forall x \in X, f(x) \updownarrow_Y g(x) \\ \forall x, y \in X, x \updownarrow_X y \Rightarrow \begin{cases} f(x) \wedge g(y) = f(y) \wedge g(x) \\ f(x) \vee g(y) = f(y) \vee g(x) \end{cases} \end{cases}$$

(note that if $x \updownarrow_X y$ then $f(x) \updownarrow_Y f(y) \updownarrow_Y g(x) \updownarrow_Y g(y)$)

$(Y^X, \leq_{Y^X}, \updownarrow_{Y^X})$ is a bistable biorder.

As shown in [19], the category of bistable biorders is bicartesian closed, the product and coproduct of bistable biorders being defined pointwise. Bistable bicpos are then defined by adding notions of completeness and continuity to bistable biorders:

▶ **Definition 12.** Let $(X, \leq, \updownarrow)$ be a bistable biorder. If E, F are directed subsets of X, write $E \updownarrow F$ if for any $x \in E$ and $y \in F$ there exists $x' \in E$ and $y' \in F$ such that $x \leq x'$, $y \leq y'$ and $x' \updownarrow y'$. $(X, \leq, \updownarrow)$ is a bistable bicpo if $(X, \leq)$ has least upper bounds of directed subsets and for any $E \updownarrow F$, $\bigvee E \updownarrow \bigvee F$ and $\bigvee E \wedge \bigvee F = \bigvee \{x \wedge y \,|\, x \in E, y \in F, x \updownarrow y\}$.

Similarly to the case of Scott domains, we say that a bistable bicpo is pointed if it has both a least and a greatest element which are $\updownarrow$-equivalent. The category of unpointed bistable bicpos and bistable continuous functions is bicartesian closed, while the full subcategory of pointed bistable bicpos is cartesian closed and has fixpoints. The standard interpretation of a base type ι with set-theoretic interpretation $[\![\iota]\!]$ is the bistable bicpo $[\![\iota]\!]_\bot^\top = ([\![\iota]\!] \cup \{\bot; \top\}, \leq, \updownarrow)$ with $x \leq y$ if and only if $x = y$ or $x = \bot$ or $y = \top$, and $x \updownarrow y$ if and only if $x = y$ or $\{x; y\} = \{\bot; \top\}$. If we choose $\mathcal{D}$ to be the category of unpointed bistable bicpos and $\diamond = \emptyset_\bot^\top$, then $\mathcal{C} = \diamond^{\mathcal{D}}$ is a category of continuations of bistable bicpos and all $[\![\iota]\!]_\bot^\top$ are objects of $\mathcal{C}$. Indeed, it was proved in [19] (for $[\![\iota]\!] = \mathbb{N}$, but the extension to arbitrary $[\![\iota]\!]$ is straightforward) that if $[\![\iota]\!]$ denotes the unpointed bistable bicpo $([\![\iota]\!], \leq, \updownarrow)$ with $x \leq y$ if and only if $x = y$ and $x \updownarrow y$ if and only if $x = y$, then $[\![\iota]\!]_\bot^\top$ is isomorphic to the space of bistable continuous functions from $\left(\emptyset_\bot^\top\right)^{[\![\iota]\!]}$ to $\emptyset_\bot^\top$.

One may wonder if the failure in Scott domains was not simply because we interpreted the datatype with values in $[\![\iota]\!]$ as $[\![\iota]\!]_\bot$ rather than $[\![\iota]\!]_\bot^\top$, which is also a Scott domain. This is not the case, since the space of monotone continuous functions from ${\emptyset_\bot^\top}^{[\![\iota]\!]}$ (where $[\![\iota]\!]$ is the unpointed Scott domain with $x \leq y$ iff $x = y$) to $\emptyset_\bot^\top$ is isomorphic to the set $\{\mathcal{E} \subseteq \mathcal{P}_{fin}([\![\iota]\!]) \,|\, E \in \mathcal{E} \wedge E \subseteq F \Rightarrow F \in \mathcal{E}\}$ ordered with inclusion, which is clearly not isomorphic to $[\![\iota]\!]_\bot^\top$. A careful analysis shows that if we restrict ourselves to bistable continuous functions, then the sets $\mathcal{E} \subseteq \mathcal{P}_{fin}([\![\iota]\!])$ above also have to satisfy: $E \in \mathcal{E} \wedge F \in \mathcal{E} \Rightarrow E \cap F \in \mathcal{E}$ and $E \cup F \in \mathcal{E} \Rightarrow E \in \mathcal{E} \vee F \in \mathcal{E}$. The only possibilities are then $\mathcal{E} = \emptyset$, $\mathcal{E} = \mathcal{P}_{fin}([\![\iota]\!])$ and $\mathcal{E} = \{E \in \mathcal{P}_{fin}([\![\iota]\!]) \,|\, v \in E\}$ for $v \in [\![\iota]\!]$, and we indeed get back $[\![\iota]\!]_\bot^\top$.

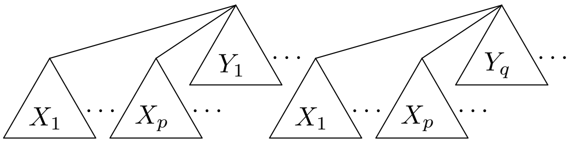
3.3 Game semantics

In this section, we take $\mathcal{C}$ to be the category of unbracketed non-innocent but single-threaded Hyland-Ong games. Hyland-Ong game semantics provide precise models of various programming languages such as PCF [13, 21, 2], also augmented with control operators [18] and higher-order references [1]. In game semantics, plays are interaction traces between a program (player P) and an environment (opponent O). A program is interpreted by a strategy for P which represents the interactions it can have with any environment. We will only define what is necessary for our result, and we refer to e.g. [10] for the full definitions and properties. In the category $\mathcal{C}$, objects are arenas and morphisms are strategies:

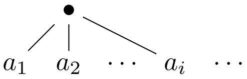
▶ **Definition 13** (Arena)**.** An *arena* is a countably branching, finite depth forest of *moves*. Each move is given a polarity O (for Opponent) or P (for Player or Proponent): a root is of polarity O and a move which is not a root has the inverse polarity than that of his parent. A root of an arena is also called an initial move.

▶ **Definition 14** (Play, Strategy)**.** A play on an arena X is a justified sequence of moves of X with alternating polarities, starting with an O-move. A *strategy* on X is a non-empty even-prefix-closed set of even-length plays on X which is deterministic and single-threaded.

A play on an arena is the trace of an interaction between a program and a context, each one performing an action alternately, and a strategy represents all the interactions that a given program can have with its environment. The definitions of justified sequence, determinism and single-threadedness are standard and can be found for example in [10]. $\mathcal{C}$ is cartesian closed and has countable products, the I-indexed product of $\{X_i \mid i \in I\}$ being the juxtaposition of arenas X_i, and if X and Y are arenas consisting of the trees $X_1 \ldots X_p \ldots$ and $Y_1 \ldots Y_q \ldots$, then the arena Y^X can be represented as follows (roots are at the top):



The standard interpretation of a base type ι in $\mathcal{C}$ is the flat arena $[\![\iota]\!]^\dagger$ associated to the set-theoretic interpretation $[\![\iota]\!]$ of ι. This flat arena is the tree with one root move and a child move for each element of $[\![\iota]\!] = \{a_1; a_2; \ldots; a_i; \ldots\}$:



As in the cases of Scott domains and bistable bicpos, if we want to get a category of continuations $\diamond^{\mathcal{D}}$, we must first find out what $\mathcal{D}$ should be. In game semantics, however, there is no natural notion of unpointed arena, since the strategy consisting of only the empty play is always a least element. We will therefore simply take $\mathcal{D}$ to be the countable coproduct completion $\mathrm{Fam}\,(\mathcal{C})$ of $\mathcal{C}$:

▶ **Definition 15** ($\mathrm{Fam}\,(\mathcal{C})$)**.** The objects of $\mathrm{Fam}\,(\mathcal{C})$ are families of objects of $\mathcal{C}$ indexed by countable sets, and a morphism from $\{X_i \mid i \in I\}$ to $\{Y_j \mid j \in J\}$ is a function $f : I \to J$ together with a family of morphisms of $\mathcal{C}$ from X_i to $Y_{f(i)}$, for $i \in I$.

$\mathrm{Fam}\,(\mathcal{C})$ is a distributive category, the empty product being the singleton family $\{\mathbf{1}\}$, the product of $\{X_i \mid i \in I\}$ and $\{Y_j \mid j \in J\}$ being $\{X_i \times Y_j \mid (i,j) \in I \times J\}$, the empty coproduct being the empty family $\{\}$, and the coproduct of two families being the disjoint union of the two families. $\mathrm{Fam}\,(\mathcal{C})$ is not cartesian closed, but has exponentials of all singleton families: the object of functions from $\{X_i \mid i \in I\}$ to the singleton family $\{Y\}$ is the singleton family $\left\{\Pi_{i\in I} Y^{X_i}\right\}$. This property implies that if $\diamond$ is a singleton family, then $\diamond^{\mathcal{D}} = \diamond^{\mathrm{Fam}(\mathcal{C})}$ is a category of continuations. We now prove that if we choose $\diamond$ carefully, we can completely reveal the continuation structure of $\mathcal{C}$:

▶ **Lemma 16.** *If $\diamond = \left\{\emptyset^\dagger\right\}$, then the category $\diamond^{\mathcal{D}} = \diamond^{\mathrm{Fam}(\mathcal{C})}$ is isomorphic to the category $\mathcal{C}$.*

Proof. Since $\left\{\emptyset^\dagger\right\}$ is a singleton family, the objects of $\diamond^{\mathcal{D}}$ are all singleton families and we have a functor from $\diamond^{\mathcal{D}}$ to $\mathcal{C}$ which is full, faithful and strictly injective on objects and which maps $\{X\}$ to X and $(Id, \phi) \in \diamond^{\mathcal{D}}(\{X\}, \{Y\})$ to $\phi \in \mathcal{C}(X, Y)$. We now show that it is also surjective on objects, so it is an isomorphism of categories. This amounts to show that any arena X can be written as $\Pi_{i\in I}\left(\emptyset^\dagger\right)^{Y_i}$, but this is immediate if one takes I to be the set of roots of X and Y_i to be the forest under root $i \in I$ in X. ◀

Therefore $\mathcal{C}$ is (isomorphic to) a category of continuations $\diamond^{\mathcal{D}}$ where $\diamond \neq \iota$, which was our goal. Note that this result is stronger than the one on bistable bicpos, since $\diamond^{\mathcal{D}}$ is not only a category of continuations of arenas which has the natural interpretation of ι as an object, but $\diamond^{\mathcal{D}}$ is isomorphic to the category $\mathcal{C}$ of arenas.

We now examine the extraction technique presented at the beginning of Section 3 in the particular case of game semantics. In order to have a realizability model, we define $\{\!|\iota|\!\}$ to be the set of all strategies on $\llbracket\iota\rrbracket^{\dagger}$ but $\{\epsilon\}$, so we have $\{\!|\iota|\!\} \simeq \llbracket\iota\rrbracket$. We write $\underline{a}$ for the strategy on $\llbracket\iota\rrbracket^{\dagger}$ corresponding to $a \in \llbracket\iota\rrbracket$ (it answers a to the unique initial move), so $\{\!|\iota|\!\} = \{\underline{a} \,|\, a \in \llbracket\iota\rrbracket\}$. The extraction technique for classical proofs presented at the beginning of Section 3 becomes in this setting:

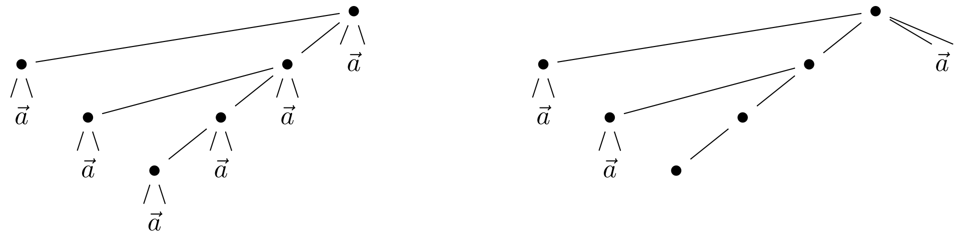
▶ **Theorem 17** (Extraction). *Suppose given a strategy $\zeta_A \in |A|_c$ for each $A \in \mathcal{A}x$. We can extract from a proof of $\mathcal{A}x \vdash_c \forall x \,\exists y \, P\left(\vec{t}\right)$ where $FV\left(\vec{t}\right) \subseteq \{x; y\}$ a strategy ϕ on the arena $\left(\llbracket\iota\rrbracket^{\dagger}\right)^{\llbracket\iota\rrbracket^{\dagger}}$ such that for any $a \in \llbracket\iota\rrbracket$, $\phi\, \underline{a} = \underline{b}$ for some $b \in \llbracket\iota\rrbracket$, and $\vec{t}^{*}\, \{\underline{a}/x, \underline{b}/y\} \in \{\!|P|\!\}$.*

Let us compare now the arena where realizers of the formula $\forall x \,\exists y \, P\left(\vec{t}\right)$ live in the standard and new interpretations. First, we have:

$$\left(\forall x \,\exists y \, P\left(\vec{t}\right)\right)^{*c} = \left(\forall x \,\neg \forall y \,\neg \, P\left(\vec{t}\right)\right)^{*c} = \iota \to (\iota \to (\diamond \to \diamond) \to \diamond) \to \diamond$$

In the standard interpretation, $\diamond = \iota$ is $\llbracket\iota\rrbracket^{\dagger}$, the flat arena for $\llbracket\iota\rrbracket$, so the arena of realizers is $\llbracket\iota\rrbracket^{\dagger} \Rightarrow \left(\llbracket\iota\rrbracket^{\dagger} \Rightarrow \left(\llbracket\iota\rrbracket^{\dagger} \Rightarrow \llbracket\iota\rrbracket^{\dagger}\right) \Rightarrow \llbracket\iota\rrbracket^{\dagger}\right) \Rightarrow \llbracket\iota\rrbracket^{\dagger}$. In the new interpretation however, $\diamond$ is the one-move arena $\emptyset^{\dagger}$, and a " $\parr\, \iota$" is added, so the arena of the realizers of the same formula is $\left(\llbracket\iota\rrbracket^{\dagger} \Rightarrow \left(\llbracket\iota\rrbracket^{\dagger} \Rightarrow (\emptyset^{\dagger} \Rightarrow \emptyset^{\dagger}) \Rightarrow \emptyset^{\dagger}\right) \Rightarrow \emptyset^{\dagger}\right) \parr \llbracket\iota\rrbracket^{\dagger}$. The two arenas are as follows:

standard interpretation: new interpretation:



where $\vec{a}$ represents the sequence of all elements of $\llbracket\iota\rrbracket$. We can observe that the flat arenas for the atomic formulas in the standard interpretation are replaced with one-move arenas in the new interpretation, and a set of moves $\vec{a}$ is added, but only under the root (this corresponds to the " $\parr\, \iota$"). On the computational level, when in the standard interpretation a realizer gives a value for an atomic formula, this value is copied to the various parts of the arena which are in the call stack. Conversely, in the new interpretation the realizer writes the value directly under the root and the computation stops (this corresponds to the interpretation of μ-variables as "channels").

4 Conclusion

We defined a method to extract strategies of game semantics from classical proofs. This method uses peculiarities of the games model allowing the interpretation of the $\bot$ formula as an empty type while still being able to extract computational content, through the use of an external "output channel" on which the extracted value is transmitted, without going through all the call stack. It would be interesting to compare these results with the technique described in [20], which also reduces the amount of recursive calls, but takes place in an untyped, syntactic setting. Working with non-innocent games also has the advantage that we have access to higher-order references which could be used to write efficient realizers.

References

1 Samson Abramsky, Kohei Honda, and Guy McCusker. A Fully Abstract Game Semantics for General References. In *13th Annual IEEE Symposium on Logic in Computer Science*, pages 334–344. IEEE Computer Society, 1998.

2 Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full Abstraction for PCF. *Information and Computation*, 163(2):409–470, 2000.

3 Roberto Amadio and Pierre-Louis Curien. *Domains and Lambda-Calculi*, volume 46 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.

4 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.

5 Ulrich Berger, Wilfried Buchholz, and Helmut Schwichtenberg. Refined program extraction from classical proofs. *Annals of Pure and Applied Logic*, 114(1-3):3–25, 2002.

6 Ulrich Berger and Paulo Oliva. Modified bar recursion and classical dependent choice. In *Logic Colloquium 2001, Proceedings of the Annual European Summer Meeting of the Association for Symbolic Logic*, volume 20 of *Lecture Notes in Logic*, pages 89–107. A K Peters, Ltd., 2005.

7 Valentin Blot and Colin Riba. On Bar Recursion and Choice in a Classical Setting. In *11th Asian Symposium on Programming Languages and Systems*, volume 8301 of *Lecture Notes in Computer Science*, pages 349–364. Springer, 2013.

8 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.

9 Timothy Griffin. A Formulae-as-Types Notion of Control. In *17th Symposium on Principles of Programming Languages*, pages 47–58. ACM Press, 1990.

10 Russ Harmer. *Games and full abstraction for non-deterministic languages.* PhD thesis, Imperial College London (University of London), 1999.

11 Hugo Herbelin. On the Degeneracy of Sigma-Types in Presence of Computational Classical Logic. In *7th International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Mathematics, pages 209–220. Springer, 2005.

12 Martin Hofmann and Thomas Streicher. Completeness of Continuation Models for $\lambda\mu$-Calculus. *Information and Computation*, 179(2):332–355, 2002.

13 Martin Hyland and Luke Ong. On Full Abstraction for PCF: I, II, and III. *Information and Computation*, 163(2):285–408, 2000.

14 Stephen Cole Kleene. On the Interpretation of Intuitionistic Number Theory. *Journal of Symbolic Logic*, 10(4):109–124, 1945.

15 Jean-Louis Krivine. Dependent choice, 'quote' and the clock. *Theoretical Computer Science*, 308(1–3):259–276, 2003.

16 Jean-Louis Krivine. Realizability in classical logic. *Panoramas et synthèses*, 27:197–229, 2009.

17 Yves Lafont, Bernhard Reus, and Thomas Streicher. Continuations Semantics or Expressing Implication by Negation. Technical Report 93-21, Ludwig-Maximilians-Universität, München, 1993.

18 James Laird. Full Abstraction for Functional Languages with Control. In *12th Annual IEEE Symposium on Logic in Computer Science*, pages 58–67. IEEE Computer Society, 1997.

19 James Laird. Bistable Biorders: A Sequential Domain Theory. *Logical Methods in Computer Science*, 3(2), 2007.

20 Alexandre Miquel. Existential witness extraction in classical realizability and via a negative translation. *Logical Methods in Computer Science*, 7(2), 2011.

21 Hanno Nickau. Hereditarily Sequential Functionals. In *Third International Symposium on Logical Foundations of Computer Science*, Lecture Notes in Computer Science, pages 253–264. Springer, 1994.

22 Paulo Oliva and Thomas Streicher. On Krivine's Realizability Interpretation of Classical Second-Order Arithmetic. *Fundamenta Informaticae*, 84(2):207–220, 2008.

23 Michel Parigot. $\lambda\mu$-Calculus: An Algorithmic Interpretation of Classical Natural Deduction. In *3rd International Conference on Logic Programming and Automated Reasoning*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 1992.

24 Christophe Raffalli. Getting results from programs extracted from classical proofs. *Theoretical Computer Science*, 323(1-3):49–70, 2004.

25 Peter Selinger. Control categories and duality: on the categorical semantics of the $\lambda\mu$ calculus. *Mathematical Structures in Computer Science*, 11(2):207–260, 2001.

Proving Correctness of Logically Decorated Graph Rewriting Systems*

Jon Haël Brenas[1], Rachid Echahed[2], and Martin Strecker[3]

1 **CNRS and Université Grenoble Alpes, Saint Martin d'Hères, France**
`Jon-Hael.Brenas@imag.fr`

2 **CNRS and Université Grenoble Alpes, Saint Martin d'Hères, France**
`Jon-Hael.Brenas@imag.fr`

3 **IRIT – Université de Toulouse, Toulouse, France**
`martin.strecker@iri.fr`

Abstract

We first introduce the notion of logically decorated rewriting systems where the left-hand sides are endowed with logical formulas which help to express positive as well as negative application conditions, in addition to classical pattern-matching. These systems are defined using graph structures and an extension of combinatory propositional dynamic logic, $\mathcal{CPDL}$, with restricted universal programs, called $\mathcal{C2PDL}$. In a second step, we tackle the problem of proving the correctness of logically decorated graph rewriting systems by using a Hoare-like calculus. We introduce a notion of specification defined as a tuple (Pre, Post, R, S) with Pre and Post being formulas of $\mathcal{C2PDL}$, R a rewriting system and S a rewriting strategy. We provide a sound calculus which infers proof obligations of the considered specifications and establish the decidability of the verification problem of the (partial) correctness of the considered specifications.

1998 ACM Subject Classification D.2.4 Software/Program Verification, F.4.1 Mathematical Logic, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases Graph Rewriting, Hoare Logic,Combinatory PDL, Rewrite Strategies, Program Verification

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.14

1 Introduction

Rewriting techniques and particularly term rewriting systems have been very successful in different areas such as theorem proving or declarative programming languages. Term rewriting systems have a wide range of interesting results such as confluence analysis, termination orderings and even very powerful proof techniques based, in particular, on equational reasoning and structural induction. However, the structure of terms (trees) is not well suited to specify easily problem handling graph structures, unless one uses cumbersome encodings.

In this paper we will focus on a class of rewriting systems that manipulate graphs. Graphs are data structures that have become ubiquitous. In addition to discrete mathematics and computer science, they are also used to model data in various fields such as biology, geography, physics etc. The transformation of graphs is nowadays a domain of research in its own. One may distinguish two main streams, in the literature, for graph transformations : (i) the algorithmic approaches, which describe explicitly the algorithms involved in the application

* This work has been funded by the project CLIMT(ANR/(ANR-11-BS02-016).

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 14; pp. 14:1–14:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

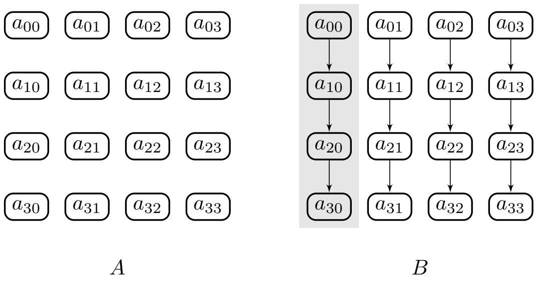


Figure 1 A – A Sudoku grid. B – An illustration of the correct definition of columns. We omit the labels of the edges.

of a rule to a graph, and (ii) the algebraic approaches which define abstractly a graph transformation step using basic constructs borrowed from category theory.

In this paper, we follow the algorithmic approach as proposed in [7] and consider rewrite rules of the form $lhs \rightarrow \alpha$ where lhs is a graph and α is a sequence of elementary actions that perform the desired transformations on the subgraph matched by lhs. We define the notion of *logically decorated graph rewriting systems* (LDRS) where the left-hand sides of rules are graphs attributed by formulas in a dynamic logic, called $\mathcal{C}2\mathcal{PDL}$[5], and whose models are also graphs. Such formulas, within the left-hand sides, could be seen as additional conditions to be fulfilled when matching a subgraph.

After the introduction of LDRS systems, we tackle the problem of their verification with the objective of building a decidable procedure. For that we define a Hoare-like calculus the aim of which is to prove that a transformation is correct, i.e., given a set of rewriting rules, a strategy stating how to apply them, a pre-condition indicating what are the properties to be satisfied before the application of the transformations and a post-condition stating which property is to be verified after the transformations, whether for any graph G satisfying the pre-condition every graph obtained by transforming G will satisfy the post-condition. To do so, we define a calculus that generates weakest-pre-conditions and verification conditions for each intermediate step of the strategy. This infers a weakest pre-condition and a verification condition for the whole transformations. That weakest pre-condition is then compared to the given pre-condition. We show that the proposed Hoare-like calculus is sound and that the considered correctness problem is decidable.

▶ **Example 1.** To clarify what is our aim and how our system works, we will be using a running example: we propose to study a simple program dealing with Sudoku grids. For reason of clarity and conciseness, we will study 4x4 grids instead of the normal 9x9. Nonetheless, the example can be easily extended to the normal Sudoku. An example of such a grid is shown in Figure 1.A. The goal is to fill each blank cell with a number between 1 and 4 such that the same number doesn't appear twice on a line, a column or a square. The goal of this example is not to show that graph transformations are efficient for solving Sudokus but just to provide a rather simple and common example in order to illustrate how to carry out the correctness proof of a program defined as a graph rewrite system.

The paper is organized as follows. In the following section, the dynamic logic $\mathcal{C}2\mathcal{PDL}$ used to express pre- and post-conditions is presented briefly. Then, the class of logically decorated rewriting systems is defined in Section 3 together with a notion of rewrite strategies. In Section 4, we start by setting the verification problem we consider and show how the proof obligations are generated. We also prove that the presented verification process is sound and decidable. An overview of related work is given in Section 5. Section 6 concludes the paper.

2 The dynamic logic $\mathcal{C}2\mathcal{PDL}$

In this section, we introduce $\mathcal{C}2\mathcal{PDL}$ [5], a logic that we use to specify assertions. It is a mix of Converse Propositional Dynamic Logic [8] and Combinatory Propositional Dynamic Logic [15], both commonly known as $\mathcal{CPDL}$. $\mathcal{C}2\mathcal{PDL}$ contains elements of Propositional Dynamic Logic, that allows one to define complex role constructors, and Hybrid Logic, which allows one to use the power of nominals. $\mathcal{C}2\mathcal{PDL}$ further extends the $\mathcal{CPDL}$s in two ways: it splits the universe into elements that are part of the model and elements that may be created by an action or that have been deleted; it also extends the notion of universal role to "total" roles over subsets of the universe in order to be able to deal with these modifications of the universe.

▶ **Definition 2** (Syntax of $\mathcal{C}2\mathcal{PDL}$). Given three countably infinite and pairwise disjoint alphabets Σ, the set of *names*, Φ_0, the set of *atomic propositions*, Π_0, the set of *atomic programs*, the language of $\mathcal{C}2\mathcal{PDL}$ is composed of *formulas* and *programs*[1]. We partition the set of names Σ into two countably infinite alphabets Σ_1 and Σ_2 such that $\Sigma_1 \cup \Sigma_2 = \Sigma$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$. Formulas ϕ and programs α are defined as:

$$\begin{array}{lll} \phi & := & i \mid \phi_0 \mid \neg\phi \mid \phi \vee \phi \mid \langle\alpha\rangle\phi \\ \alpha & := & \alpha_0 \mid \nu_S \mid \alpha;\alpha \mid \alpha \cup \alpha \mid \alpha^* \mid \alpha^- \mid \phi? \end{array}$$

where $i \in \Sigma$, $\phi_0 \in \Phi_0$, $\alpha_0 \in \Pi_0$ and $S \subseteq \Sigma$.

We denote by Π the set of programs and by Φ the set of formulas. As ususal, $\phi \wedge \psi$ stands for $\neg(\neg\phi \vee \neg\psi)$ and $[\alpha]\phi$ stands for $\neg(\langle\alpha\rangle\neg\phi)$.

For now, the splitting of Σ seems artificial. It is actually grounded in the use we want to make of the logic. Roughly speaking, Σ_1 stands for the names that are used in "the" current model whereas Σ_2 stands for the names that may be used in the future (or have been used in the past but do not participate in the current model).

▶ **Definition 3** (Model). A *model* is a tuple $\mathcal{M} = (M, R, \chi, V)$ where M is a set called the *universe*, $\chi : \Sigma \to M$ is a surjective mapping such that $\chi(\Sigma_1) \cap \chi(\Sigma_2) = \emptyset$, $R : \Pi \to \mathcal{P}(M^2)$ and $V : \Phi \to \mathcal{P}(M)$ are mappings such that:

- For each $\alpha_0 \in \Pi_0$, $R(\alpha_0) \in \mathcal{P}(\chi(\Sigma_1)^2)$
- $R(\nu_S) = \chi(S)^2$ for $S \subseteq \Sigma$
- $R(\alpha \cup \beta) = R(\alpha) \cup R(\beta)$
- $R(A?) = \{(s, s) | s \in V(A)\}$
- $R(\alpha^-) = \{(s, t) | (t, s) \in R(\alpha)\}$
- $R(\alpha^*) = \bigcup_{k<\omega} R(\alpha^k)$ where α^k stands for the sequence $\alpha; \ldots; \alpha$ of length k
- $R(\alpha; \beta) = \{(s, t) | \exists v.((s, v) \in R(\alpha) \wedge (v, t) \in R(\beta))\}$
- For each $i \in \Sigma$, $V(i) = \{\chi(i)\}$
- For each $\phi_0 \in \Phi_0$, $V(\phi_0) \in \mathcal{P}(\chi(\Sigma_1))$
- $V(\neg A) = M \backslash V(A)$
- $V(A \vee B) = V(A) \cup V(B)$
- $V(\langle\alpha\rangle A) = \{s | \exists t \in M.((s, t) \in R(\alpha) \wedge t \in V(A))\}$

In the following, we write $sR_\alpha t$ for $(s, t) \in R(\alpha)$.

$\mathcal{C}2\mathcal{PDL}$ will be used in this paper to label nodes and to express properties of attributed graphs.

[1] This notion of *programs* is borrowed from PDL logic.

▶ **Definition 4** (Attributed Graph). An *attributed graph* G is a tuple $(N,E,\mathcal{C},\mathcal{R},L_N,L_E,s,t)$ where N is the set of *nodes*, E is the set of *edges*, $\mathcal{C}$ is the set of *node labels* or *concepts*, $\mathcal{R}$ is the set of *edge labels* or *roles*, L_N is the *node labeling* (total) function, $L_N : N \to \mathcal{P}(\mathcal{C})$, L_E is the *edge labeling* (total) function, $L_E : E \to \mathcal{R}$, s is the *source function* $s : E \to N$ and t is the *target function* $t : E \to N$.

From now on, we will only consider graphs such that $\mathcal{C} = \Phi$ and $\mathcal{R} = \Pi$. Given a graph $G = (N, E, \mathcal{C}, \mathcal{R}, L_N, L_E, s, t)$ and a formula ϕ, we say that $G \models \phi$ if there exists $n \in N$ such that $n \models \phi$ where the universe M is the set of nodes N. R and V are defined as usual: for $\phi_0 \in \Phi_0$ (resp. $\pi_0 \in \Pi_0$), $V(\phi_0) = \{x \in N|\phi_0 \in L_N(x)\}$ (resp. $R(\pi_0) = \{(x, y) \in N^2|\exists e \in E.s(e) = x \wedge t(e) = y \wedge L_E(e) = \pi_0\}$). R and V are then extended to non-atomic propositions and programs following the same rules defined in the models. As usual, a formula ϕ is satisfiable if there exists a graph G such that $G \models \phi$ and unsatisfiable otherwise and it is valid if for all models G, $G \models \phi$ and invalid otherwise. We denote by S a subset of $\mathcal{C}$ which consists of names such that for each name $s \in S$ there is at most one node $n \in N$ such that $n \models s$. One may remark that all models can be considered as graphs. The converse is false.

Often, we will write $i : C$ instead of $i : \{C\}$ to say that node i is labelled with the formula C. In Figure 2, we give an example of models depicted as graphs.

Attributed graphs where all nodes are named will be called *named graphs*. This notion of graphs will be used in the proof section.

▶ **Definition 5** (Named Graph). A *named graph* G is an attributed graph such that the set of names $S \subseteq C$ satisfies:

(a) $\forall s \in S.\ \exists n \in N.\ s \in L_N(n)$,
(b) $\forall n \in N.\ L_N(n) \cap S \neq \emptyset$,
(c) $\forall n, n' \in N, n \neq n',\ L_N(n) \cap L_N(n') \cap S = \emptyset$.

Notation: From (a), (b) and (c), it is obvious that, as each name labels at least one node, each node is labeled by at least one name and each name labels at most one node, it is possible to define two functions θ and μ such that $\forall s \in S, \theta(s)$ is the node named s and $\forall n \in N, \mu(n)$ is a name of n. This allows to define the surjective mapping of models χ and thus named graphs and models are equivalent structures. We will thus consider from now on that formulae are interpreted over named graphs.

▶ **Example 6.** Going back to the Sudoku example, we will use $\mathcal{C2PDL}$ to state some properties. We are going to use a name for each cell of the grid (a_{ij} with $0 \leq i, j \leq 3$ where i is the row and j the column in which the cell can be found). We will also use three atomic programs R (resp. C and SQ) to state which cells are on the same row (resp. column and square). Finally, we will need eight atomic propositions 1 (resp. 2,3 and 4) and P_1 (resp. P_2,P_3 and P_4) to state that a cell *is known to* contain 1 (resp. 2, 3 or 4) and that a cell *may* contain 1 (resp. 2, 3 or 4). Thus we require that $\{a_{ij}|i, j \in [0, 3]\} \subset \Sigma$, $\{i|i \in [1, 4]\} \cup \{P_i|i \in [1, 4]\} \subset \Phi_0$ and $\{R, C, SQ\} \subset \Pi_0$. As we do not create or delete nodes, we do not make use of the possibility to change the set of definition of the total program. For this example, ν stands for ν_{Σ_1}. We define below a few relevant formulae.

- The atomic program C should describe columns, as shown in Figure 1.B:
  $c_j = \langle\nu\rangle(a_{0j}\ \wedge \langle C\rangle(a_{1j}\ \wedge \langle C\rangle(a_{2j}\ \wedge \langle C\rangle a_{3j})))$ for $j \in [0, 3]$. c_j describes the successive elements of a column.
  $\overline{c}_j = \langle\nu\rangle(a_{0j}\ \wedge [C](a_{1j}\ \wedge [C](a_{2j}\ \wedge [C](a_{3j} \wedge [C]\bot))))$ for $j \in [0, 3]$. $\overline{c}_j$ says that there are no more elements in a column than those specified by c_j. Thus a column is specified by $c_j \wedge \overline{c}_j$.

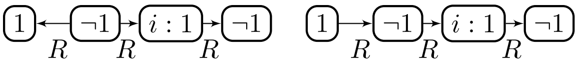


Figure 2 Model and counter-model. All nodes of the left graph satisfy the formula $[\nu](1 \Rightarrow [R^-][R^{-*}]\neg 1)$. This is not the case for the graph given on the right since $i \not\models (1 \Rightarrow [R^-][R^{-*}]\neg 1)$.

- A cell should contain, at most, one value: $u_{I,J} = [\nu](I \Rightarrow \neg J)$ for $I, J \in [1,4]$, $I \neq J$ and there is no doubt about it (e.g., once a cell is assigned the value I, it is no longer a candidate for any future potential assignement P_J): $\overline{u}_{I,J} = [\nu](I \Rightarrow \neg P_J)$ for $I, J \in [1,4]$
- If a cell has a value J, J cannot be the value of any other cell on the same row, column or square $v_{r,J} = [\nu](J \Rightarrow (([r][r^*]\neg J) \wedge ([r^-][r^{-*}]\neg J)))$ for $J \in [1,4]$ and $r \in \{R, C, SQ\}$. Figure 2 shows an example of a model and a counter-model of part of this expression.

▶ **Theorem 7.** *Given a formula ϕ of $\mathcal{C2PDL}$, the satisfiability and the validity of ϕ are decidable.*

More on $\mathcal{C2PDL}$ including the proof of this theorem can be found in [5].

3 Logically Decorated Rewriting Systems LDRS

In this section we introduce the notion of *logically decorated rewriting systems*, LDRS. These are extensions of graph rewriting systems defined in [7] where graphs are attributed with $\mathcal{C2PDL}$ formulas. The left-hand sides of the rules are thus logically decorated graphs whereas the right-hand sides are defined as sequences of elementary actions. These actions constitute a set of elementary transformations used in graph transformation processes. The operational way the right-hand sides are defined in this paper departs from those classically used in algebraic approaches [18] such as simple pushout or double pushout where rules are defined by means of graph morphisms.

▶ **Definition 8** (Elementary Action, Action)**.** An *elementary action*, say a, has one of the following forms:

- a *concept addition* $add_C(i, c)$ (resp. *concept deletion* $del_C(i, c)$) where i is a node and c is a basic concept (a proposition name) in Φ_0. It adds the node i to (resp. removes the node i from) the valuation of the concept c.
- a *role addition* $add_R(i, j, r)$ (resp. *role deletion* $del_R(i, j, r)$) where i and j are nodes and r is an atomic role (edge label) in Π_0. It adds the pair (i, j) to (resp. removes the pair (i, j) from) the valuation of the role r.
- a *node addition* $add_I(i)$ (resp. *node deletion* $del_I(i)$) where i is a new node (resp. an existing node). It creates the node i. i has no incoming nor outgoing edge and there is no basic concept (in Φ_0) such that i belongs to its valuation (resp. it deletes i and all its incoming and outgoing edges).
- a *global edge redirection* $i \gg j$ where i and j are nodes. It redirects all incoming edges of i toward j.

The result of performing the elementary action α on a graph $G = (N^G, E^G, C^G, R^G, L_N^G, L_E^G, s^G, t^G)$, written $G[\alpha]$, produces the graph $G' = (N^{G'}, E^{G'}, C^{G'}, R^{G'}, L_N^{G'}, L_E^{G'}, s^{G'}, t^{G'})$ as defined in Figure 3. An *action*, say α, is a sequence of elementary actions of the form $\alpha = a_1; a_2; \ldots; a_n$. The result of performing α on a graph G is written $G[\alpha]$. $G[a; \alpha] = (G[a])[\alpha]$ and $G[\epsilon] = G$, ϵ being the empty sequence.

If $\alpha = add_C(i,c)$ **then:**
$N^{G'} = N^G$,$E^{G'} = E^G$,$C^{G'} = C^G$,$R^{G'} = R^G$,$L_E^{G'} = L_E^G$

$$L_N^{G'}(n) = \begin{cases} L_N^G(n) \cup c & \text{if } n = i \\ L_N^G(n) & \text{if } n \neq i \end{cases}$$

$s^{G'} = s^G$, $t^{G'} = t^G$

If $\alpha = add_R(i,j,r)$ **then:**
$N^{G'} = N^G$, $C^{G'} = C^G$,$R^{G'} = R^G$, $L_N^{G'} = L_N^G$
$E^{G'} = E^G \cup e$ where e is a new element

$$L_E^{G'}(e') = \begin{cases} r & \text{if } e' = e \\ L_E^G(e') & \text{if } e' \neq e \end{cases}$$

$$s^{G'}(e') = \begin{cases} i & \text{if } e' = e \\ s^G(e') & \text{if } e' \neq e \end{cases}$$

$$t^{G'}(e') = \begin{cases} j & \text{if } e' = e \\ t^G(e') & \text{if } e' \neq e \end{cases}$$

If $\alpha = add_I(i)$ **then:**
$N^{G'} = N^G \cup i$ where i is a new node
$C^{G'} = C^G$, $R^{G'} = R^G$

$$L_N^{G'}(n') = \begin{cases} \emptyset & \text{if } n' = i \\ L_N^G(n') & \text{if } n' \neq i \end{cases}$$

$E^{G'} = E^G$, $L_E^{G'} = L_E^G$,$s^{G'} = s^G$, $t^{G'} = t^G$

If $\alpha = i \gg j$ **then:**
$N^{G'} = N^G$, $E^{G'} = E^G$, $C^{G'} = C^G$
$R^{G'} = R^G$, $L_N^{G'} = L_N^G$, $L_E^{G'} = L_E^G$,$s^{G'} = s^G$

$$t^{G'}(e) = \begin{cases} j & \text{if } t^G(e) = i \\ t^G(e) & \text{if } t^G(e) \neq i \end{cases}$$

If $\alpha = del_C(i,c)$ **then:**
$N^{G'} = N^G$,$E^{G'} = E^G$,$C^{G'} = C^G$,$R^{G'} = R^G$, $L_E^{G'} = L_E^G$

$$L_N^{G'}(n) = \begin{cases} L_N^G(n) \backslash c & \text{if } n = i \\ L_N^G(n) & \text{if } n \neq i \end{cases}$$

$s^{G'} = s^G$, $t^{G'} = t^G$

If $\alpha = del_R(i,j,r)$ **then:**
$N^{G'} = N^G$, $C^{G'} = C^G$,$R^{G'} = R^G$, $L_N^{G'} = L_N^G$
$E^{G'} = E^G \backslash \{e | s^G(e) = i \wedge t^G(e) = j \wedge L_E^G(e) = r\}$

$L_E^{G'}$ is the restriction of L_E^G to $E^{G'}$

$s^{G'}$ is the restriction of s^G to $E^{G'}$

$t^{G'}$ is the restriction of L_E^G to $E^{G'}$

If $\alpha = del_I(i)$ **then:**
$E^{G'} = E^G \backslash \{e | s^G(e) = i \vee t^G(e) = i\}$
$N^{G'} = N^G \backslash i$, $C^{G'} = C^G$,$R^{G'} = R^G$

$L_N^{G'}$ is the restriction of L_N^G to $N^{G'}$

$L_E^{G'}$ is the restriction of L_E^G to $E^{G'}$
$s^{G'}$ is the restriction of s^G to $E^{G'}$
$t^{G'}$ is the restriction of L_E^G to $E^{G'}$

Figure 3 Summary of the effects of elementary actions.

▶ **Definition 9** (Rule, LDRS). A *rule* ρ is a pair (lhs,α) where lhs, called the left-hand side, is an attributed graph with $\mathcal{C}2\mathcal{PDL}$ formulae as attributes and α, called the right-hand side, is an action. Rules are usually written $lhs \to \alpha$. A logically decorated rewriting system, LDRS, is a set of rules.

It is noteworthy that the left-hand side of a rule is an attributed graph, that is it can contain nodes labeled with $\mathcal{C}2\mathcal{PDL}$ formulae. This is not insignificant. Indeed, these formulae can express reachability expression (closure of a program), non-local properties (universal program), ... These node labelings allow to write more concise and simpler rewriting systems. For instance, Figure 4 provides two LDRSs, GRS_0 and GRS_1 which remove unreachable nodes from a start state (labeled by S) of an automaton. Without the closure constructor $(*)$, one would tag that the start states are reachable (label R) (rule ρ_{00}), then say that every neighbor of a reachable node is reachable (rule ρ_{01}) and finally that all nodes that have not been reached by applying the first two rules as much as possible are to be removed (rule ρ_{02}). GRS_0 requires the explicit computation of the reachability making the algorithm more complex. On the other hand, GRS_1 only uses one rule which says that all nodes that are not reachable from a start state are to be removed (rule ρ_1).

It is worth noting that the impact of $\mathcal{C}2\mathcal{PDL}$ formulae in node labels is not limited to graph rewriting. Indeed, let us consider the term rewriting system of integer arithmetic with multiplication. The classical way to deal with 0s in such a case is to have rules saying that $0 \times x \rightsquigarrow 0$ and $x \times 0 \rightsquigarrow 0$. Considering terms as trees, and thus as graphs, it is also possible to improve on this set of rules by using, for instance, the rule $i : \times \wedge \langle((L \cup R); \times?)^*\rangle 0 \rightsquigarrow i : 0$. This rule states that if a node i is such that $i : \times \wedge \langle((L \cup R); \times?)^*\rangle 0$, that is to say, node i is labeled by the multiplication operator $(i : \times)$ and there is a path of left- or right-operands $(L \cup R)$, crossing nodes labeled by the multiplication operator $(\times?)$, that leads to a node labeled by 0 $(i : \langle((L \cup R); \times?)^*\rangle 0)$, then node i could be labeled by 0 $(i : 0)$.

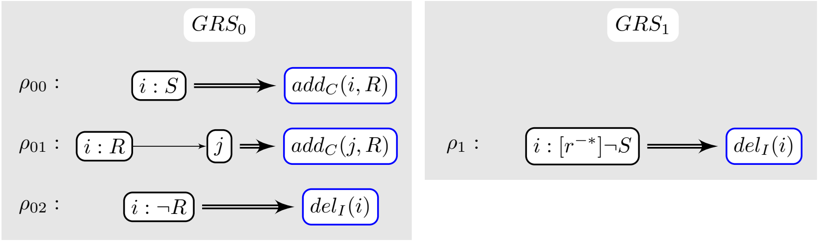


Figure 4 Two LDRS dealing with the suppression of unreachable nodes. Rules with atomic formulae are on the left. The rule with a non-atomic formula is on the right.

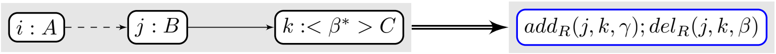


Figure 5 An example of LDRS. The dashed line represents the program (label) α and the plain line the program β. The first edge labeled β after an access α on a path toward a C gets a new tag γ.

In Figure 5, we provide an additional toy example which is used later. It consists of one rule which relabels the edge going from node j to k with label (program) γ whenever node k has access to "information" C. This rule may have different interpretations such as the modification of access policy to information tagged C.

▶ **Example 10.** Back to our running example, we provide a very simple graph rewriting system $\mathcal{R}$ that tries to produce a full and correct grid. It contains 16 rules, that are summarized in Figure 6. The rules $\rho_{r,J}$ make sure that when a line (resp. a column or a square) contains a cell with value J, P_J is no longer available for all the cells on the line (resp. column or square). The rules ρ_J are used to pick one choice among those that are available.

▶ **Definition 11** (Match). A *match* h between a left-hand side lhs and a graph G is a pair of functions $h = (h_N, h_E)$, with $h_N : N_{lhs} \to N_G$ and $h_E : E_{lhs} \to E_G$ such that:

1. $\forall n \in N_{lhs}, \forall c \in L_{N_{lhs}}(n), h_N(n) \models c$
2. $\forall e \in E_{lhs}, L_{E_{lhs}}(e) = L_{E_G}(h_E(e))$
3. $\forall e \in E_{lhs}, s_G(h_E(e)) = h_N(s_{lhs}(e))$
4. $\forall e \in E_{lhs}, t_G(h_E(e)) = h_N(t_{lhs}(e))$

The third and the fourth conditions are classical and say that the source and target functions and the match have to agree. The first condition says that for every node n of the left-hand side, the node to which it is associated, $h(n)$, in G has to satisfy every concept that n satisfies. This condition clearly expresses additional negative and positive conditions which are added to the "structural" pattern matching. The second one ensures that the match respects edge labeling.

▶ **Definition 12** (Rule Application). A graph G rewrites to graph G' using a rule $\rho = (lhs, \alpha)$ iff there exists a match h from lhs to G. G' is obtained from G by performing actions in $h(\alpha)$[2]. Formally, $G' = G[h(\alpha)]$. We write $G \to_\rho G'$ or $G \to_{\rho,h} G'$.

Confluence of graph rewriting systems is not easy to establish. For instance, orthogonal graph rewrite systems are not always confluent, see e.g.,[7]. That is why we use a notion rewrite strategies to control the use of possible rules. Informally, a strategy specifies the

[2] $h(\alpha)$ is obtained from α by replacing every node name,n, of lhs by $h(n)$.

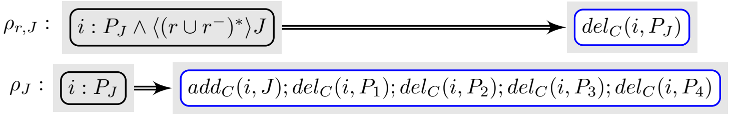

$\rho_{r,J}$:
$i : P_J \wedge \langle (r \cup r^-)^* \rangle J$
$del_C(i, P_J)$
ρ_J :
$i : P_J$
$add_C(i, J); del_C(i, P_1); del_C(i, P_2); del_C(i, P_3); del_C(i, P_4)$

Figure 6 A summary of the rules of $\mathcal{R}$. In $\rho_{r,J}$, r must be replaced by either R, C or SQ and J by 1, 2, 3 or 4. In ρ_J, J must be replaced by 1, 2, 3 or 4.

$$\frac{G \rightarrow_\rho G'}{G \Rightarrow_\rho G'}\ \text{(Rule)} \qquad \frac{G \Rightarrow_{s_0} G'' \quad G'' \Rightarrow_{s_1} G'}{G \Rightarrow_{s_0;s_1} G'}\ \text{(Strategy composition)}$$

$$\frac{G \Rightarrow_{s_0} G'}{G \Rightarrow_{s_0 \oplus s_1} G'}\ \text{(Choice left)} \qquad \frac{G \Rightarrow_{s_1} G'}{G \Rightarrow_{s_0 \oplus s_1} G'}\ \text{(Choice right)}$$

$$\frac{\neg \mathbf{App}(s, G)}{G \Rightarrow_{s^*} G}\ \text{(Closure false)} \qquad \frac{G \Rightarrow_s G'' \quad G'' \Rightarrow_{s^*} G' \quad \mathbf{App}(s, G)}{G \Rightarrow_{s^*} G'}\ \text{(Closure true)}$$

Figure 7 Strategy application rules.

application order of different rules. It does not point to where the matches are to be found nor does it ensure the unicity of the reduction outcome.

▶ **Definition 13** (Strategy)**.** Given a graph rewriting system $\mathcal{R}$, a *strategy* is a non-empty word of the following language defined by s:

$s := \quad \rho \quad \text{(Rule)} \mid s; s \quad \text{(Composition)} \mid s \oplus s \quad \text{(Choice)} \mid s^* \quad \text{(Closure)}$

where ρ is any rule in $\mathcal{R}$.

We write $G \Rightarrow_{\mathcal{S}} G'$ when G rewrites to G' following the rules given by the strategy $\mathcal{S}$.

Informally, the strategy "$\rho_1; \rho_2$" means that rule ρ_1 should be applied first, followed by the application of rule ρ_2. The strategy "$\rho_0^*; (\rho_1 \oplus \rho_2)$" means that rule ρ_0 is applied as far as possible, then followed either by ρ_1 or ρ_2. It is worth noting that the closure is the standard "while" construct: if the strategy we use is s^*, the strategy s is used as long as it is possible and not an undefined number of times.

▶ **Example 14.** For the Sudoku example, a possible strategy $\mathcal{S}_1$ could be: "As long as one can eliminate possibilities, do it. Then, when it is no longer the case, make a choice in one of the blank cells and go back to the first step". $\mathcal{S}_1$ may be defined as $\mathcal{S}_1 = (\bigoplus_{r\in\{R,C,SQ\},J\in[1,4]} \rho_{r,J})^*; ((\bigoplus_{J\in[1,4]} \rho_J); (\bigoplus_{r\in\{R,C,SQ\},J\in[1,4]} \rho_{r,J})^*)^*$.

In Figure 7, we provide the rules that specify how strategies are used to rewrite a graph. For that, we use the predicate $\mathbf{App}(s, G)$ which holds whenever graph G can be rewritten by the strategy s. It is defined as follows:

$\mathbf{App}(\rho, G) = \mathit{true}$ iff there exists a match h from the left-hand side of ρ to G
$\mathbf{App}(s_0 \oplus s_1, G) = \mathbf{App}(s_0, G) \vee \mathbf{App}(s_1, G)$
$\mathbf{App}(s_0^*, G) = \mathit{true}$
$\mathbf{App}(s_0; s_1, G) = \mathbf{App}(s_0, G)$

It is worth noting that $\mathbf{App}(s, G)$ is not meant to denote that the whole strategy can be applied to G, just that the next step can be applied. Indeed, let's assume the strategy $s = s_0; s_1$ where s_0 can be applied but may lead to a state where s_1 cannot. In this case $\mathbf{App}(s, G)$ will hold saying that the strategy s can be applied on G.

$$\begin{array}{ll}
wp(add_C(i,c),\ Q) = Q[add_C(i,c)] & wp(del_C(i,c),\ Q) = Q[del_C(i,c)] \\
wp(add_R(i,j,r),\ Q) = Q[add_R(i,j,r)] & wp(del_R(i,j,r),\ Q) = Q[del_R(i,j,r)] \\
wp(add_I(i),\ Q) = Q[add_I(i)] & \bullet\ wp(del_I(i),\ Q) = Q[del_I(i)] \\
wp(i >> j,\ Q) = Q[i >> j] & wp(\epsilon,\ Q) = Q \\
wp(a;\alpha,\ Q) = wp(a, wp(\alpha, Q)) &
\end{array}$$

Figure 8 Weakest pre-conditions for actions.

$$\begin{array}{ll}
wp(s_0; s_1,\ Q) = wp(s_0, wp(s_1,\ Q)) & wp(s_0 \oplus s_1,\ Q) = wp(s_0, Q) \wedge wp(s_1, Q) \\
wp(s^*,\ Q) = inv_s & wp(\rho,\ Q) = App(tag(\rho)) \Rightarrow wp(tag(\alpha_\rho), Q)
\end{array}$$

Figure 9 Weakest pre-conditions for strategies.

4 Proving Correctness of LDRS's

Equational reasoning and structural induction method represent the main core of proof techniques dedicated to reason about term rewrite systems. Unfortunately, graph rewrite systems do not benefit yet from such established techniques. For example, generalization of equational reasoning to graph rewriting systems is not complete [6]. In this section, we propose a way to specify properties of LDRSs for which we establish a decidable proof procedure.

▶ **Definition 15** (Specification). A *specification* SP is a tuple $(Pre,\ Post,\ \mathcal{R},\ \mathcal{S})$ where Pre and $Post$ are $\mathcal{C2PDL}$ formulas, $\mathcal{R}$ is a graph rewriting system and $\mathcal{S}$ is a strategy.

A specification SP is said to be *correct* iff for all graphs G, G' such that $G \Rightarrow_{\mathcal{S}} G'$ and $G \models Pre$, then $G' \models Post$.

In order to show the correctness of a specification, we follow a Hoare-calculus style and compute the weakest pre-condition $wp(\mathcal{S}, Post)$. For that, we define the weakest pre-conditions of a formula $Post$ induced by a strategy, a rule, an action and an elementary action. They are presented in Figure 8 for the elementary actions and Figure 9 for strategies.

The weakest pre-condition of an elementary action, say a, and a post-condition Q is defined as $wp(a, Q) = Q[a]$ where $Q[a]$ stands for the pre-condition consisting of Q to which is applied a substitution induced by the action a that we denote by $[a]$. The notion of substitution used here is the one of Hoare-calculi [11].

▶ **Definition 16** (Substitutions). To each elementary action a is associated a *substitution*, written $[a]$, such that for any graph G, $(G \models \phi[a]) \Leftrightarrow (G[a] \models \phi)$.

It is worth noting that substitutions are, in all generality, not defined as formulae of $\mathcal{C2PDL}$. They are defined as a new constructor whose meaning is that the weakest pre-conditions as defined above are correct. There is no reason whatsoever to think that the addition of a constructor for substitutions is harmless, in general. It is a very interesting problem to figure out for which logics they can be introduced as it is a strong indication that such a logic may be suitable to study the correction of programs. It is one of the central results of [5] that $\mathcal{C2PDL}$ is closed under substitutions allowing us to use them freely in the following.

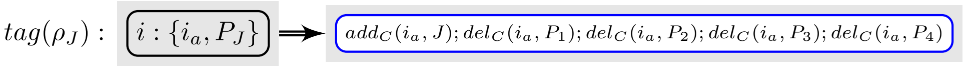

$tag(\rho_J)$:
$i : \{i_a, P_J\}$
$add_C(i_a, J); del_C(i_a, P_1); del_C(i_a, P_2); del_C(i_a, P_3); del_C(i_a, P_4)$

Figure 10 The rule ρ_J is modified into $tag(\rho_J)$ with $tag(i) = i_a$.

Apart from $wp(\rho,\ Q)$, the weakest pre-conditions defined in here are the usual ones. As in any other framework using Hoare logic, it requires the definition of an invariant(inv) for each loop that has to be provided by the user.

The weakest pre-condition of a rule $\rho = (lhs_\rho, \alpha_\rho)$ and a post-condition Q is given by $wp(\rho,\ Q) = App(tag(\rho)) \Rightarrow wp(tag(\alpha_\rho), Q)$ where $tag : N_{lhs_\rho} \to \Sigma$ is a function which associates to every node, n, of the left-hand side of rule ρ, a fresh name in Σ. These new names are used to keep track of the matching locations within potential graphs rewritten by different instances of ρ during the execution of a strategy. $tag(\rho) = (tag(lhs_\rho), tag(\alpha_\rho))$ where $tag(lhs_\rho)$ is a named graph which consists of the graph lhs_ρ where the node labeling function is augmented by tag, i.e., for all nodes, n, of lhs_ρ, $L_{N_{tag(lhs_\rho)}}(n) = L_{N_{lhs_\rho}}(n) \cup tag(n)$. $tag(\alpha_\rho)$ is obtained from α_ρ by substituting every node (in N_{lhs_ρ}), say i, by $tag(i)$. Figure 10 gives an example turning the left-hand side of a rule into a named graph via a function tag.

The formula $App(tag(\rho))$ expresses the applicability of the rule $tag(\rho)$. In other words, it expresses the existence of a match of the left-hand side of $tag(\rho)$. More precisely $App(tag(\rho))$ is defined as follows:

$$App(tag(\rho)) = \phi_{nodes} \wedge \phi_{edges}$$

where

$$\phi_{nodes} = \bigwedge_{u \in tag(N_{lhs_\rho})} \langle \nu_{\Sigma_1} \rangle (u \wedge \bigwedge_{\phi \in L_N^{\text{þ}(lhs_\rho)}(\theta(u))} \phi)$$

and

$$\phi_{edges} = \bigwedge_{e \in E | s(e) = \theta(u)} \langle \nu_{\Sigma_1} \rangle (u \wedge \langle L_E^{\text{þ}(lhs_\rho)}(e) \rangle tag(t(e)) \wedge$$
$$\bigwedge_{e \in E | t(e) = \theta(u)} \langle \nu_{\Sigma_1} \rangle (u \wedge \langle L_E^{\text{þ}(lhs_\rho)}(e)^- \rangle tag(s(e))\,.$$

ϕ_{nodes} states that the first condition of Definition 11 is satisfied i.e., all formulae satisfied by a node of the left-hand side have to be satisfied by its image in the graph. ϕ_{edges} does the same with edges: the label of the corresponding edges are the same and the source and target functions fulfill the matching compatibly condition.

Tagging a rule may seem to reduce its applicability. Indeed, by choosing a new name for each node of the left-hand side, the rule can now be applied only at the nodes of the graph named accordingly. Let ρ be a rule such that $LHS_\rho = (N_\rho = \{i_0, \dots i_n\}, E_\rho, \mathcal{C}_\rho, \mathcal{R}_\rho, L_{N_\rho}, L_{E_\rho}, s_\rho, t_\rho)$ is its left-hand side. In order to prove that the application of ρ on a graph $G = (N_G, E_G, \mathcal{C}_G, \mathcal{R}_G, L_{N_G}, L_{E_G}, s_G, t_G)$ is correct, one has to verify that for every match $h = (h_N, h_E)$ (as defined in Definition 11), the post-condition is satisfied after the transformation associated with ρ is applied at the $h_N(i_k)$'s. Instead of showing that, the verification procedure proves that for any graph, if the rule can be applied at the $\theta(u)$'s, where θ is the function that associates to each name of Σ a node of G and $u \in tag(\rho)$, the post-condition is satisfied after performing the transformation. As the u's are fresh names, they do not have any impact on the previous characterization of the graphs. Thus, the validity of

$$\begin{array}{ll} vc(\rho,\ Q) = \top & vc(s_0; s_1,\ Q) = vc(s_0, wp(s_1,\ Q)) \wedge vc(s_1, Q) \\ vc(s_0 \oplus s_1,\ Q) = vc(s_0, Q) \wedge vc(s_1, Q) & \\ vc(s^*,\ Q) = (inv_s \wedge \mathit{NApp(s)} \Rightarrow Q) & \wedge\ (inv_s \Rightarrow wp(s, inv_s)) \wedge vc(s, inv_s) \end{array}$$

Figure 11 Verification conditions.

$wp(\rho, Post)$ actually states that whatever the choice of θ is, if the rule can be applied, then the post-condition will be satisfied after it is fired.

The weakest pre-condition associated to the closure of a strategy, say s^*, and a post-condition Q is defined as $wp(s^*,\ Q) = inv_s$ where inv_s is an invariant ($\mathcal{C2PDL}$ formula) associated to the strategy s. Roughly speaking s^* could be seen as a while-loop which needs invariants associated with additional proof obligations defined by means of the function vc (verification conditions) given in Figure 11.

Informally, the predicate *NApp(s)*, used in the definition of vc, the verification conditions function, says that the strategy s cannot be applied. To express *NApp(s)*, one may wonder whether it is possible to use the negation of $App(s)$. The answer is negative since App has been defined using dedicated names, that is to say a rule is applied at a specific place defined by the added names introduced by the function tag. Intuitively, if one wants to express that there is no match for a rule whose left-hand side contains a cycle (e.g., the second graph of Figure 12), then universal quantification is required. For instance, in that case it would be $\forall i, j.[\nu_{\Sigma 1}](i \Rightarrow [R](j \Rightarrow [R]\neg i))$. One of them can be discarded to produce the expression $\forall i.[\nu_{\Sigma 1}](i \Rightarrow [R][R]\neg i)$. Alas, this cannot be expressed in $\mathcal{C2PDL}$. The names only allow to express existential quantifiers. Indeed, to express universal quantifiers, one needs to extend the logic with the binder $\downarrow$ of hybrid logic which is enough to make the logic undecidable[2].

To express formally *NApp(s)* in $\mathcal{C2PDL}$, one needs to introduce some additional definitions first.

▶ **Definition 17** (Explicitly Named Nodes, Non-Oriented Cycles)**.** An *explicitly named node* is a node such that each disjunct of the disjunctive normal form of its label contains a name. A *non-oriented cycle*, c, is a finite list of nodes $c = [n_0, \ldots, n_k]$ such that (i) $n_k = n_0$ and (ii) $\forall \kappa \in [0, k-1], \exists r \in \Pi_0$ such that $(n_\kappa, n_{\kappa+1}) \in R(r)$ or $(n_{\kappa+1}, n_\kappa) \in R(r)$.

▶ **Definition 18** (Grove and Thicket)**.** A *grove* is a disjoint union of thickets. A *thicket* is a connected graph such that it does not contain any non-oriented cycle composed only of non explicitely named nodes. We call a strategy $\mathcal{S}$ relative to a rewriting system $\mathcal{R}$ a *grove-strategy* if the left-hand sides of all the rules appearing under a closure are groves.

Let lhs be a left-hand side. Let us split N_{lhs} into T_E, the set of explicitly named nodes, and $T_I = N_{lhs} \backslash T_E$. For each maximally connected subgraph composed only of nodes in T_I, a distinguished node r_i is selected. If there is a maximally connected subgraph composed only of explicitly named nodes, an r_i is also picked for it. Now, everything is ready to define *NApp(s)*.

1. $NApp(\rho) = \bigvee_{r_i} [\nu_{\Sigma_1}] NA(r_i, \emptyset)$
2. $NA(n, V) = (\bigvee_{\phi \in L_N(n)} \neg\phi) \vee (\bigvee_{e \in E | s(e) = n | s(e) \in T_E \cup (T_I \backslash V)} [L_E(e)] NA(t(e), V \cup \{n\})) \vee$ $(\bigvee_{e \in E | t(e) = n | t(e) \in T_E \cup (T_I \backslash V)} [L_E(e)^-] NA(s(e), V \cup \{n\}))$ if $n \notin V$
3. $NA(n, V) = \neg\mu(n)$ if $n \in T_E \cap V$
4. $NApp(s_0 \oplus s_1) = NApp(s_0) \wedge NApp(s_1)$
5. $NApp(s^*) = false$
6. $NApp(s_0; s_1) = NApp(s_0)$

Figure 12 These two graphs are indistinguishable without names.

Rule 2 is the most involved. $NA(n, V)$ is used to describe what has to be true for a node different from n. V is used to track which nodes have already been visited. $\bigvee_{\phi \in L_N(n)} \neg\phi$ states that there is at least one of the formulae satisfied by n that is not satisfied while $\bigvee_{e \in E | s(e)=n | s(e) \in T_E \cup (T_I \setminus V)} [L_E(e)] NA(t(e), V \cup \{n\})$ means that there is a neighbor of n using an outgoing edge that cannot find a match. $\bigvee_{e \in E | t(e)=n | t(e) \in T_E \cup (T_I \setminus V)} [L_E(e)^-] NA(s(e), V \cup \{n\})$ has the same signification but for incoming edges. Rule 3 is used to recall the names of the explicitly named nodes that have already been visited without creating a cycle in the execution. Rules 4 to 6 are exactly the negation of $App(s, G)$. Rule 1 says that for at least one r_i, it is not possible to find a node that would be a match. It is noteworthy that there is no rule for $N \in T_I \cap V$ since the considered left-hand side has to be a grove.

▶ **Example 19.** The rules of the Sudoku example are simple, having only one-node left-hand side, and thus not very interesting as far as $NApp$ is concerned. We will thus consider the rule, say ρ, of Figure 5. There is only one connected subgraph so one has to pick one distinguished node r_0. Let us choose, randomly, the node j as r_0. Then $NApp(\rho) = [\nu_{\Sigma_1}] NA(j, \emptyset)$. As j is not explicitly named, $NA(j, \emptyset) = \neg B \vee [\beta] NA(k, \{j\}) \vee [\alpha^-] NA(i, \{j\})$. As i is not explicitly named either, $NA(i, \{j\}) = \neg A$ as the only neighbor of i is j. Finally, $NA(k, \{j\}) = [\beta^*]\neg C$ as the only neighbor of k is j. Thus $NApp(\rho) = [\nu_{\Sigma_1}](\neg B \vee [\beta][\beta^*]\neg C \vee [\alpha^-]\neg A)$.

Given a specification SP, the correctness of SP amounts to verify the validity of the formula $Corr_{SP} = vc(\mathcal{S}, post) \wedge (pre \Rightarrow wp(\mathcal{S}, post))$. It can be shown that $Corr_{SP}$ is a $\mathcal{C2PDL}$ formula due to the closure by substitutions of $\mathcal{C2PDL}$ [5]. Therefore we can state the soundness of our calculus in the following theorem.

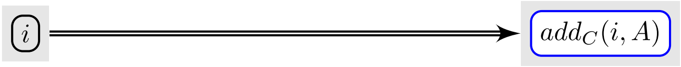
▶ **Theorem 20.** *Let $SP = (Pre,\ Post,\ \mathcal{R},\ \mathcal{S})$ be a specification where Pre and Post are $\mathcal{C2PDL}$ formulas, $\mathcal{S}$ is a grove-strategy relative to LDRS $\mathcal{R}$. If the formula $Corr_{SP} = vc(\mathcal{S}, post) \wedge (pre \Rightarrow wp(\mathcal{S}, post))$ is valid then for all graphs G and G', if $G \Rightarrow_{\mathcal{S}} G'$, then $G \models pre$ implies $G' \models post$.*

$Corr_{SP}$ being a formula of $\mathcal{C2PDL}$, one gets the following corollary from theorem 7.

▶ **Corollary 21.** *Let $SP = (Pre,\ Post,\ \mathcal{R},\ \mathcal{S})$ be a specification where Pre and Post are $\mathcal{C2PDL}$ formulas, $\mathcal{S}$ is a grove-strategy relative to LDRS $\mathcal{R}$. The verification of the correctness of SP is decidable.*

▶ **Example 22.** The example of Figure 13 contains one rule that is applied as long as possible. As the closure is used, an invariant has to be specified. We choose $\langle \nu_{\Sigma_1} \rangle x$ which is obviously an invariant of the loop. The specifications differ on the post-condition, the first one being what we would expect, that is all elements are labelled A, and the other one being the exact opposite. Both specifications are deemed partially correct. Indeed, as $wp(\rho_1^*, Post) = inv = \langle \nu_{\Sigma_1} \rangle x$, $Pre \Rightarrow wp(\rho_1^*, Post) = \top$. Furthermore, as $vc(\rho_1^*, Post) = (inv \wedge App(\rho_1) \Rightarrow inv[add_C(i, A)]) \wedge (inv \wedge NApp(\rho_1) \Rightarrow Post) \wedge vc(\rho_1, inv)$ and $vc(\rho_1, inv) = \top$ and $inv \Rightarrow inv = \top$, $vc(\rho_1^*, Post) = inv \wedge NApp(\rho_1) \Rightarrow Post)$. But then, as $NApp(\rho_1) = [\nu_{\Sigma_1}]\bot$, $inv \wedge NApp(\rho_1) = \bot$ and thus $vc(\rho_1^*, Post) = \top$ independently of the post-condition. The fact that two specifications leading to opposite results would be both considered correct

ρ_1 :



$$SP_{10} = (\langle \nu_{\Sigma_1} \rangle (x \wedge \neg A), [\nu_{\Sigma_1}]A, \{\rho_1\}, \rho_1^*)$$

$$SP_{11} = (\langle \nu_{\Sigma_1} \rangle (x \wedge \neg A), \langle \nu_{\Sigma_1} \rangle \neg A, \{\rho_1\}, \rho_1^*)$$

Figure 13 Scholastic examples of specification. Both of the specifications are partially correct.

seems to be an error but the fact is that the program will never stop and, actually, no result is ever reached.

It is noteworthy that the choice of the invariant is, as usual in Hoare logic [11], far from innocent. Let's assume we chose $inv = \top$ instead. Then, the only difference is that now $inv \wedge NApp(\rho_1) = NApp(\rho_1) = [\nu_{\Sigma_1}]\bot$. But then, $inv \wedge NApp(\rho_1) \Rightarrow [\nu_{\Sigma_1}]\neg A$ is true but not $inv \wedge NApp(\rho_1) \Rightarrow \langle \nu_{\Sigma_1} \rangle A$. A poor choice of inv can prevent someone from proving a program correct.

The example of the Sudoku can be proven to be valid with a suitable choice of Pre, $Post$ and inv.

5 Related work

One of the main features of the class of rewriting systems we introduce in this paper consists in specifying application conditions as logic formulas that label nodes of the left-hand sides. This is, to our knowledge, a new approach to specify application conditions. In [9] the idea of additional (negative) application conditions has been introduced in a framework based on category theory.

Our second contribution is to introduce a decidable verification procedure for the considered rewrite systems and the logic used to express system properties. The problem of reasoning about graph transformations is known to be complex in its full generality [12].

One approach to program verification, as exemplified by [10, 16], is similar to the one we pursue here, i.e., the goal is to generate the weakest pre-condition for a condition to be satisfied. Our method strongly diverges from theirs in very key points, though. First and foremost, their rewriting systems are based on algebraic methods whereas ours are purely algorithmic. Secondly, our graphs are logically attributed, that is each node is labeled with formulas. Another difference lies in the considered logics. Indeed, the logic presented in [16] is equivalent to monadic second-order logic. It thus allows to express second-order quantification which is out of the scope of our logic. The one in [10] is equivalent to first-order graph formulas whose expressivity is not comparable to ours (i.e., there are formulas that can be expressed in one logic but not in the other and vice versa). Both monadic second-order and first-order graph logics are undecidable. On the other hand, we have carefuly chosen an extension of dynamic logic that preserves decidability. Last but not least, the way they compute the pre-conditions is also much different from ours. We use the notion of substitutions whereas their conditions are built incrementally on the rules.

In [13], the authors also use a stronger logic, monadic second-order logic again, and the verification problem is decidable. This is due to the fact that their transformations are bisimulation-generic whereas ours are not. Furthermore they do not label nodes. In both cases, the core of the reflexion is centered in modifications of the structure of the data (that is how general classes of nodes and edges interact). On the other hand, we are able to do the core of these transformation but our focus is on localized modifications that is a change at the instance level.

Another approach to the verification of graph transformation is the classical model-checking applied in the case where states are specified by graphs [17]. This approach, which needs the development of a possibly infinite transition systems, departs from ours.

A verification method of graph transformation is tackled in [14] where a set of forbidden graphs is given and where context-free graph grammars are used instead of logics to define post-conditions. This method is quite different from ours.

In [3], a logic dedicated to mimic graph transformations has been introduced. That logic can also be used to specify properties over graph transformations. The expressive power of that logic is very rich but its validity problem is not decidable.

In [6], an extension of equational logic to graph transformation has been investigated. Theories generated by such logic are not recursively enumerable in general. Thus no completeness nor decidability results can be expected. In addition, in the considered logic bisimilar graphs cannot be distinguished.

The modification of Knowledge Bases is a much different but very active field. Belief revision [1] deals with the addition of new knowledge and how to modify the Knowledge Base so that it is still consistent. That means that a lot of modifications may be hidden in one update action. This is quite a different approach as we want, on the other hand, to know exactly what actions are performed and use that to define our new knowledge.

The work in [4] is heading toward the same goal as the present paper but both the programming language, that is slightly less expressive and imperative, and the considered logic are different.

6 Conclusion

We have presented a new class of graph rewrite systems, LDRSs, where left-hand sides can express additional application conditions defined as $\mathcal{C2PDL}$ formulas. We defined computations with these systems by means of rewrite strategies. There is certainly much work to be done around such systems with logically decorated left-hand sides. For instance, the extension to narrowing derivations, which is a matter of future work, would use an involved unification algorithm taking into account the underlying logic. We have also presented a sound Hoare-like calculus for specifications with pre- and post-conditions in $\mathcal{C2PDL}$ and show that the considered correctness problem is decidable. These positive achievements deserve to be extended to other logics which we intend to investigate in the future.

References

1 Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *J. Symb. Log.*, 50(2):510–530, 1985. doi:10.2307/2274239.

2 Carlos Areces and Balder ten Cate. Hybrid logics. In *Studies in Logic and Practical Reasoning*, volume 3, pages 821–868. Elsevier, 2007.

3 Philippe Balbiani, Rachid Echahed, and Andreas Herzig. A dynamic logic for termgraph rewriting. In *5th International Conference on Graph Transformations (ICGT)*, volume 6372 of *Lecture Notes in Computer Science*, pages 59–74. Springer, 2010.

4 Jon Haël Brenas, Rachid Echahed, and Martin Strecker. A Hoare-like calculus using the SROIQ σ logic on transformations of graphs. In *Theoretical Computer Science – 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*, pages 164–178, 2014. doi:10.1007/978-3-662-44602-7_14.

5 Jon Hael Brenas, Rachid Echahed, and Martin Strecker. $\mathcal{C2PDLS}$: A combination of combinatory and converse PDL with substitutions. 2015. URL: `http://lig-membres.imag.fr/echahed/c2pdls.pdf`.

6 Ricardo Caferra, Rachid Echahed, and Nicolas Peltier. A term-graph clausal logic: Completeness and incompleteness results. *Journal of Applied Non-classical Logics*, 18(4):373–411, 2008.

7 Rachid Echahed. Inductively sequential term-graph rewrite systems. In *4th International Conference on Graph Transformations, ICGT*, volume 5214 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2008.

8 Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979. `doi:10.1016/0022-0000(79)90046-1`.

9 Annegret Habel, Reiko Heckel, and Gabriele Taentzer. Graph grammars with negative application conditions. *Fundam. Inform.*, 26(3/4):287–313, 1996. `doi:10.3233/FI-1996-263404`.

10 Annegret Habel and Karl-Heinz Pennemann. Correctness of high-level transformation systems relative to nested conditions. *MSCS*, 19:245–296, 2009.

11 C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969. `doi:10.1145/363235.363259`.

12 Neil Immerman, Alex Rabinovich, Tom Reps, Mooly Sagiv, and Greta Yorsh. The boundary between decidability and undecidability for transitive-closure logics. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic*, volume 3210 of *Lecture Notes in Computer Science*, pages 160–174. Springer Berlin / Heidelberg, 2004. URL: `http://www.cs.umass.edu/~immerman/pub/cslPaper.pdf`.

13 Kazuhiro Inaba, Soichiro Hidaka, Zhenjiang Hu, Hiroyuki Kato, and Keisuke Nakano. Graph-transformation verification using monadic second-order logic. In *Proceedings of the 13th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 20-22, 2011, Odense, Denmark*, pages 17–28, 2011. `doi:10.1145/2003476.2003482`.

14 Barbara König and Javier Esparza. Verification of graph transformation systems with context-free specifications. In *Graph Transformations – 5th International Conference, ICGT 2010, Enschede, The Netherlands, September 27 – October 2, 2010. Proceedings*, pages 107–122, 2010. `doi:10.1007/978-3-642-15928-2_8`.

15 Solomon Passy and Tinko Tinchev. An essay in combinatory dynamic logic. *Inf. Comput.*, 93(2):263–332, 1991. `doi:10.1016/0890-5401(91)90026-X`.

16 Christopher M. Poskitt and Detlef Plump. Verifying monadic second-order properties of graph programs. In *Graph Transformation – 7th International Conference, ICGT 2014, Held as Part of STAF 2014, York, UK, July 22-24, 2014. Proceedings*, pages 33–48, 2014. `doi:10.1007/978-3-319-09108-2_3`.

17 Arend Rensink, Ákos Schmidt, and Dániel Varró. Model checking graph transformations: A comparison of two approaches. In *Graph Transformations, Second International Conference, ICGT 2004, Rome, Italy, September 28 – October 2, 2004, Proceedings*, pages 226–241, 2004. `doi:10.1007/978-3-540-30203-2_17`.

18 Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations.* World Scientific, 1997.

New Results on Morris's Observational Theory: The Benefits of Separating the Inseparable

Flavien Breuvart[1], Giulio Manzonetto[2], Andrew Polonsky[3], and Domenico Ruoppolo[4]

1 **Inria Sophia Antipolis, France**
`flavien.breuvart@inria.fr`

2 **Université Paris 13, Laboratoire LIPN, CNRS UMR 7030, France**
`giulio.manzonetto@lipn.univ-paris13.fr`

3 **Université Paris Diderot, Laboratoire IRIF, CNRS UMR 8243, France**
`andrew.polonsky@gmail.com`

4 **Université Paris 13, Laboratoire LIPN, CNRS UMR 7030, France**
`domenico.ruoppolo@lipn.univ-paris13.fr`

Abstract

Working in the untyped lambda calculus, we study Morris's λ-theory $\mathcal{H}^+$. Introduced in 1968, this is the original extensional theory of contextual equivalence. On the syntactic side, we show that this λ-theory validates the ω-rule, thus settling a long-standing open problem. On the semantic side, we provide sufficient and necessary conditions for relational graph models to be fully abstract for $\mathcal{H}^+$. We show that a relational graph model captures Morris's observational preorder exactly when it is extensional and λ-*König*. Intuitively, a model is λ-König when every λ-definable tree has an infinite path which is witnessed by some element of the model.

Both results follows from a weak separability property enjoyed by terms differing only because of some infinite η-expansion, which is proved through a refined version of the Böhm-out technique.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Lambda calculus, relational models, fully abstract, Böhm-out, ω-rule

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.15

1 Introduction

The problem of determining when two programs are equivalent is crucial in computer science: for instance, it allows to verify that the optimizations performed by a compiler preserve the meaning of the input program. For λ-calculi, it has become standard to regard two λ-terms M and N as equivalent when they are contextually equivalent with respect to some fixed set $\mathcal{O}$ of *observables*. This means that one can plug either M or N into any context $C[-]$ without noticing any difference in the global behaviour: $C[M]$ reduces to an observable in $\mathcal{O}$ exactly when $C[N]$ does. The underlying intuition is that the terms in $\mathcal{O}$ represent sufficient stable amounts of information coming out of the computation. The problem of working with this definition, is that the quantification over all possible contexts is difficult to handle. Therefore, various researchers undertook a quest for characterizing observational equivalences both semantically, by defining fully abstract denotational models, and syntactically, by comparing (possibly infinite) trees representing the programs executions.

The observational equivalence obtained by considering the λ-terms in head normal form as observables is by far the most famous and well studied since it enjoys many interesting properties. By definition, it corresponds to the λ-theory $\mathcal{H}^*$ which is the greatest sensible

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 15; pp. 15:1–15:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

consistent λ-theory. As shown in [1, Thm. 16.2.7], two λ-terms are equivalent in $\mathcal{H}^*$ exactly when their Böhm trees are equal up to denumerable many η-expansions of (possibly) infinite depth. These kinds of characterizations based on infinite trees have been recently rewritten using the modern approach of infinitary rewriting, thus initiating an interesting line of research [25]. From a semantic perspective, it is well known that $\mathcal{H}^*$ is the theory induced by Scott's model $\mathcal{D}_\infty$, a result first reported in [14, 26]. In other words, the model $\mathcal{D}_\infty$ is fully abstract for $\mathcal{H}^*$. Until recently, researchers were only able to introduce individual models of $\mathcal{H}^*$ [11], or at best to provide sufficient conditions for models living in some class to be fully abstract [18]. A substantial advance was made by Breuvart in [4] where he proposed the notion of *hyperimmune* model of λ-calculus, and showed that a continuous K-model is fully abstract for $\mathcal{H}^*$ iff it is extensional and hyperimmune, thus providing a characterization.

In the present paper we study Morris's observational equivalence [21] generated by considering the β-normal forms as observables, and we denote by $\mathcal{H}^+$ the corresponding λ-theory. (This λ-theory is denoted by $\mathcal{T}_{\mathrm{NF}}$ in Barendregt's book [1].) The λ-theory $\mathcal{H}^+$ is extensional and sensible; therefore, as $\mathcal{H}^*$ is maximal, we have $\mathcal{H}^+ \subsetneq \mathcal{H}^*$. Even if it has been less ubiquitously studied in the literature, also the equality in $\mathcal{H}^+$ has been characterized both syntactically, in terms of trees, and semantically. Indeed, two λ-terms are equivalent in $\mathcal{H}^+$ if and only if their Böhm trees are equal up to denumerable many η-expansions of finite depth [13], and this holds exactly when they have the same interpretation in Coppo, Dezani and Zacchi's filter model [6]. More recently, Manzonetto and Ruoppolo defined a class of relational graph models (rgms, for short), and proved that every extensional rgm preserving the polarities of the empty multiset (in a technical sense) is fully abstract for $\mathcal{H}^+$.

Inspired by the work done in [4], we are going to strengthen this result and provide sufficient and necessary conditions on rgms to induce $\mathcal{H}^+$ as λ-theory. Now, as all extensional rgms equate *at least as* $\mathcal{H}^+$, the difficult part is to find a condition guaranteeing that they do not equate more. In other words, we need to analyze in detail the equations in $\mathcal{H}^* \setminus \mathcal{H}^+$. We show that if two λ-terms M, N are equal in $\mathcal{H}^*$, but not in $\mathcal{H}^+$, then their Böhm trees are similar but there exists a position σ where they differ because of an infinite η-expansion of a variable x, that follows the structure of some computable infinite tree T. Thanks to a refined (almost chirurgical) Böhm-out technique, we prove that it is always possible to extract such a difference by defining a suitable context $C[-]$. To ensure that this difference is still detectable in an rgm, we introduce the notion of *λ-König model*: intuitively, an rgm is λ-König when every computable infinite tree T has an infinite path (which always exists by König's lemma) witnessed by some element of the model. In our main result (Theorem 36) we prove that an rgm $\mathcal{D}$ is fully abstract for $\mathcal{H}^+$ iff it is extensional and λ-König, thus providing a complete characterization. From our syntactic weak separation theorem it also follows that $\mathcal{H}^+$ satisfies the ω-rule, a property of extensionality stronger than the η-rule. Hence, our Theorem 40 answers positively this longstanding open problem, and brings us closer to the solution of Sallé's conjecture $\mathcal{B}\omega \subsetneq \mathcal{H}^+$ (cf. [1, Thm. 17.4.16]).

2 Preliminaries

Sequences and Trees

We denote by $\mathbb{N}$ the set of natural numbers and by $\mathbb{N}^{<\omega}$ the set of finite sequences over $\mathbb{N}$. The empty sequence is denoted by ε.

Let $\sigma = \langle n_1, \ldots, n_k\rangle$ and $\tau = \langle m_1, \ldots, m_{k'}\rangle$ be two sequences and let $n \in \mathbb{N}$. We write:

- $\ell(\sigma)$ for the length of σ,
- $\sigma.n$ for the sequence $\langle n_1, \ldots, n_k, n\rangle$,
- $\sigma \star \tau$ for the concatenation of σ and τ, that is for the sequence $\langle n_1, \ldots, n_k, m_1, \ldots, m_{k'}\rangle$.

We say that σ is a *subsequence of* τ, denoted $\sigma \subseteq \tau$, when $\tau = \sigma \star \sigma'$ for some $\sigma' \in \mathbb{N}^{<\omega}$. Given a map $f : \mathbb{N} \to \mathbb{N}$, its *prefix of length* n is the sequence $\langle f|n\rangle := \langle f(0), \dots, f(n-1)\rangle$.

▶ **Definition 1.** A *tree* is a partial function $T : \mathbb{N}^{<\omega} \rightharpoonup \mathbb{N}$ such that $\mathsf{dom}(T)$ is closed under prefixes and for all $\sigma \in \mathsf{dom}(T)$ and $n \in \mathbb{N}$ we have $\sigma.n \in \mathsf{dom}(T)$ if and only if $n < T(\sigma)$.

The elements of $\mathsf{dom}(T)$ are called *positions.* For all $\sigma \in \mathsf{dom}(T)$, $T(\sigma)$ gives the number of children of the node in position σ; therefore $T(\sigma) = 0$ when σ corresponds to a leaf.

Let T be a tree. We say that T is: *recursive* if the function T is partial recursive; *finite* if $\mathsf{dom}(T)$ is finite; *infinite* otherwise. We denote by $\mathbb{T}^{\infty}_{\mathsf{rec}}$ the set of all recursive infinite trees.

The *subtree of* T *at* σ is the tree $T{\restriction_\sigma}$ defined by $T{\restriction_\sigma}(\tau) = T(\sigma \star \tau)$ for all $\tau \in \mathbb{N}^{<\omega}$.

A map $f : \mathbb{N} \to \mathbb{N}$ is *an infinite path of* T if $\langle f|n\rangle \in \mathsf{dom}(T)$ for all $n \in \mathbb{N}$.

We denote by $\Pi(T)$ the set of all infinite paths of T. By König's lemma, a tree T is infinite if and only if $\Pi(T) \neq \emptyset$.

The Lambda Calculus

We generally use the notation of Barendregt's book [1] for λ-calculus. The set Λ of λ-terms over an infinite set $\mathbb{V}$ of variables is defined by the following grammar:

$$\Lambda : \quad M, N ::= x \mid \lambda x.M \mid MN \qquad \text{for all } x \in \mathbb{V}.$$

We assume that application associates to the left, while λ-abstraction to the right. Application has a higher precedence than λ-abstraction. E.g., $\lambda xyz.xyz := \lambda x.(\lambda y.(\lambda z.((xy)z)))$.

The set $\mathsf{FV}(M)$ of *free variables* of M and the α-conversion are defined as in [1, Ch. 1§2]. A λ-term M is *closed* whenever $\mathsf{FV}(M) = \emptyset$ and in this case it is also called a *combinator.* The set of all combinators is denoted by Λ^o. Hereafter, we consider λ-terms up to α-conversion and we adopt the variable convention [1, Conv. 2.1.13]. We fix the following combinators:

$$\begin{array}{lll} \mathbf{I} := \lambda x.x & \Delta := \lambda x.xx & \mathbf{1}_n := \lambda x y_1 \dots y_n.x y_1 \cdots y_n \\ \mathbf{K} := \lambda xy.x & \Omega := \Delta\,\Delta & \mathbf{Y} := \lambda f.(\lambda x.f(xx))(\lambda x.f(xx)) \\ \mathbf{S}^+ := \lambda nfs.nf(fs) & \mathsf{c}_n := \lambda fz.f^n(z) & \mathbf{J} := \mathbf{Y}(\lambda zxy.x(zy)) \end{array}$$

where $f^n(z) := f(\cdots f(f(z))\cdots)$. We will simply denote by $\mathbf{1}$ the combinator $\mathbf{1}_1 := \lambda xy.xy$.

Given two λ-terms M, N we denote by $M\{N/x\}$ the capture-free simultaneous substitution of N for all free occurrences of x in M. The β- and η- reductions are defined by:

$$(\beta) \quad (\lambda x.M)N \to_\beta M\{N/x\} \qquad\qquad (\eta) \quad \lambda x.Mx \to_\eta M \text{ provided } x \notin \mathsf{FV}(M).$$

Given a reduction $\to_{\mathtt{R}}$, we write $\twoheadrightarrow_{\mathtt{R}}$ for its transitive-reflexive closure, we denote by $\mathsf{nf}_{\mathtt{R}}(M)$ the $\mathtt{R}$-normal form of M (if it exists) and by $\mathrm{NF}_{\mathtt{R}}$ the set of all $\mathtt{R}$-normal forms. We denote by $=_{\mathtt{R}}$ the corresponding $\mathtt{R}$-conversion.

A *context* $C[-]$ is a λ-term with a *hole* denoted by $[-]$. Given a *context* $C[-]$, we write $C[M]$ for the λ-term obtained from C by substituting M for the hole possibly with capture of free variables in M. A context $C[-]$ is called: a *head context* if it has the shape $(\lambda x_1 \dots x_k.[-])M_1 \cdots M_n$ for $k, n \geq 0$; *applicative* if it is of the form $[-]M_1 \cdots M_n$ for $n \geq 0$. A head (resp. applicative) context $C[-]$ is *closed* when all the M_i's are closed λ-terms.

▶ **Definition 2.** A λ-term M is *solvable* when there is a (head) context $C[-]$ such that $C[M] \twoheadrightarrow_\beta \mathbf{I}$. Otherwise M is called *unsolvable.*

Wadsworth proved in [26] that a λ-term M is solvable if and only if M has a *head normal form* (*hnf*), which means that $M \twoheadrightarrow_\beta \lambda x_1 \dots x_n.yN_1 \cdots N_k$ (for some $n, k \geq 0$).

The *principal head normal form* of a λ-term M, denoted $\mathrm{phnf}(M)$, is the head normal form obtained from M by *head reduction* [1, Def. 8.3.10].

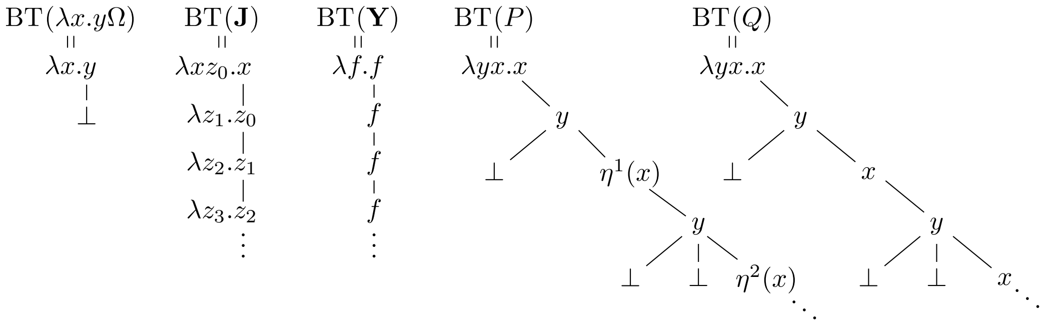


Figure 1 Some examples of Böhm trees. We refer to [1, Lemma 16.4.4] for the definition of P, Q.

Böhm trees

The *Böhm tree* $\mathrm{BT}(M)$ of a λ-term M is defined coinductively: if M is unsolvable then $\mathrm{BT}(M) = \bot$; if M is solvable and $\mathrm{phnf}(M) = \lambda x_1 \dots x_n.yN_1 \cdots N_k$ then:

$$\mathrm{BT}(M) = \begin{array}{c} \lambda x_1 \dots x_n.y \\ \diagup \qquad \diagdown \\ \mathrm{BT}(N_1) \quad \cdots \quad \mathrm{BT}(N_k) \end{array}$$

More generally, we say that T is a *Böhm tree* if there is a λ-term M such that $\mathrm{BT}(M) = T$. In Figure 1 we provide some examples of Böhm trees. Since every Böhm tree can be seen as a labelled tree over $\mathcal{L} = \{\bot\} \cup \{\lambda\vec{x}.y \mid \vec{x}, y \in \mathbb{V}\}$, we adopt the same notions and notations introduced for trees. However, we will write $\sigma \in \mathrm{BT}(M)$ rather than $\sigma \in \mathsf{dom}(\mathrm{BT}(M))$.

Given two Böhm trees T, T' we set $T \leq_\bot T'$ if and only if T results from T' by replacing some subtrees with $\bot$. When T is finite, we say that T *is a finite approximant of* T'.

The set $\mathrm{NF}_\bot$ of *finite approximants* is the set of normal λ-terms possibly containing $\bot$ inductively defined as follows: $\bot \in \mathrm{NF}_\bot$; if $t_i \in \mathrm{NF}_\bot$ for $i \in [1..n]$ then $\lambda\vec{x}.yt_1 \cdots t_n \in \mathrm{NF}_\bot$. The *size* of a finite approximant $t \in \mathrm{NF}_\bot$, written $\mathrm{size}(t)$, is defined as usual:

- $\mathrm{size}(\bot) = 0$ and $\mathrm{size}(\lambda x_1 \dots x_n.yt_1 \cdots t_k) = \mathrm{size}(t_1) + \cdots + \mathrm{size}(t_k) + n + 1$.

The set $\mathrm{BT}^k(M)$ of all finite approximants of $\mathrm{BT}(M)$ of size at most k is defined by

- $\mathrm{BT}^k(M) = \{t \in \mathrm{NF}_\bot \mid \mathrm{size}(t) \leq k, t \leq_\bot \mathrm{BT}(M)\}$.

The set $\mathrm{BT}^*(M) = \bigcup_{k\in\mathbb{N}} \mathrm{BT}^k(M)$ is therefore the set of all finite approximants of $\mathrm{BT}(M)$.

Observational Preorders and Lambda Theories

Observational preorders and λ-theories become the main object of study when considering the computational equivalence more important than the process of calculus.

A relation on Λ is *compatible* if it is compatible with application and λ-abstraction.

▶ **Definition 3.** A *preorder theory* is any compatible preorder on Λ containing the β-conversion. A λ*-theory* is any compatible equivalence on Λ containing the β-conversion.

Given a λ-theory (resp. preorder theory) $\mathcal{T}$ we write $M =_{\mathcal{T}} N$ ($M \sqsubseteq_{\mathcal{T}} N$) for $(M, N) \in \mathcal{T}$. The set of all λ-theories, ordered by inclusion, forms a (quite rich) complete lattice [17].

A λ-theory $\mathcal{T}$ is called: *consistent* if it does not equate all λ-terms; *extensional* if it contains the η-conversion; *sensible* if it equates all unsolvables.

We denote by $\boldsymbol{\lambda}$ the least λ-theory, by $\boldsymbol{\lambda}\eta$ the least extensional λ-theory, by $\mathcal{H}$ the least sensible λ-theory, and by $\mathcal{B}$ the (sensible) λ-theory equating all λ-terms having the same Böhm tree. Given a λ-theory $\mathcal{T}$, we write $\mathcal{T}\eta$ for the least λ-theory containing $\mathcal{T} \cup \boldsymbol{\lambda}\eta$.

Several interesting λ-theories are obtained via suitable observational preorders defined with respect to a set $\mathcal{O}$ of *observables*. Given $\mathcal{O} \subseteq \Lambda$, we write $M \in_{\mathtt{R}} \mathcal{O}$ if $M \twoheadrightarrow_{\mathtt{R}} M' \in \mathcal{O}$.

Given $\mathcal{O} \subseteq \Lambda$, the $\mathcal{O}$*-observational preorder* is given by:

$$M \sqsubseteq^{\mathcal{O}} N \iff \forall C[-] \,.\, C[M] \in_\beta \mathcal{O} \text{ entails } C[N] \in_\beta \mathcal{O}.$$

The induced *equivalence* $M \equiv^{\mathcal{O}} N$ is defined as $M \sqsubseteq^{\mathcal{O}} N$ and $N \sqsubseteq^{\mathcal{O}} M$.

▶ **Definition 4.** We focus on the following observational preorders and equivalences:

- Hyland's preorder $\sqsubseteq^{\mathrm{hnf}}$ and equivalence $\equiv^{\mathrm{hnf}}$ are obtained by taking as $\mathcal{O}$ the set of head normal forms. They are maximal among preorder theories and λ-theories, respectively.
- for *Morris's preorder* $\sqsubseteq^{\mathrm{nf}}$ and *equivalence* $\equiv^{\mathrm{nf}}$ we consider as $\mathcal{O}$ the set NF_β [21].

We denote by $\mathcal{H}^*$ and $\mathcal{H}^+$ the λ-theories corresponding to $\equiv^{\mathrm{hnf}}$ and $\equiv^{\mathrm{nf}}$, respectively.

The ω-Rule

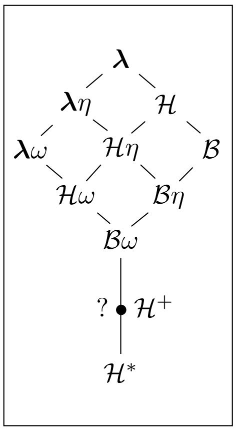
The ω-rule is a strong form of extensionality defined as follows:

$$(\omega) \qquad \forall P \in \Lambda^o . MP = NP \text{ entails } M = N.$$

We write (ω^0) for the ω-rule restricted to combinators $M, N \in \Lambda^o$. Given a λ-theory $\mathcal{T}$ we denote its closure under the ω-rule by $\mathcal{T}\omega$. We say that $\mathcal{T}$ *satisfies the* ω*-rule*, written $\mathcal{T} \vdash \omega$, if $\mathcal{T} = \mathcal{T}\omega$. By collecting some results in [1, §4.1] we have for all λ-theories $\mathcal{T}$:

- $\mathcal{T}\eta \subseteq \mathcal{T}\omega$,
- $\mathcal{T} \vdash \omega$ if and only if $\mathcal{T} \vdash \omega^0$,
- $\mathcal{T} \subseteq \mathcal{T}'$ entails $\mathcal{T}\omega \subseteq \mathcal{T}'\omega$.

The picture on the right, where $\mathcal{T}$ is above $\mathcal{T}'$ if $\mathcal{T} \subsetneq \mathcal{T}'$, is taken from [1, Thm. 17.4.16] and shows some facts about the λ-theories under consideration. In [1, §17.4], Sallé conjectured that $\mathcal{B}\omega \subsetneq \mathcal{H}^+$.



The counterexample showing that $\boldsymbol{\lambda}\eta \nvdash \omega$ is based on Plotkin's terms [1, Def. 17.3.26]. Since these terms are unsolvable, they become useless when considering sensible λ-theories. The techniques to analyze the ω-rule for sensible λ-theories are discussed in Section 3.2.

3 The Lambda Theories $\mathcal{H}^+$ and $\mathcal{H}^*$

In this section we recall the characterizations of $\mathcal{H}^+$ and $\mathcal{H}^*$ in terms of "extensional" Böhm-trees, and we discuss the ω-rule for sensible λ-theories in the interval $[\mathcal{B}, \mathcal{H}^*]$.

3.1 Böhm Trees and Their η-Expansions

Morris's theory $\mathcal{H}^+$ and preorder $\sqsubseteq^{\mathrm{nf}}$ correspond to the contextual theories where the observables are the β-normal forms. Notice that it is equivalent to take as observables the $\beta\eta$-normal forms since $M \twoheadrightarrow_\beta \mathsf{nf}_\beta(M)$ exactly when $M \twoheadrightarrow_{\beta\eta} \mathsf{nf}_{\beta\eta}(M)$, so $\boldsymbol{\lambda}\eta \subseteq \mathcal{H}^+$.

Moreover, by the Context Lemma for β-normalizable terms [7, Lemma 1.2], the context $C[-]$ separating two combinators $M \not\sqsubseteq^{\mathrm{nf}} N$ can be chosen applicative and closed.

▶ **Theorem 5** ([13, Thm. 2.6])**.** *Let $M, N \in \Lambda$. Then $M =_{\mathcal{H}^+} N$ if and only if* $\mathrm{BT}(M)$ *and* $\mathrm{BT}(N)$ *are equal up to denumerable many η-expansions of finite depth.*

This means that there exists a Böhm tree T such that both $\mathrm{BT}(M)$ and $\mathrm{BT}(N)$ can be transformed into T by performing a denumerable (possibly finite) number of η-expansions.

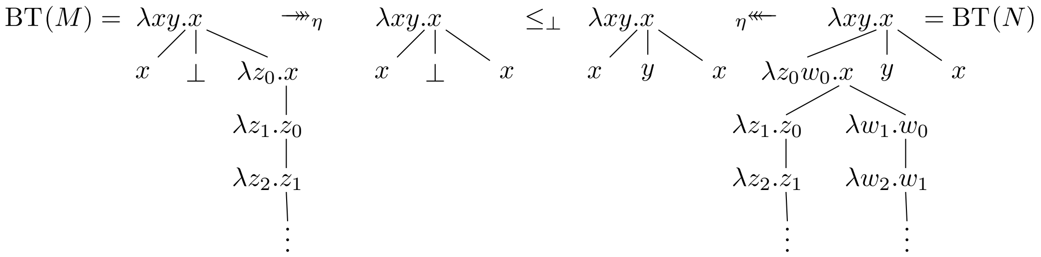


Figure 2 A situation witnessing the fact that $M \sqsubseteq^{\mathrm{hnf}} N$ holds.

Consider, for instance, the two λ-terms P, Q whose Böhm trees are depicted in Figure 1, where $\eta^n(x)$ denotes the η-expansion of x having depth n. (Such terms exist by [1, §16.4].) Now, $\mathrm{BT}(P)$ is such that at every level $2n$ the variable x is η-expanded n times (in depth). We have $P =_{\mathcal{H}^+} Q$ because we can perform infinitely many finite η-expansions of increasing depth in $\mathrm{BT}(Q)$ and obtain $\mathrm{BT}(P)$ (but in general we may need to η-expand both trees).

As a brief digression, notice that the existence of such P, Q entails that $\mathcal{B}\eta \subsetneq \mathcal{H}^+$. Indeed, for $M, N \in \Lambda$, $M \to_\eta N$ entails that $\mathrm{BT}(M)$ can be obtained from $\mathrm{BT}(N)$ by performing at most *one* η-expansion at every level (see [1, Lemma 16.4.3]). It is easy to show that $P \neq_{\mathcal{B}\eta} Q$.

Hyland's theory $\mathcal{H}^*$ and preorder $\sqsubseteq^{\mathrm{hnf}}$ correspond to the contextual theories where the observables are the head normal forms (equivalently, the solvable λ-terms). It is easy to show that $M \sqsubseteq^{\mathrm{nf}} N$ entails $M \sqsubseteq^{\mathrm{hnf}} N$, so $\mathcal{H}^+ \subsetneq \mathcal{H}^*$. Terms M, N such that $M \not\sqsubseteq^{\mathrm{hnf}} N$ are called *semi-separable* in [1]. Also in this case, the context lemma for solvable terms [26, Lemma 6.1] entails that the semi-separating contexts can be chosen applicative and closed.

The characterization of $\mathcal{H}^*$ in terms of trees, needs the notion of *infinite η-expansion* of a Böhm tree. The classical definition is given in terms of tree extensions in [1, Def. 10.2.10].

▶ **Definition 6.** Given two Böhm trees T and T', we say that T is a (possibly) *infinite η-expansion* of T', written $T \twoheadrightarrow_\eta T'$, if T is obtained from T' by performing denumerable many η-expansions of possibly infinite depth.

We prefer not to use Barendregt's classical notation $T \geq_\eta T'$ as it could be confusing. Our notation is borrowed from infinite rewriting since $\twoheadrightarrow_\eta$ can also be defined in this way [25]. (We refer here to the strongly converging η-reduction of [25] restricted to Böhm trees.)

▶ **Theorem 7** ([1, Thm. 19.2.9]). *Let $M, N \in \Lambda$.*

- *$M \sqsubseteq^{\mathrm{hnf}} N$ iff there are Böhm trees T, T' such that $\mathrm{BT}(M)\ {}_\eta\!\twoheadleftarrow T \leq_\perp T' \twoheadrightarrow_\eta \mathrm{BT}(N)$.*
- *$M =_{\mathcal{H}^*} N$ iff $\mathrm{BT}(M)$ and $\mathrm{BT}(N)$ are equal up to denumerable many (possibly) infinite η-expansions; this means that there is a Böhm tree T such that $\mathrm{BT}(M)\ {}_\eta\!\twoheadleftarrow T \twoheadrightarrow_\eta \mathrm{BT}(N)$.*

The typical example is $\mathbf{I} \sqsubseteq^{\mathrm{hnf}} \mathbf{J}$, since clearly $\mathrm{BT}(\mathbf{J}) \twoheadrightarrow_\eta \mathbf{I}$ (on the contrary, $\mathbf{I} \not\sqsubseteq^{\mathrm{nf}} \mathbf{J}$), but in general to show that $M \sqsubseteq^{\mathrm{hnf}} N$ one may need infinitely η-expand also $\mathrm{BT}(M)$ and cut some subtree of $\mathrm{BT}(N)$. This is the case for $M = \lambda xy.xx\Omega(\mathbf{J}x)$ and $N = \lambda xy.x(\mathbf{J}x)yx$, see Fig. 2. We recall from [14, Thm. 5.4(a)] that the relation $\sqsubseteq^{\mathrm{hnf}}$ can be stratified as follows.

▶ **Lemma 8.** *For $M, N \in \Lambda$, $M \sqsubseteq^{\mathrm{hnf}}_k N$ iff either $k = 0$, or M is unsolvable, or $k > 0$ and*

$$M =_\beta \lambda x_1 \ldots x_{n_1}.yM_1 \cdots M_{m_1} \text{ and } N =_\beta \lambda x_1 \ldots x_{n_2}.yN_1 \cdots N_{m_2}$$

where $n_1 - m_1 = n_2 - m_2$ and either y is free or $y = x_j$ for some $j \leq \min\{n_1, n_2\}$. So if, say, $m_1 \leq m_2$ then $n_1 \leq n_2$ and there exists $p \geq 0$ such that $n_2 = n_1 + p$ and $m_2 = m_1 + p$.

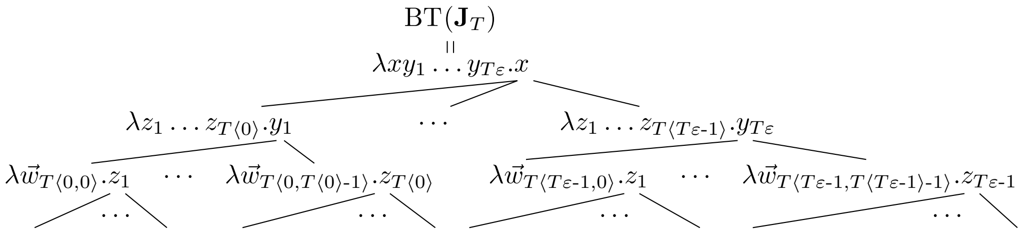

$\mathrm{BT}(\mathbf{J}_T)$
$\lambda x y_1 \dots y_{T\varepsilon}.x$
$\lambda z_1 \dots z_{T\langle 0\rangle}.y_1$
$\cdots$
$\lambda z_1 \dots z_{T\langle T\varepsilon\text{-}1\rangle}.y_{T\varepsilon}$
$\lambda \vec{w}_{T\langle 0,0\rangle}.z_1$
$\cdots$
$\lambda \vec{w}_{T\langle 0,T\langle 0\rangle\text{-}1\rangle}.z_{T\langle 0\rangle}$
$\lambda \vec{w}_{T\langle T\varepsilon\text{-}1,0\rangle}.z_1$
$\cdots$
$\lambda \vec{w}_{T\langle T\varepsilon\text{-}1,T\langle T\varepsilon\text{-}1\rangle\text{-}1\rangle}.z_{T\varepsilon\text{-}1}$

Figure 3 The Böhm tree of $\mathbf{J}_T$, an infinite η-expansion of $\mathbf{I}$ following $T \in \mathbb{T}^{\infty}_{\mathsf{rec}}$. To lighten the notations we write $T\sigma$ rather than $T(\sigma)$ and we let $\vec{w}_n := w_1, \dots, w_n$.

Moreover $M_i \sqsubseteq^{\mathrm{hnf}}_{k-1} N_i$ for all $i \leq m_1$ and $x_{n_1+j} \sqsubseteq^{\mathrm{hnf}}_{k-1} N_{m_1+j}$ for all $j \leq p$. (The case $m_1 > m_2$ is symmetrical.) Finally we have $M \sqsubseteq^{\mathrm{hnf}} N$ iff $M \sqsubseteq^{\mathrm{hnf}}_k N$ for all $k \in \mathbb{N}$.

The Infinite η-Expansion $\mathbf{J}_T$

Note that $\mathbf{J}$ is not the unique infinite η-expansion of the identity. For each $T \in \mathbb{T}^{\infty}_{\mathsf{rec}}$ there is a λ-term $\mathbf{J}_T$ which is an infinite η-expansion of $\mathbf{I}$ *following* T in the sense of Figure 3.

We could just say that the existence of such a $\mathbf{J}_T$ follows directly from the fact that T is recursive and [1, Thm. 10.1.23]. We prefer to give a more explicit definition.

Let us fix a bijective encoding $\# : \mathbb{N}^* \to \mathbb{N}$ and let $\mathtt{Cons} \in \Lambda^0$ be such that

$$\mathtt{Cons}\,\mathsf{c}_{\#\sigma}\,\mathsf{c}_n =_\beta \mathsf{c}_{\#(\sigma.n)}$$

for all $\sigma \in \mathbb{N}^*, n \in \mathbb{N}$. Notice that such a combinator $\mathtt{Cons}$ exists by Church's thesis.

Given $M, N \in \Lambda$ and $x \notin \mathsf{FV}(M)$, we set:

$$[M, N] := \lambda x.xMN, \qquad\qquad M \circ N := \lambda x.M(Nx).$$

We associate with every tree T a partial map $f_T : \mathbb{N} \rightharpoonup \mathbb{N}$ such $n \in \mathsf{dom}(f_T)$ iff $n = \#\sigma$ for $\sigma \in \mathsf{dom}(T)$, and in this case $f_T(n) = T(\sigma)$. When T is recursive f_T is clearly λ-definable.

▶ **Definition 9.** Let $F \in \Lambda^o$ be the term λ-defining f_T. Define (for $X \in \Lambda$ arbitrary):

1. $L^X p := \lambda z.p(\lambda nxy.z(\mathbf{S}^+ n)(x(Xny)))$;
2. $M^X nx := nL^X[\mathsf{c}_0, x](\mathbf{KI})$;
3. $Ns := M^{N\circ(\mathtt{Cons}\,s)}(Fs)$, using the fixed point combinator $\mathbf{Y}$;
4. $\mathbf{J}_T := N\mathsf{c}_{\#\varepsilon}$.

▶ **Lemma 10.** *$\mathbf{J}_T$ is such that the underlying tree of $\mathrm{BT}(\mathbf{J}_T)$ is T and $\mathrm{BT}(\mathbf{J}_T) \twoheadrightarrow_\eta \mathbf{I}$.*

Proof sketch. First one verifies that for all for $X \in \Lambda$ and $n \in \mathbb{N}$ the following hold:

1. $(L^X)^n[\mathsf{c}_0, x] =_\beta \lambda z y_1 \dots y_n.z\mathsf{c}_n(x(X\mathsf{c}_0 y_1)\cdots(X\mathsf{c}_{n-1}y_n))$;
2. $M^X\mathsf{c}_n x =_\beta \lambda y_1 \dots y_n.x(X\mathsf{c}_0 y_1)\cdots(X\mathsf{c}_{n-1}y_n)$;
3. $N\mathsf{c}_{\#\sigma}x =_\beta \lambda y_1 \dots y_n.x(N\mathsf{c}_{\#(\sigma.0)}y_1)\cdots(N\mathsf{c}_{\#(\sigma.(n-1))}y_n)$, where $n = T(\sigma)$;

In particular, $\mathrm{BT}(\mathbf{J}_T)$ is $\perp$-free. From this, and the fact that $\mathbf{I}$ is $\beta\eta$-normal, we have that $\mathrm{BT}(\mathbf{J}_T) \twoheadrightarrow_\eta \mathbf{I}$ if and only if $\mathbf{J}_T \sqsubseteq^{\mathrm{hnf}}_k \mathbf{I}$ for all $k \in \mathbb{N}$ (by Lemma 8).

We prove by induction on $k - \ell(\sigma)$, that for all $k \in \mathbb{N}$ and $\sigma \in \mathsf{dom}(T)$ such that $\ell(\sigma) < k$, we have $N\mathsf{c}_{\#\sigma}x \sqsubseteq^{\mathrm{hnf}}_{k-\ell(\sigma)} x$. If $k - \ell(\sigma) = 0$ or $T(\sigma) = 0$ then it is trivial. Otherwise, it follows from (3) and the induction hypothesis since $\sigma.i \in \mathsf{dom}(T)$ for all $i < T(\sigma)$ and $k - \ell(\sigma.i) < k - \ell(\sigma)$. Finally, we conclude since $\mathbf{J}_T := N\mathsf{c}_{\#\varepsilon}$. ◀

▶ **Lemma 11.** *For all $T \in \mathbb{T}^{\infty}_{\mathsf{rec}}$, we have $\mathbf{J}_T \mathbf{1}_{T(\varepsilon)} =_{\mathcal{B}} \mathbf{J}_T$.*

In Section 4 we consider $M \sqsubseteq^{\mathrm{hnf}} N$ for some M, N such that M is β-normal and $\mathrm{BT}(N)$ is infinite. We show that M and $\mathrm{BT}(N)$ have similar structure, except for the fact that there is a position σ in $\mathrm{BT}(N)$ where an infinite η-expansion following some $T \in \mathbb{T}^{\infty}_{\mathsf{rec}}$ occurs. Moreover, we show that this difference can be extracted via a suitable Böhm-out technique.

3.2 The ω-Rule for Sensible Theories

The validity of the ω-rule for λ-theories $\mathcal{T}$ containing $\mathcal{B}$ can be tricky to prove (or disprove). We have seen that the terms P, Q of Figure 1 are equated in $\mathcal{H}^+$, but different in $\mathcal{B}\eta$. Perhaps surprisingly, they can also be used to prove that $\mathcal{B}\eta \subsetneq \mathcal{B}\omega$ since $P =_{\mathcal{B}\omega} Q$ holds. The following argument is due to Barendregt, see [1, Lemma 16.4.4].

Recall the following basic fact: for every $M \in \Lambda^o$, there exists $k \geq 0$ such that $M\Omega \cdots \Omega$, k times, becomes unsolvable (see [1, Lemma 17.4.4]). By inspecting Figure 1, we notice that in $\mathrm{BT}(P)$ the variable y is applied to an increasing number of Ω's (represented by $\bot$). So, when substituting some $M \in \Lambda^o$ for y in $\mathrm{BT}(Py)$, there will be a level k of the tree where $M\Omega \cdots \Omega$ become $\bot$, thus cutting $\mathrm{BT}(PM)$ at level k. The same reasoning can be done for $\mathrm{BT}(QM)$. Therefore $\mathrm{BT}(PM)$ and $\mathrm{BT}(QM)$ only differ because of finitely many η-expansions. Since $\mathcal{B}\eta \subseteq \mathcal{B}\omega$, we conclude that $P =_{\mathcal{B}\omega} Q$.

The fact that $\mathcal{H}^* \vdash \omega$ is clearly a consequence of its maximality. However, there are several direct proofs: see [1, §17.2] for a syntactic demonstration and [26] for a semantic one. The longstanding open question whether $\mathcal{H}^+ \vdash \omega$ will be answered positively in Theorem 40. We believe that the λ-terms P and Q, suitably modified to get rid of the Ω's in their Böhm trees, are good candidates to show that $\mathcal{B}\omega \subsetneq \mathcal{H}^+$, but the question remains open.

4 Böhming-out Infinite η-Expansions

The Böhm out technique [3, 1, 23] aims to build a context which extracts (an instance of) the subterm of a λ-term M at position σ. It is used for separating two λ-terms M, N provided that their structure is sufficiently different (depending on the notion of *separation* under consideration). We show that when $M \not\sqsubseteq^{\mathrm{nf}} N$ this difference can be Böhmed out via an appropriate head context, even when M and N have a similar structure (i.e. $M \sqsubseteq^{\mathrm{hnf}} N$).

4.1 Morris Separators

We start by providing a notion of *Morris separator*, that is a sequence σ witnessing the fact that $M \not\sqsubseteq^{\mathrm{nf}} N$ for λ-terms M and N such that M is β-normal and $M \sqsubseteq^{\mathrm{hnf}} N$ holds.

We recall from Section 2 that we use for Böhm trees, the same notions and notations introduced for trees. However, we write $\sigma \in \mathrm{BT}(M)$ to indicate that $\sigma \in \mathsf{dom}(\mathrm{BT}(M))$.

Given $\sigma \in \mathrm{BT}(M)$ we define the *subterm M_σ of M at σ (relative to its Böhm tree)* by:

- $M_\varepsilon = M$,
- $M_{i.\sigma} = (M_{i+1})_\sigma$ whenever $\mathrm{phnf}(M) = \lambda\vec{x}.yM_1 \cdots M_n$.

▶ **Definition 12.** We say that $\sigma \in \mathrm{BT}(M) \cap \mathrm{BT}(N)$ is a *Morris separator for M, N*, written $\sigma : M \not\sqsubseteq^{\mathrm{nf}} N$, if there exists $i > 0$ such that, for some $p \geq i$, we have:

$$M_\sigma =_\beta \lambda x_1 \ldots x_n.yM_1 \cdots M_m \quad \text{and} \quad N_\sigma =_\beta \lambda x_1 \ldots x_{n+p}.yN_1 \cdots N_{m+p}$$

where $N_{m+i} =_{\mathcal{B}} \mathbf{J}_T x_{n+i}$ for some $T \in \mathbb{T}^{\infty}_{\mathsf{rec}}$.

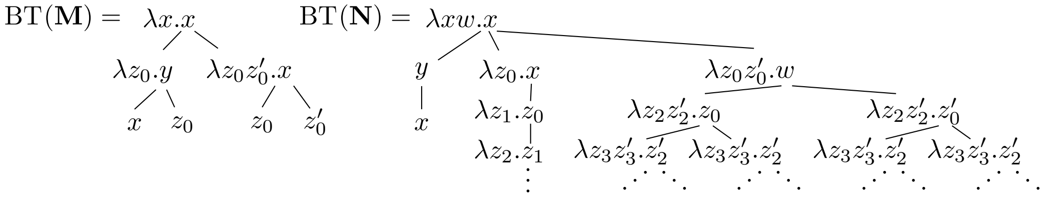


Figure 4 Two terms $\mathbf{M}, \mathbf{N}$ such that $\mathbf{M}$ is β-normal, $\mathbf{M} \sqsubseteq^{\mathrm{hnf}} \mathbf{N}$, but $\mathbf{M} \not\sqsubseteq^{\mathrm{nf}} \mathbf{N}$.

It is easy to check that $\sigma : M \not\sqsubseteq^{\mathrm{nf}} N$ and $\sigma = \langle k \rangle \star \tau$ entail $\tau : M_{k+1} \not\sqsubseteq^{\mathrm{nf}} N_{k+1}$.

Recall that we are considering λ-terms M, N such that M is β-normal, $M \not\sqsubseteq^{\mathrm{nf}} N$, but $M \sqsubseteq^{\mathrm{hnf}} N$. Obviously such M and N are not (semi-)separable, using the terminology of [1]. Since M is β-normal, its Böhm tree is finite and $\bot$-free. Since $M \sqsubseteq^{\mathrm{hnf}} N$, by Lemma 8 also $\mathrm{BT}(N)$ is $\bot$-free; moreover, at every position $\sigma \in \mathrm{BT}(M) \cap \mathrm{BT}(N)$, M_σ and N_σ have similar hnfs (the number of abstractions and applications can be matched via η-expansions). Note that $\mathrm{BT}(M)$ might have η-expansions that are not present in $\mathrm{BT}(N)$. As $M \not\sqsubseteq^{\mathrm{nf}} N$, the Böhm tree of N must have infinite subtrees of the form $\mathrm{BT}(\mathbf{J}_T x)$ for some $x \in \mathbb{V}, T \in \mathbb{T}^{\infty}_{\mathsf{rec}}$.

To explain the idea behind Morris separators, we use the terms $\mathbf{M}$ and $\mathbf{N}$ whose Böhm trees are depicted in Figure 4. This example admits two Morris separators: ε and $\langle 1, 0 \rangle$.

The empty sequence ε is a separator since $\mathbf{N}_{\langle 2 \rangle} =_{\mathcal{B}} \mathbf{J}_{T_2} w$ where T_2 is the complete binary tree. The sequence $\langle 1, 0 \rangle$ is a separator because $\mathbf{N}_{\langle 1,0,0 \rangle} =_{\mathcal{B}} \mathbf{J}_{T_1} z_1$ where T_1 is the complete unary tree (i.e. $\mathbf{J}_{T_1} =_{\mathcal{B}} \mathbf{J}$).

▶ **Proposition 13.** *Let $M, N \in \Lambda$ such that M is β-normal, $N \notin_\beta \mathrm{NF}_\beta$ and $M \sqsubseteq^{\mathrm{hnf}} N$. Then there exists a position $\sigma \in \mathrm{BT}(M) \cap \mathrm{BT}(N)$ such that $\sigma : M \not\sqsubseteq^{\mathrm{nf}} N$.*

Proof. Since M is β-normal, N does not have a β-normal form and $M \sqsubseteq^{\mathrm{hnf}} N$, $\mathrm{BT}(N)$ must be infinite and $\bot$-free. By König's lemma there is $f \in \Pi(\mathrm{BT}(N))$ and since $\mathrm{BT}(M)$ is finite there exists $n > 0$ such that $\sigma := \langle f | n-1 \rangle \in \mathrm{BT}(M) \cap \mathrm{BT}(N)$ but $\langle f | n \rangle \notin \mathrm{BT}(M)$. By applying Lemma 8, there exists $p > 0$ such that $M_\sigma =_\beta \lambda x_1 \dots x_{n_1}.y M_1 \cdots M_{m_1}$ and $N_\sigma =_\beta \lambda x_1 \dots x_{n_1+p}.y N_1 \cdots N_{m_1+p}$. Moreover, $x_{n_1+j} \sqsubseteq^{\mathrm{hnf}} N_{m_1+j}$ where $j = f(n) + 1 - m$. Since $\mathrm{BT}(N_{m_1+j})$ is infinite we must have $N_{m_1+j} =_{\mathcal{B}} \mathbf{J}_T x_{n_1+j}$ for some $T \in \mathbb{T}^{\infty}_{\mathsf{rec}}$. ◀

4.2 A Böhm-out Technique for Separating the Inseparable

The following combinators will be used (among others) to build the Böhm-out context:

$$\mathbf{U}^n_k := \lambda x_1 \dots x_n.x_k \qquad\qquad \mathbf{P}_n := \lambda x_1 \dots x_n.\lambda z.z x_1 \cdots x_n$$

The combinator $\mathbf{U}^n_k$ is called *projector* and $\mathbf{P}_n$ *tupler* as they enjoy the following properties.

▶ **Lemma 14.** *Let $k \geq n \geq 0$ and $X_1, \dots, X_n, Y_1, \dots, Y_{k-n} \in \Lambda^o$.*

1. $(\mathbf{P}_k X_1 \cdots X_n) Y_1 \cdots Y_{k-n} =_\beta \lambda z.z X_1 \cdots X_n Y_1 \cdots Y_{k-n}$
2. $(\lambda z.z X_1 \cdots X_n) \mathbf{U}^n_i =_\beta X_i$

When $\mathbf{U}^n_k$ is substituted for y in $\lambda \vec{x}.y M_1 \cdots M_n$, it extracts an instance of M_k. Let us consider the λ-term $\mathbf{N}$ whose Böhm tree is given in Figure 4. The context $[-]\mathbf{U}^2_1$ extracts from $\mathbf{N}$ the subterm yx where x is replaced by $\mathbf{U}^2_1$. The idea of the Böhm-out technique is to replace every variable along the path σ with the correct projector.

The issue is when the same variable occurs several times in σ and we must select different children in these occurrences. For example, to extract $\mathbf{N}_{\langle 1,0 \rangle}$, the first occurrence of x should be replaced by $\mathbf{U}_2^3$, the second by $\mathbf{U}_1^1 := \mathbf{I}$. The problem was originally solved by Böhm in [3] by first replacing the occurrences of the same variables along the path by different variables using the tupler, and then replacing each variable by the suitable projector. In the example under consideration, the context $[-]\mathbf{P}_3\Omega\mathbf{U}_2^3\mathbf{U}_1^1\Omega\Omega\mathbf{U}_1^3$ extracts from $\mathbf{N}$ the instance of $\mathbf{N}_{\langle 1,0\rangle}$ where z_0 is replaced by $\mathbf{I}$.

Obviously, finite η-differences can be destroyed during the process of Böhming out. In contrast, we show that infinite η-differences can always be preserved.

▶ **Lemma 15** (Böhm-out). *Let $M, N \in \Lambda$ such that $M \sqsubseteq^{\mathrm{hnf}} N$, let $\vec{y} = \mathsf{FV}(MN)$ and $\sigma : M \not\sqsubseteq^{\mathrm{nf}} N$. Then for all $k \in \mathbb{N}$ large enough, there are combinators $\vec{X} \in \Lambda^o$ such that $M\{\mathbf{P}_k/\vec{y}\}\vec{X} =_\beta \mathbf{1}_{T(\varepsilon)}$ and $N\{\mathbf{P}_k/\vec{y}\}\vec{X} =_{\mathcal{B}} \mathbf{J}_T$ for some $T \in \mathbb{T}^{\infty}_{\mathsf{rec}}$.*

Proof. We proceed by induction on σ.

Base case: $\boldsymbol{\sigma = \varepsilon}$. Then there exists $i > 0$ such that, for some $p \geq i$, we have:

$$M =_\beta \lambda x_1 \ldots x_n.yM_1 \cdots M_m \text{ and } N =_\beta \lambda x_1 \ldots x_{n+p}.yN_1 \cdots N_{m+p}$$

where $N_{m+i} =_{\mathcal{B}} \mathbf{J}_T x_{n+i}$. For any $k \geq n+m+p$ let us set $\vec{X} := \mathbf{P}_k^{\sim n}\mathbf{1}_{T(\varepsilon)}^{\sim p}\Omega^{\sim k-m-p}\mathbf{U}_{m+i}^k$, where $M^{\sim n}$ denotes the sequence of λ-terms containing n copies of M.

We split into cases depending on whether y is free or $y = x_j$ for some $j \leq n$. We consider the former case, as the latter is analogous. On the one side we have:

$$\begin{array}{ll}
(\lambda x_1 \ldots x_n.yM_1 \cdots M_m)\{\mathbf{P}_k/\vec{y}\}\vec{X} = & \\
(\lambda x_1 \ldots x_n.\mathbf{P}_k M_1' \cdots M_m')\vec{X} =_\beta & \text{where } M_\ell' := M_\ell\{\mathbf{P}_k/\vec{y}\} \\
(\mathbf{P}_k M_1'' \cdots M_m'')\mathbf{1}_{T(\varepsilon)}^{\sim p}\Omega^{\sim k-m-p}\mathbf{U}_{m+i}^k =_\beta & \text{where } M_\ell'' := M_\ell'\{\mathbf{P}_k/\vec{x}\} \\
(\lambda z.zM_1'' \cdots M_m''\mathbf{1}_{T(\varepsilon)}^{\sim p}\Omega^{\sim k-m-p})\mathbf{U}_{m+i}^k =_\beta \mathbf{1}_{T(\varepsilon)} & \text{by Lemma 14.1 and 14.2.}
\end{array}$$

On the other side, we have:

$$\begin{array}{ll}
(\lambda x_1 \ldots x_{n+p}.x_j N_1 \cdots N_{m+p})\{\mathbf{P}_k/\vec{y}\}\vec{X} = & \\
(\lambda x_1 \ldots x_{n+p}.\mathbf{P}_k N_1' \cdots N_{m+p}')\vec{X} =_\beta & \text{for } N_\ell' := N_\ell\{\mathbf{P}_k/\vec{y}\} \\
(\lambda x_{n+1} \ldots x_{n+p}.\mathbf{P}_k N_1'' \cdots N_{m+p}'')\mathbf{1}_{T(\varepsilon)}^{\sim p}\Omega^{\sim k-m-p}\mathbf{U}_{m+i}^k = & \text{for } N_\ell'' := N_\ell'\{\mathbf{P}_k/x_1, \ldots, x_n\} \\
(\mathbf{P}_k N_1''^{*} \cdots N_{m+p}''^{*})\Omega^{\sim k-m-p}\mathbf{U}_{m+i}^k = & \text{for } N_\ell''^{*} := N_\ell''\{\mathbf{1}_{T(\varepsilon)}/x_{n+1}, \ldots, x_{n+p}\} \\
(\lambda z.zN_1''^{*} \cdots N_{m+p}''^{*}\Omega^{\sim k-m-p})\mathbf{U}_{m+i}^k = & \text{by Lemma 14.1} \\
N_{m+i}''^{*} = (\mathbf{J}_T x_{n+i})\{\mathbf{1}_{T(\varepsilon)}/x_{n+i}\} = \mathbf{J}_T\mathbf{1}_{T(\varepsilon)} = \mathbf{J}_T & \text{by Lemma 14.2 and 11}
\end{array}$$

Induction case: $\boldsymbol{\sigma = \langle i\rangle \star \sigma'}$. By Lemma 8, for $n - m = n' - m'$ and $i + 1 \leq \min\{m, m'\}$ we have:

$$M = \lambda x_1 \ldots x_n.yM_1 \cdots M_m \qquad N = \lambda x_1 \ldots x_{n'}.yN_1 \cdots N_{m'}$$

where $M_j \sqsubseteq^{\mathrm{hnf}} N_j$ for all $j \leq \min\{m, m'\}$ and either y is free or $y = x_j$ for $j \leq \min\{n, n'\}$. Suppose that, say, $n \leq n'$. Then there is $p \geq 0$ such that $n' = n + p$ and $m' = m + p$. Since $M_{i+1} \sqsubseteq^{\mathrm{hnf}} N_{i+1}$ and $\sigma' : M_{i+1} \not\sqsubseteq^{\mathrm{nf}} N_{i+1}$ we apply the induction hypothesis and get, for any k' large enough, $\vec{Y} \in \Lambda^o$ such that $M_{i+1}\{\mathbf{P}_{k'}/\vec{y}, \vec{x}\}\vec{Y} =_\beta \mathbf{1}_{T(\varepsilon)}$ and $N_{i+1}\{P_{k'}/\vec{y}, \vec{x}\}\vec{Y} =_{\mathcal{B}} \mathbf{J}_T$. For any $k \geq \max\{k', n + m + p\}$, we set $\vec{X} := \mathbf{P}_k^{\sim n+p}\Omega^{\sim k-m-p}\mathbf{U}_{i+1}^k\vec{Y}$.
We suppose that y is free, the other case being analogous. On the one side we have:

$$\begin{array}{ll}
(\lambda x_1 \ldots x_n.yM_1 \cdots M_m)\{\mathbf{P}_k/\vec{y}\}\vec{X} = & \\
(\lambda x_1 \ldots x_n.\mathbf{P}_k M_1' \cdots M_m')\mathbf{P}_k^{\sim n+p}\Omega^{\sim k-m-p}\mathbf{U}_{i+1}^k\vec{Y} =_\beta & \text{where } M_\ell' := M_\ell\{\mathbf{P}_k/\vec{y}\} \\
(\mathbf{P}_k M_1'' \cdots M_m'')\mathbf{P}_k^{\sim p}\Omega^{\sim k-m-p}\mathbf{U}_{i+1}^k\vec{Y} =_\beta & \text{where } M_\ell'' := M_\ell'\{\mathbf{P}_k/\vec{x}\} \\
(\lambda z.zM_1'' \cdots M_m''\mathbf{P}_k^{\sim p}\Omega^{\sim k-m-p})\mathbf{U}_{i+1}^k\vec{Y} =_\beta & \text{by Lemma 14.1} \\
M_{i+1}''\vec{Y} = M_{i+1}\{\mathbf{P}_k/\vec{y}, \vec{x}\}\vec{Y} =_\beta \mathbf{1}_{T(\varepsilon)} & \text{by Lemma 14.2 and IH}
\end{array}$$

On the other side, we have:

$$\begin{array}{ll}
(\lambda x_1 \dots x_{n+p}.y N_1 \cdots N_{m+p})\{\mathbf{P}_k/\vec{y}\}\vec{X} = & \\
(\lambda x_1 \dots x_{n+p}.\mathbf{P}_k N'_1 \cdots N'_{m+p})\vec{X} =_\beta & \text{where } N'_\ell := N_\ell\{\mathbf{P}_k/\vec{y}\} \\
(\mathbf{P}_k N''_1 \cdots N''_{m+p})\Omega^{\sim k-m-p}\mathbf{U}^k_{i+1}\vec{Y} =_\beta & \text{where } N''_\ell := N'_\ell\{\mathbf{P}_k/\vec{x}\} \\
(\lambda z.z N''_1 \cdots N''_{m+p}\Omega^{\sim k-m-p})\mathbf{U}^k_{i+1}\vec{Y} =_\beta & \text{by Lemma 14.1} \\
N''_{i+1}\vec{Y} = N_{i+1}\{\mathbf{P}_k/\vec{y},\vec{x}\}\vec{Y} =_\mathcal{B} \mathbf{J}_T & \text{by Lemma 14.2 and IH}
\end{array}$$

◀

From Proposition 13 we get this immediate corollary of Lemma 15.

▶ **Corollary 16.** *Let $M, N \in \Lambda$ such that M is β-normal, $N \notin_\beta \mathrm{NF}_\beta$ and $M \sqsubseteq^{\mathrm{hnf}} N$. Then there is a head context $C[-]$ such that $C[M] =_{\beta\eta} \mathbf{I}$ and $C[N] =_\mathcal{B} \mathbf{J}_T$ for some $T \in \mathbb{T}^\infty_{\mathsf{rec}}$.*

▶ **Theorem 17** (Morris Separation)**.** *Let $M, N \in \Lambda$ such that $M \sqsubseteq^{\mathrm{hnf}} N$ while $M \not\sqsubseteq^{\mathrm{nf}} N$. There is a head context $C[-]$ such that $C[M] =_{\beta\eta} \mathbf{I}$ and $C[N] =_\mathcal{B} \mathbf{J}_T$ for some $T \in \mathbb{T}^\infty_{\mathsf{rec}}$. When $M, N \in \Lambda^o$ the context $C[-]$ can be chosen closed and applicative.*

Proof. Since $M \not\sqsubseteq^{\mathrm{nf}} N$, there is a head context $D_2[-]$ such that $D_2[M] \in_\beta \mathrm{NF}_\beta$, while $D_2[N] \notin_\beta \mathrm{NF}_\beta$. From $M \sqsubseteq^{\mathrm{hnf}} N$ we obtain $D_2[M] \sqsubseteq^{\mathrm{hnf}} D_2[N]$. Therefore we can apply Corollary 16, and get a head context $D_1[-]$ such that $D_1[D_2[M]] =_{\beta\eta} \mathbf{I}$ and $D_1[D_2[N]] =_\mathcal{B} \mathbf{J}_T$ for some $T \in \mathbb{T}^\infty_{\mathsf{rec}}$. Hence the head context $C[-]$ we are looking for is actually $D_1[D_2[-]]$. When M, N are closed, all the contexts can be chosen closed and applicative. ◀

5 Relational Graph Models

In this section we recall the definition of *relational graph models* (rgm, for short). Individual examples of such models were previously studied in the literature (e.g., in [15]), but the class of rgms was formally introduced in [20]. We refer the reader to [22] for a detailed analysis.

5.1 The Relational Semantics

Relational graph models are called *relational* since they are reflexive objects in the cartesian closed category **MRel** [5], which is the Kleisli category of **Rel** with respect to the finite multisets comonad $\mathcal{M}_\mathrm{f}(-)$. Before going further we briefly recall the category **MRel**, but we refer to [5] for a detailed presentation.

Given a set A, a *multiset* over A is any map $a : A \to \mathbb{N}$. Given $\alpha \in A$ and a multiset a over A, the *multiplicity of α in a* is given by $a(\alpha)$. A multiset a is called *finite* if its *support* $\mathrm{supp}(a) = \{\alpha \in A \mid a(\alpha) \neq 0\}$ is finite. A finite multiset a is represented by the unordered list of its elements $[\alpha_1, \dots, \alpha_n]$, possibly with repetitions, and the empty multiset is denoted by $[]$. We write $\mathcal{M}_\mathrm{f}(A)$ for the set of all finite multisets over A. Given $a_1, a_2 \in \mathcal{M}_\mathrm{f}(A)$, their *multiset union* is denoted by $a_1 + a_2$ and defined as a pointwise sum.

The objects of **MRel** are all the sets. A morphism $f \in \mathbf{MRel}(A, B)$ is any relation between $\mathcal{M}_\mathrm{f}(A)$ and B, in other words $\mathbf{MRel}(A, B) = \mathcal{P}(\mathcal{M}_\mathrm{f}(A) \times B)$. The composition of $f \in \mathbf{MRel}(A, B)$ and $g \in \mathbf{MRel}(B, C)$ is defined as follows:

$$f\,;g = \{(a_1 + \cdots + a_k, \gamma) \mid \exists \beta_1, \dots, \beta_k, \text{ such that } (a_i, \beta_i) \in f,\ ([\beta_1, \dots, \beta_k], \gamma) \in g\}.$$

The identity of A is the relation $\mathrm{Id}_A = \{([\alpha], \alpha) \mid \alpha \in A\}$. It is easy to check that the product is the disjoint union $A \uplus B$ and the exponential object $A \Rightarrow B$ is $\mathcal{M}_\mathrm{f}(A) \times B$. As usual, we will silently use Seely's isomorphisms between $\mathcal{M}_\mathrm{f}(A \uplus B)$ and $\mathcal{M}_\mathrm{f}(A) \times \mathcal{M}_\mathrm{f}(B)$.

Relational graph models correspond to a particular subclass of reflexive objects in living in **MRel**. In particular, they are all *linear* in the sense that the morphisms inducing the retraction are linear [19]. Therefore, they are also models of the resource calculus [8].

▶ **Definition 18.** A *relational graph model* $\mathcal{D} = (D, i)$ is given by an infinite set D and a total injection $i : \mathcal{M}_{\mathrm{f}}(D) \times D \to D$. We say that $\mathcal{D}$ is *extensional* when i is bijective.

We denote $i(a, \alpha)$ by $a \to_i \alpha$, or simply by $a \to \alpha$ when i is clear.

Notice that any function $f : A \to B$ can be sent to a relation $f^\dagger \in \mathbf{MRel}(A, B)$ by setting $f^\dagger = \{([a], f(a)) \mid a \in A\}$. Therefore, every rgm $\mathcal{D} = (D, i)$ induces a reflexive object.

▶ Remark. The set D is a reflexive object since $i^\dagger ; (i^{-1})^\dagger = \mathrm{Id}_{D \Rightarrow D}$. If $\mathcal{D}$ is extensional (in the sense that i is bijective) the model is extensional in the sense that $(i^{-1})^\dagger ; i^\dagger = \mathrm{Id}_D$.

Note that, when i is just injective, there are different (linear) morphisms that can be chosen as inverses. There are therefore linear reflexive objects in **MRel** that are not rgms. However, the class of extensional rgms coincide with the class of extensional (linear) reflexive objects.

Relational graph models, just like the regular ones [2], can be built by performing the free completion of a partial pair.

▶ **Definition 19.** A *partial pair* $\mathcal{A}$ is a pair (A, j) where A is a non-empty set of elements (called *atoms*) and $j : \mathcal{M}_{\mathrm{f}}(A) \times A \to A$ is a partial injection. We say that $\mathcal{A}$ is *extensional* when j is a bijection between $\mathsf{dom}(j)$ and A.

Hereafter, we will only consider partial pairs $\mathcal{A}$ whose underlying set A does not contain any pair. This is not restrictive because partial pairs can be considered up to isomorphism.

▶ **Definition 20.** The *completion* $\overline{\mathcal{A}}$ of a partial pair $\mathcal{A}$ is the pair $(\overline{A}, \overline{j})$ defined as follows: $\overline{A} = \bigcup_{n \in \mathbb{N}} A_n$, where $A_0 = A$ and $A_{n+1} = ((\mathcal{M}_{\mathrm{f}}(A_n) \times A_n) - \mathsf{dom}(j)) \cup A$; moreover

$$\overline{j}(a, \alpha) = \begin{cases} j(a, \alpha) & \text{if } (a, \alpha) \in \mathsf{dom}(j), \\ (a, \alpha) & \text{otherwise.} \end{cases}$$

We say that an atom $\alpha \in A$ has *rank 0*, whilst an element $\alpha \in \overline{A} - A$ has *rank k* if $\alpha \in A_k - A_{k-1}$. Note that, for every rgm $\mathcal{D}$ we have that $\overline{\mathcal{D}} = \mathcal{D}$ (up to isomorphism).

▶ **Proposition 21.** *If $\mathcal{A}$ is an (extensional) partial pair, then $\overline{\mathcal{A}}$ is an (extensional) rgm.*

Proof. The proof of the fact that $\overline{\mathcal{A}}$ is an rgm is analogous to the one for regular graph models [2]. It is easy to check that when j is bijective, also its completion $\overline{j}$ is. ◀

▶ **Example 22.** We define the relational analogues of:

- Engeler's model [10]: $\mathcal{E} = \overline{(\mathbb{N}, \emptyset)}$, first defined in [15],
- Scott's model [24]: $\mathcal{D}_\omega = \overline{(\{\star\}, \{([], \star) \mapsto \star\})}$, first defined (up to isomorphism) in [5],
- Coppo, Dezani and Zacchi's model [6]: $\mathcal{D}_\star = \overline{(\{\star\}, \{([\star], \star) \mapsto \star\})}$, introduced in [20].

Notice that $\mathcal{D}_\omega$ and $\mathcal{D}_\star$ are extensional, while $\mathcal{E}$ is not.

5.2 The Approximation Theorem

We now show how λ-terms and Böhm trees can be interpreted in an rgm, and we recall the main properties enjoyed by these models. Using the terminology of [1], rgms are *continuous* models, in the sense that they all enjoy the approximation theorem (Theorem 27). From this, it follows that the λ-theory induced by an extensional rgm always includes $\mathcal{H}^+$ (Corollary 29).

Given two n-uples $\vec{a}, \vec{b} \in \mathcal{M}_{\mathrm{f}}(A)^n$ we write $\vec{a} + \vec{b}$ for $(a_1 + b_1, \dots, a_n + b_n) \in \mathcal{M}_{\mathrm{f}}(A)^n$.

▶ **Definition 23.** Let $M \in \Lambda$ and let $\vec{x} \in \mathbb{V}^n$ be such that $\mathsf{FV}(M) \subseteq \vec{x}$. The *interpretation of M in $\mathcal{D}$ w.r.t. $\vec{x}$* is the relation $[\![M]\!]_{\vec{x}} \subseteq \mathcal{M}_{\mathrm{f}}(D)^n \times D$ defined inductively as follows:

- $[\![x_k]\!]_{\vec{x}} = \{(([], \dots, [], [\alpha], [], \dots, []), \alpha) \mid \alpha \in D\}$, where $[\alpha]$ stands in k-th position,
- $[\![MN]\!]^{\mathcal{D}}_{\vec{x}} = \{((\vec{a}_0 + \dots + \vec{a}_k), \alpha) \mid \exists \beta_1, \dots, \beta_k \in \mathcal{D}$, such that $(\vec{a}_0, [\beta_1, \dots, \beta_k] \to \alpha) \in [\![M]\!]^{\mathcal{D}}_{\vec{x}}$ and, for all $1 \leq \ell \leq k$, $(\vec{a}_\ell, \beta_\ell) \in [\![N]\!]^{\mathcal{D}}_{\vec{x}}\}$.
- $[\![\lambda y.N]\!]^{\mathcal{D}}_{\vec{x}} = \{(\vec{a}, (b \to \alpha)) \mid ((\vec{a}, b), \alpha) \in [\![N]\!]^{\mathcal{D}}_{\vec{x},y}\}$, where by α-equivalence we assume $y \notin \vec{x}$.

This definition extends to approximants $t \in \mathrm{NF}_\perp$ by setting $[\![\perp]\!]^{\mathcal{D}}_{\vec{x}} = \emptyset$ and to Böhm trees by interpreting all their finite approximants $[\![\mathrm{BT}(M)]\!]^{\mathcal{D}}_{\vec{x}} = \bigcup_{t \in \mathrm{BT}^*(M)} [\![t]\!]^{\mathcal{D}}_{\vec{x}}$.

Whenever we write $[\![M]\!]^{\mathcal{D}}_{\vec{x}}$ we always assume that $\mathsf{FV}(M) \subseteq \vec{x}$. When M is a combinator, we consider $[\![M]\!]^{\mathcal{D}} \subseteq D$. In all our notations we will omit the model $\mathcal{D}$ when it is clear.

▶ **Example 24.** Let $\mathcal{D}$ be any rgm. Then we have:

1. $[\![\mathbf{I}]\!]^{\mathcal{D}} = \{[\alpha] \to \alpha \mid \alpha \in D\}$ and $[\![\mathbf{1}]\!]^{\mathcal{D}} = \{[a \to \alpha] \to a \to \alpha \mid \alpha \in D, a \in \mathcal{M}_{\mathrm{f}}(D)\}$, thus:
2. $[\![\mathbf{J}]\!]^{\mathcal{D}} = \{[\alpha] \to \alpha \mid \alpha \in D'\} \subseteq [\![\mathbf{1}]\!]^{\mathcal{D}} \subseteq [\![\mathbf{I}]\!]^{\mathcal{D}}$, where D' is the smallest subset of D satisfying: if $\alpha \in D$ then $[] \to \alpha \in D'$; if $\alpha \in D$ and $a \in \mathcal{M}_{\mathrm{f}}(D')$ then $a \to \alpha \in D'$,
3. $[\![\Delta]\!]^{\mathcal{D}} = \{(a + [a \to \alpha]) \to \alpha \mid \alpha \in D, a \in \mathcal{M}_{\mathrm{f}}(D)\}$ therefore $[\![\Omega]\!]^{\mathcal{D}} = [\![\perp]\!]^{\mathcal{D}} = \emptyset$,
4. $[\![\lambda x.x\Omega]\!]^{\mathcal{D}} = \{[[] \to \alpha] \to \alpha \mid \alpha \in D\}$.

It follows that $[\![\mathbf{I}]\!] = [\![\mathbf{1}]\!]$ in both $\mathcal{D}_\omega$ and $\mathcal{D}_\star$, but $[\![\mathbf{I}]\!]^{\mathcal{D}_\omega} = [\![\mathbf{J}]\!]^{\mathcal{D}_\omega}$, while $\star \in [\![\mathbf{I}]\!]^{\mathcal{D}_\star} - [\![\mathbf{J}]\!]^{\mathcal{D}_\star}$.

Rgms satisfy the following substitution property, and are sound models of λ-calculus.

▶ **Lemma 25** (cf. [22]). *Let $M, N \in \Lambda$ and $\mathcal{D}$ be a relational graph model.*

1. *(Substitution) $(\vec{a}, \alpha) \in [\![M\{N/y\}]\!]^{\mathcal{D}}_{\vec{x}}$ iff there are $\beta_1, \dots, \beta_k \in D$, $\vec{a}_0, \dots, \vec{a}_k \in \mathcal{M}_{\mathrm{f}}(D)^n$ such that $(\vec{a}_\ell, \beta_\ell) \in [\![N]\!]^{\mathcal{D}}_{\vec{x}}$, for $1 \leq \ell \leq k$, $((a_0, [\beta_1, ..., \beta_k]), \alpha) \in [\![M]\!]^{\mathcal{D}}_{\vec{x},y}$ and $\vec{a} = \sum_{\ell=0}^{k} \vec{a}_\ell$.*
2. *(Soundness) If $M =_\beta N$ then $[\![M]\!]^{\mathcal{D}}_{\vec{x}} = [\![N]\!]^{\mathcal{D}}_{\vec{x}}$ for all $\vec{x}$ containing $\mathsf{FV}(M) \cup \mathsf{FV}(N)$.*

The λ-*theory* induced by $\mathcal{D}$ is defined by $\mathrm{Th}(\mathcal{D}) = \{(M, N) \in \Lambda \times \Lambda \mid [\![M]\!]_{\vec{x}} = [\![N]\!]_{\vec{x}}\}$. The *preorder theory* induced by $\mathcal{D}$ is given by $\mathrm{Th}_\sqsubseteq(\mathcal{D}) = \{(M, N) \in \Lambda \times \Lambda \mid [\![M]\!]_{\vec{x}} \subseteq [\![N]\!]_{\vec{x}}\}$. We will write $\mathcal{D} \models M = N$ when $(M, N) \in \mathrm{Th}(\mathcal{D})$, and $\mathcal{D} \models M \sqsubseteq N$ when $(M, N) \in \mathrm{Th}_\sqsubseteq(\mathcal{D})$.

▶ **Definition 26.** A model $\mathcal{D}$ is called *inequationally $\mathcal{O}$-fully abstract* when $\mathcal{D} \models M \sqsubseteq N$ iff $M \sqsubseteq^{\mathcal{O}} N$. A model $\mathcal{D}$ is called *$\mathcal{O}$-fully abstract* whenever $\mathcal{D} \models M = N$ iff $M \equiv^{\mathcal{O}} N$.

It is easy to check that in every extensional rgm $\mathcal{D}$ we have $\mathcal{D} \models \mathbf{I} = \mathbf{1}$, thus $\boldsymbol{\lambda}\eta \subseteq \mathrm{Th}(\mathcal{D})$. As a consequence, the λ-theories induced by rgms and by ordinary graph models are different, since no graph model is extensional. For instance, the λ-theory of $\mathcal{D}_\omega$, the relational analogue of Scott's model $\mathcal{D}_\infty$, is $\mathcal{H}^\star$ [18]. In other words, the model $\mathcal{D}_\omega$ is hnf-fully abstract.

All rgms enjoy the approximation theorem for Böhm trees below. As usual, this result can be proved via techniques based on finite approximants (see [18]). However, in [20] we provided a new proof exploiting the following facts: rgms are models of Ehrhard and Regnier's resource calculus [8]; they satisfy the Taylor expansion [19]; two λ-terms have the same Böhm tree iff the normal form of the support of their Taylor expansions coincide [9].

Recall from page 4 that $\mathrm{BT}^*(M)$ denotes the set of all finite approximants of $\mathrm{BT}(M)$.

▶ **Theorem 27** (Approximation Theorem [20]). *Let M be a λ-term. Then $(\vec{a}, \alpha) \in [\![M]\!]_{\vec{x}}$ if and only if there exists $t \in \mathrm{BT}^*(M)$ such that $(\vec{a}, \alpha) \in [\![t]\!]_{\vec{x}}$. Therefore $[\![M]\!]_{\vec{x}} = [\![\mathrm{BT}(M)]\!]_{\vec{x}}$.*

▶ **Corollary 28.** *For all rgms $\mathcal{D}$ we have that $\mathcal{B} \subseteq \mathrm{Th}(\mathcal{D})$. In particular $\mathrm{Th}(\mathcal{D})$ is sensible and $[\![M]\!]^{\mathcal{D}}_{\vec{x}} = \emptyset$ for all unsolvable λ-terms M.*

Proof. From Theorem 27 we get $[\![M]\!]_{\vec{x}} = [\![\mathrm{BT}(M)]\!]_{\vec{x}} = \bigcup_{t \in \mathrm{BT}^*(M)} [\![t]\!]_{\vec{x}}$. Therefore, whenever $\mathrm{BT}(M) = \mathrm{BT}(N)$ we have $[\![M]\!]_{\vec{x}} = [\![\mathrm{BT}(M)]\!]_{\vec{x}} = [\![\mathrm{BT}(N)]\!]_{\vec{x}} = [\![N]\!]_{\vec{x}}$. Thus $\mathcal{B} \subseteq \mathrm{Th}(\mathcal{D})$. ◀

In the next corollary we are going to use the following characterization of Morris's preorder $\sqsubseteq^{\mathrm{nf}}$, which is based on Lévy's notion of *extensional Böhm trees* [16]:

$$\mathrm{BT}^e(M) = \{\mathsf{nf}_\eta(t) \mid t \in \mathrm{BT}^*(M'), M' \twoheadrightarrow_\eta M\}.$$

By [13], we have $M \sqsubseteq^{\mathrm{nf}} N$ if and only if $\mathrm{BT}^e(M) \subseteq \mathrm{BT}^e(N)$.

▶ **Corollary 29.** *The order theory of any extensional* rgm $\mathcal{D}$ *contains* $\sqsubseteq^{\mathrm{nf}}$*, so* $\mathcal{H}^+ \subseteq \mathrm{Th}(\mathcal{D})$.

Proof. From Theorem 27 we obtain $[\![M]\!]_{\vec{x}} = \bigcup_{t\in\mathrm{BT}^*(M)}[\![t]\!]_{\vec{x}}$. From the extensionality of $\mathcal{D}$, we get $[\![M]\!]_{\vec{x}} = \bigcup_{M'\twoheadrightarrow_\eta M, t\in\mathrm{BT}^*(M')}[\![t]\!]_{\vec{x}} = \bigcup_{M'\twoheadrightarrow_\eta M, t\in\mathrm{BT}^*(M')}[\![\mathsf{nf}_\eta(t)]\!]_{\vec{x}} = [\![\mathrm{BT}^e(M)]\!]_{\vec{x}}$. So, we have that $\mathrm{BT}^e(M) \subseteq \mathrm{BT}^e(N)$ entails $[\![M]\!]_{\vec{x}} = [\![\mathrm{BT}^e(M)]\!]_{\vec{x}} \subseteq [\![\mathrm{BT}^e(N)]\!]_{\vec{x}} = [\![N]\!]_{\vec{x}}$. ◀

6 Characterizing Fully Abstract Relational Models of $\mathcal{H}^+$

In this section we provide a characterization of those rgms which are fully abstract for $\mathcal{H}^+$. We first introduce the notion of λ-*König* rgm, and then we show that an rgm $\mathcal{D}$ is extensional and λ-König exactly when the induced order theory is Morris's preorder $\sqsubseteq^{\mathrm{nf}}$ (Theorem 36).

6.1 Lambda König Relational Graph Models

Before entering into the technicalities we try to give the intuition behind our condition. By Lemma 8 and Theorem 17, two λ-terms M, N are equal in $\mathcal{H}^*$, but different in $\mathcal{H}^+$, when there is a position σ such that, say, $\mathrm{BT}(M')_\sigma = x$ for some $M' \twoheadrightarrow_\eta M$, while $\mathrm{BT}(N)_\sigma$ is an infinite η-expansion of x following some $T \in \mathbb{T}^\infty_{\mathsf{rec}}$.

Therefore our models need to separate x from any $\mathbf{J}_T x$ for $T \in \mathbb{T}^\infty_{\mathsf{rec}}$.

In an extensional rgm $\mathcal{D}$, every α is equal to an arrow, so we can always try to unfold it following a function f: starting from $\alpha = \alpha_0$, at level ℓ we have $\alpha_\ell = a_0 \to \cdots \to a_{f(\ell)} \to \alpha'$ and, as long as there is an $\alpha_{\ell+1} \in a_{f(\ell)}$, we can keep unfolding it at level $\ell + 1$. There are now two possibilities. If this process continues indefinitely, then we consider that α can actually be unfolded following f. Otherwise, if at some level ℓ we have $a_{f(\ell)} = []$, then the process is forced to stop and α cannot be unfolded following f.

Now, since $T \in \mathbb{T}^\infty_{\mathsf{rec}}$ is a finitely branching infinite tree, by König's lemma there is an infinite path f in $\mathrm{BT}(\mathbf{J}_T)$, and since the interpretation of $\mathbf{J}_T$ is inductive (rather than coinductive) we will have $[\alpha] \to \alpha \notin [\![\mathbf{J}_T]\!]$ for any α whose unfolding can actually follow f. In some sense such an α is witnessing *within the model* the existence of an infinite path f in T, and therefore in $\mathbf{J}_T$. The following is a coinductive definition[1] of such a witness.

▶ **Definition 30.** Let $\mathcal{D}$ be an rgm, $T \in \mathbb{T}^\infty_{\mathsf{rec}}$ and $f \in \Pi(T)$. An element $\alpha \in D$ is a *witness for* T *following* f if there exist $a_0, \ldots, a_{f(0)} \in \mathcal{M}_{\mathrm{f}}(\mathcal{D})$ and $\alpha' \in \mathcal{D}$ such that

$$\alpha = a_0 \to \cdots \to a_{f(0)} \to \alpha' \text{ and there is a witness } \beta \in a_{f(0)} \text{ for } T{\restriction_{\langle f(0)\rangle}} \text{ following } f^{\geq 1}$$

where $f^{\geq 1}$ denotes the function $k \mapsto f(k+1)$. We simply say that α is a *witness for* T when there exists an $f \in \Pi(T)$ such that α is a witness for T following f.

We denote by $\mathsf{W}_{\mathcal{D}}(T)$ (resp. $\mathsf{W}_{\mathcal{D},f}(T)$) the set of all witnesses for T (resp. following f). When the model $\mathcal{D}$ is clear from the context, we will simply write $\mathsf{W}(T)$ (resp. $\mathsf{W}_f(T)$).

We formalize the intuition given above by showing that $\mathsf{W}_{\mathcal{D}}(T)$ is constituted by those $\alpha \in D$ such that $[\alpha] \to \alpha \notin [\![\mathbf{J}_T]\!]$. We first prove the following technical lemma.

[1] I.e., we are defining the greatest relation $\mathsf{W} \subseteq D \times \mathbb{T}^\infty_{\mathsf{rec}} \times (\mathbb{N} \to \mathbb{N})$ satisfying the condition of Def. 30.

▶ **Lemma 31.** *Let $\mathcal{D}$ be an extensional* rgm*. For all $k \in \mathbb{N}$, $T \in \mathbb{T}^{\infty}_{\mathsf{rec}}$, $\alpha \in \mathsf{W}_{D}(T)$ and $t \in \mathrm{BT}^{k}(\mathbf{J}_T x)$ we have $([\alpha], \alpha) \notin [\![t]\!]_x$.*

Proof. We proceed by induction on k.

Case $k = 0$. This case is trivial since $\mathrm{BT}^0(\mathbf{J}_T x) = \{\bot\}$ and $[\![\bot]\!]_x = \emptyset$.

Case $k > 0$. If $t = \bot$ then $[\![\bot]\!]_x = \emptyset$, otherwise $t = \lambda y_0 \dots y_{T(\varepsilon)-1}.x t_0 \cdots t_{T(\varepsilon)-1}$ where each $t_i \in \mathrm{BT}^{k-1}(\mathbf{J}_{T\restriction_{\langle i \rangle}} y_i)$. As the model is extensional $\alpha = a_0 \to \cdots \to a_{T(\varepsilon)-1} \to \alpha'$, for some $a_i = [\alpha_{i,1}, \dots, \alpha_{i,k_i}]$. Hence $([\alpha], \alpha) \in [\![t]\!]_x$ if and only if $(([\alpha], a_0, \dots, a_{T(\varepsilon)-1}), \alpha') \in [\![x t_0 \cdots t_{T(\varepsilon)-1}]\!]_{x, y_0, \dots, y_{T(\varepsilon)-1}}$. By exploiting the facts that $\mathsf{FV}(t_i) \subseteq \{y_i\}$ and $[\![t_i]\!]_{y_i} \subseteq [\![y_i]\!]_{y_i}$, we obtain $([\alpha_{i,j}], \alpha_{i,j}) \in [\![t_i]\!]_{y_i}$ for all $i \leq T(\varepsilon) - 1$ and $j \leq k_i$. Since $\alpha \in \mathsf{W}_f(T)$ for some f, there exists a witness $\alpha_{f(0),j} \in a_{f(0)}$ for $T\restriction_{\langle f(0) \rangle}$ following $f^{\geq 1}$. By $\alpha_{f(0),j} \in \mathsf{W}(T\restriction_{\langle f(0) \rangle})$ and the induction hypothesis we get $([\alpha_{f(0),j}], \alpha_{f(0),j}) \notin [\![t_{f(0)}]\!]_{y_{f(o)}}$, which is a contradiction. ◀

By applying the Approximation Theorem we get the following characterization of $\mathsf{W}_{\mathcal{D}}(T)$.

▶ **Proposition 32.** *For any extensional* rgm *$\mathcal{D}$ and any tree $T \in \mathbb{T}^{\infty}_{\mathsf{rec}}$:*

$$\mathsf{W}_{\mathcal{D}}(T) = \{\alpha \in D \mid ([\alpha], \alpha) \notin [\![\mathbf{J}_T x]\!]_x\}.$$

Proof.

($\subseteq$) Follows immediately from the Approximation Theorem 27 and from Lemma 31.

($\supseteq$) Let $\alpha \in D$ such that $([\alpha], \alpha) \notin [\![\mathbf{J}_T x]\!]_x$. We coinductively construct a path f such that $\alpha \in \mathsf{W}_f(T)$. As T is infinite we have $\mathbf{J}_T x =_\beta \lambda x_0 \dots x_n.x(\mathbf{J}_{T\restriction_{\langle 0 \rangle}} x_0) \cdots (\mathbf{J}_{T\restriction_{\langle n \rangle}} x_n)$ and since $\mathcal{D}$ is extensional $\alpha = a_0 \to \cdots \to a_n \to \alpha'$. From $([\alpha], \alpha) \notin [\![\mathbf{J}_T x]\!]_x$ and Lemma 25.2 we get $([\alpha], \alpha) \notin [\![\lambda x_0 \dots x_n.x(\mathbf{J}_{T\restriction_{\langle 0 \rangle}} x_0) \cdots (\mathbf{J}_{T\restriction_{\langle n \rangle}} x_n)]\!]_x$, therefore there is an index $k \leq n$ such that $a_k \neq []$ and an element $\beta \in a_k$ such that $([\beta], \beta) \notin [\![\mathbf{J}_{T\restriction_{\langle k \rangle}} x_k]\!]_{x_k}$. By the coinductive hypothesis, there exists a function g such that $\beta \in \mathsf{W}_g(T\restriction_{\langle k \rangle})$. We conclude since $\alpha \in \mathsf{W}_f(T)$ where f is the function defined as follows: $f(0) = k$ and $f(n+1) = g(n)$ for all $n \in \mathbb{N}$. ◀

It should be now clear that in an rgm $\mathcal{D}$ fully abstract for $\mathcal{H}^+$, every infinite λ-definable tree needs an element in D witnessing its infinite path, which exists by König's lemma.

▶ **Definition 33** (λ-König models)**.** An rgm $\mathcal{D}$ is *λ-König* if for every $T \in \mathbb{T}^{\infty}_{\mathsf{rec}}$, $\mathsf{W}_{\mathcal{D}}(T) \neq \emptyset$.

6.2 The Characterization

We will focus on the λ-König condition, since the extensionality is necessary, as $\boldsymbol{\lambda}\eta \subseteq \mathcal{H}^+$.

Lambda-König Implies Inequational Full Abstraction

Let $\mathcal{D}$ be an extensional λ-König relational graph model. Since every $T \in \mathbb{T}^{\infty}_{\mathsf{rec}}$ has a non-empty set of witnesses $\mathsf{W}_{\mathcal{D}}(T)$, by Proposition 32, there is an element $\alpha \in \mathsf{W}_{\mathcal{D}}(T)$ such that $[\alpha] \to \alpha \notin [\![\mathbf{I}]\!] - [\![\mathbf{J}_T]\!]$. Thus, $\mathcal{D}$ separates $\mathbf{I}$ from all the $\mathbf{J}_T$'s.

▶ **Theorem 34** (Inequational Full Abstraction)**.** *Let $\mathcal{D}$ be an extensional λ-König* rgm*, then:*

$$M \sqsubseteq^{\mathrm{nf}} N \iff \mathcal{D} \models M \sqsubseteq N\,.$$

Proof.

($\Rightarrow$) This follows directly from Corollary 29.

($\Leftarrow$) We assume, by the way of contradiction, that $\mathcal{D} \models M \sqsubseteq N$ but $M \not\sqsubseteq^{\mathrm{nf}} N$. Since $\sqsubseteq^{\mathrm{hnf}}$ is maximal (cf. [1, Lemma 16.2.4]) and $[\![M]\!]_{\vec{x}} \subseteq [\![N]\!]_{\vec{x}}$ we must have $M \sqsubseteq^{\mathrm{hnf}} N$. By Theorem 17 there exists a context $C[-]$ such that $C[M] =_{\beta\eta} \mathbf{I}$ and $C[N] =_{\mathcal{B}} \mathbf{J}_T$ for some $T \in \mathbb{T}^{\infty}_{\mathsf{rec}}$. Since $[\![-]\!]$ is contextual and, by Corollary 29, $\mathcal{B}\eta \subseteq \mathcal{H}^+ \subseteq \mathrm{Th}(\mathcal{D})$ we have $[\![\mathbf{I}]\!] = [\![C[M]]\!] \subseteq [\![C[N]]\!] = [\![\mathbf{J}_T]\!]$. We derive a contradiction by applying Proposition 32. ◀

Inequational Full Abstraction Implies Lambda-König

▶ **Theorem 35.** *Let $\mathcal{D}$ be an rgm. If $\mathrm{Th}_{\sqsubseteq}(\mathcal{D}) = \sqsubseteq^{\mathrm{nf}}$ then $\mathcal{D}$ is extensional and λ-König.*

Proof. Obviously $\mathcal{D}$ must be extensional since $\mathcal{H}^*$ is an extensional λ-theory. By the way of contradiction, we suppose that it is not λ-König, then there is $T \in \mathbb{T}^{\infty}_{\mathsf{rec}}$ such that $\mathsf{W}_{\mathcal{D}}(T) = \emptyset$ and, by Proposition 32, we get $[\![\mathbf{I}]\!] = [\![\mathbf{J}_T]\!]$. This is impossible since $\mathbf{I} \not\sqsubseteq^{\mathrm{nf}} \mathbf{J}_T$. ◀

From Theorems 34 and 35 we get the main semantic result of the paper.

▶ **Theorem 36.** *An extensional rgm $\mathcal{D}$ is λ-König if and only if $\mathcal{D}$ is inequationally fully abstract for Morris's preorder $\sqsubseteq^{\mathrm{nf}}$.*

The following result first appeared in [20].

▶ **Corollary 37.** *The model $\mathcal{D}_\star$ of Example 22 is inequationally fully abstract for Morris's preorder $\sqsubseteq^{\mathrm{nf}}$. In particular $\mathrm{Th}(\mathcal{D}_\star) = \mathcal{H}^+$.*

7 The λ-Theory $\mathcal{H}^+$ Satisfies the ω-Rule

This section is devoted to show that $\mathcal{H}^+$ satisfies the ω-rule. We will focus on its restriction to closed terms ω^0, and conclude since $\mathcal{T} \vdash \omega$ if and only if $\mathcal{T} \vdash \omega^0$, as shown in [12].

Recall that, by the context lemma, if two closed λ-terms M and N are such that $M \not\sqsubseteq^{\mathrm{hnf}} N$ holds, then the context $C[-]$ semi-separating them can be chosen applicative and closed, that is $C[-] = [-]\vec{Z}$ for $\vec{Z} \in \Lambda^o$.

▶ **Lemma 38.** *Let $M, N \in \Lambda^o$ be such that $M \in_\beta \mathrm{NF}_\beta$, while $N \notin_\beta \mathrm{NF}_\beta$. Then, there exist $n \geq 1$ and combinators $Z_1, \dots, Z_n \in \Lambda^o$ such that $M\vec{Z} \in_\beta \mathrm{NF}_\beta$ while $N\vec{Z} \notin_\beta \mathrm{NF}_\beta$.*

Proof. By hypothesis $M \not\sqsubseteq_{\mathcal{H}^+} N$. There are two possible cases.

Case $M \sqsubseteq^{\mathrm{hnf}} N$. Therefore, by Theorem 17 there are $Z_1, \dots, Z_k \in \Lambda^o$ such that $M\vec{Z} =_{\beta\eta} \mathbf{I}$ and $N\vec{Z} =_{\mathcal{B}} \mathbf{J}_T$ for some $T \in \mathbb{T}^{\infty}_{\mathsf{rec}}$. If $k = 0$ just take the λ-term $\mathbf{1}_{T(\varepsilon)}$ as Z_1 and conclude since $\mathbf{J}_T \mathbf{1}_{T(\varepsilon)} =_{\mathcal{B}} \mathbf{J}_T$.

Case $M \not\sqsubseteq^{\mathrm{hnf}} N$. By semi-separability, there are $Z_1, \dots, Z_k \in \Lambda^o$ such that $M\vec{Z} =_\beta \mathbf{I}$ and $N\vec{Z} =_\beta U$ for some unsolvable U. If $k = 0$ just take the identity $\mathbf{I}$ as Z_1. ◀

▶ **Lemma 39.** *Let $M, N \in \Lambda^o$. If $\forall Z \in \Lambda^o, MZ =_{\mathcal{H}^+} NZ$, then*

$$\forall \vec{P} \in \Lambda^o (M\vec{P} \in_\beta \mathrm{NF}_\beta \iff N\vec{P} \in_\beta \mathrm{NF}_\beta).$$

Proof. Suppose that for all $Z, \vec{Q} \in \Lambda^o$, $MZ\vec{Q} \in_\beta \mathrm{NF}_\beta$ if and only if $NZ\vec{Q} \in_\beta \mathrm{NF}_\beta$. We show by induction on the length k of $\vec{P} \in \Lambda^o$ that $M\vec{P} \in_\beta \mathrm{NF}_\beta$ if and only if $N\vec{P} \in_\beta \mathrm{NF}_\beta$.

Base case: $k = 0$. Since the contrapositive holds by Lemma 38.

Induction case: $k > 0$. It follows trivially from the induction hypothesis. ◀

This shows that $\mathcal{H}^+ \vdash \omega^0$. As a consequence, we get our main syntactic result.

▶ **Theorem 40.** *$\mathcal{H}^+$ satisfies the ω-rule.*

This solves positively the question whether $\mathcal{H}^+ \vdash \omega$ holds. The question whether a more constructive proof can be provided will be addressed in further works.

Sallé's conjecture saying that the inclusion $\mathcal{B}\omega \subseteq \mathcal{H}^+$ is proper, which is stated in the proof of [1, Thm. 17.4.16], is still open and under investigation.

▶ Remark. In first-order logic, ω-completeness is closely related to the notion of a *standard model*, which has as domain the Herbrand's universe generated by the signature of the given theory. In higher-order languages, Morris-style observational equality is the untyped analogue of extensional equality defined by a logical relation on – again – the ground *term model* of the higher-order language.

Our result perhaps gives some indication that these logical and computational aspects are not at all independent, with the universal property of observational equality being powerful enough to imply ω-completeness in the purely syntactic sense of validating the ω-rule.

Acknowledgements. This work is partly supported by ANR Project Coquas 12JS0200601. We are grateful to Barendregt, Intrigila and Ronchi della Rocca for interesting discussions. We thank the anonymous reviewers for providing helpful remarks to improve the presentation of this paper.

References

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

2 C. Berline. From computation to foundations via functions and application: The λ-calculus and its webbed models. *Theoretical Computer Science*, 249(1):81–161, 2000.

3 C. Böhm. Alcune proprietà delle forme β-η-normali nel λ-K-calcolo. *Pub. INAC*, 696, 1968.

4 F. Breuvart. On the characterization of models of $\mathcal{H}^*$. In T. Henzinger and D. Miller, editors, *Joint Meeting CSL-LICS'14*, pages 24:1–24:10. ACM, 2014.

5 A. Bucciarelli, T. Ehrhard, and G. Manzonetto. Not enough points is enough. In *CSL'07*, volume 4646 of *Lecture Notes in Comput. Sci.*, pages 298–312. Springer, 2007.

6 M. Coppo, M. Dezani, and M. Zacchi. Type theories, normal forms and D_∞-lambda-models. *Inf. Comput.*, 72(2):85–116, 1987.

7 M. Dezani-Ciancaglini and E. Giovannetti. From Bohm's theorem to observational equivalences: an informal account. *Electr. Notes Theor. Comput. Sci.*, 50(2):83–116, 2001.

8 T. Ehrhard and L. Regnier. The differential λ-calculus. *Th. Comp. Sci.*, 309(1):1–41, 2003.

9 T. Ehrhard and L. Regnier. Böhm trees, Krivine's machine and the Taylor expansion of lambda-terms. In *CiE*, volume 3988 of *Lecture Notes in Comput. Sci.*, pages 186–197, 2006. doi:10.1007/11780342_20.

10 E. Engeler. Algebras and combinators. *Algebra Universalis*, 13(3):389–392, 1981.

11 P. Di Gianantonio, G. Franco, and F. Honsell. Game semantics for untyped $\lambda\beta\eta$-calculus. In *TLCA'99*, volume 1581 of *Lecture Notes in Comput. Sci.*, pages 114–128. Springer, 1999.

12 R. Hindley and G. Longo. Lambda-calculus models and extensionality. *Z. Math. Logik Grundlag. Math.*, 26(4):289–310, 1980.

13 J.M.E. Hyland. A survey of some useful partial order relations on terms of the λ-calculus. In *Lambda-Calculus and Comp. Sci. Th.*, volume 37 of *LNCS*, pages 83–95. Springer, 1975.

14 J.M.E. Hyland. A syntactic characterization of the equality in some models for the λ-calculus. *J. London Math. Soc. (2)*, 12(3):361–370, 1975/76.

15 J.M.E. Hyland, M. Nagayama, J. Power, and G. Rosolini. A category theoretic formulation for Engeler-style models of the untyped λ-calculus. *Electr. Notes in TCS*, 161:43–57, 2006.

16 J.-J. Lévy. Le lambda calcul – notes du cours, 2005. In French. URL: `http://pauillac.inria.fr/~levy/courses/X/M1/lambda/dea-spp/jjl.pdf`.

17 S. Lusin and A. Salibra. The lattice of λ-theories. *J. Log. Comput.*, 14(3):373–394, 2004.

18 G. Manzonetto. A general class of models of $\mathcal{H}^\star$. In *MFCS 2009*, volume 5734 of *Lecture Notes in Comput. Sci.*, pages 574–586. Springer, 2009.

19 G. Manzonetto. What is a categorical model of the differential and the resource λ-calculi? *Mathematical Structures in Computer Science*, 22(3):451–520, 2012.

20 G. Manzonetto and D. Ruoppolo. Relational graph models, Taylor expansion and extensionality. *Electr. Notes Theor. Comput. Sci.*, 308:245–272, 2014.

21 J.H. Morris. *Lambda calculus models of programming languages.* PhD thesis, MIT, 1968.

22 L. Paolini, M. Piccolo, and S. Ronchi Della Rocca. Essential and relational models. *Mathematical Structures in Computer Science*, 2015. `doi:10.1017/S0960129515000316`.

23 S. Ronchi Della Rocca and L. Paolini. *The Parametric Lambda Calculus – A Metamodel for Computation.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

24 D. Scott. Continuous lattices. In Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, volume 274 of *Lecture Notes in Math.*, pages 97–136. Springer, 1972.

25 P. Severi and F.J. de Vries. The infinitary lambda calculus of the infinite eta Böhm trees. *To appear in Math. Struct. Comp. Sci.*, 2016.

26 C.P. Wadsworth. The relation between computational and denotational properties for Scott's D_∞-models of the lambda-calculus. *SIAM J. Comput.*, 5(3):488–521, 1976.

Modular Focused Proof Systems for Intuitionistic Modal Logics

Kaustuv Chaudhuri[1], Sonia Marin*[2], and Lutz Straßburger[3]

1 **Inria & LIX/École polytechnique, France**
`kaustuv.chaudhuri@inria.fr`

2 **Inria & LIX/École polytechnique, France**
`sonia.marin@inria.fr`

3 **Inria & LIX/École polytechnique, France**
`lutz.strassburger@inria.fr`

Abstract

Focusing is a general technique for syntactically compartmentalizing the non-deterministic choices in a proof system, which not only improves proof search but also has the representational benefit of distilling sequent proofs into synthetic normal forms. However, since focusing is usually specified as a restriction of the sequent calculus, the technique has not been transferred to logics that lack a (shallow) sequent presentation, as is the case for some of the logics of the modal cube. We have recently extended the focusing technique to classical nested sequents, a generalization of ordinary sequents. In this work we further extend focusing to intuitionistic nested sequents, which can capture all the logics of the intuitionistic S5 cube in a modular fashion. We present an internal cut-elimination procedure for the focused system which in turn is used to show its completeness.

1998 ACM Subject Classification F.4.2 [Mathematical logic] Proof theory

Keywords and phrases intuitionistic modal logic, focusing, proof search, cut elimination, nested sequents

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.16

1 Introduction

When one adds features to a proof system, one generally expects that the meta-theory of the system becomes more complicated. Take, for example, the one-sided sequent calculus G3c for classical propositional logic, which has just as many logical rules as connectives and two additional structural rules of identity and cut. Eliminating cuts from this system is relatively straightforward: there is a single cut rule and a simple lexicographic induction on the cut rank and heights of derivations. If we move to a system with multiplicative rules and structural rules of contraction and weakening, such as Gentzen's original system LK, then the single cut rule and lexicographic measure is no longer sufficient to handle permutations of cuts with contraction and weakening: we either need to add additional rules such as mix or we need to use a sophisticated induction measure that takes the number of contractions on the cut formula into account. Extending the system with the modal connectives $\Box$ and $\Diamond$ and the modal axiom k adds new forms of cuts and further complications to the measure. Adding other modal axioms such as t or 4 causes new structural rules to appear that now need to be considered for cut permutations. Other modal axioms such as 5 causes the very structure of (list-like) sequents to no longer be adequate for building analytic proof systems, so the notion

* Supported by the ERC Advanced Grant *ProofCert*.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 16; pp. 16:1–16:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of sequent needs to be generalized, say to *labeled sequents* [15, 20] or to *hypersequents* [2]. Needless to say, the cut rules for such generalized sequents – indeed, the entire cut-elimination procedure – needs to be re-examined. Finally, moving to a two-sided sequent calculus, which is essential for intuitionistic versions of these calculi, doubles the number of inference rules, and hence doubles the number of cases to consider in the cut-elimination argument.

In this paper, we report on some observations that seem to suggest that this trend of increasing syntactic and meta-theoretic complexity can be halted and even reversed if one designs the proof system to enforce certain normal forms. We use the intuitionistic propositional logics of the modal S5 cube (see Figure 1) as our testbed, as it contains all permutations of the complications mentioned in the previous paragraph. We did not begin this work with the goal of improving the proof systems for such logics; we were instead interested in the pragmatic question of automated proof search in modal logics, in both their classical and intuitionistic dialects. For such logics, proof systems based on *nested sequents*, a generalization of the usual list-like sequents (as formulated by Gentzen) to tree-like structures [4], turn out to have certain desirable properties from the perspective of proof search. Specifically, (1) they are *analytic*, meaning that every theorem can be proved using only sequents built from subformulas of the theorem; (2) their meta-theory is *internal*, which means that procedures such as cut-elimination operate directly on the proofs in the system rather than by translation to a different system; and (3) they are *modular*, which means that the axes of extension in the modal cube correspond exactly to the choice of specific inference rules. This final desideratum of modularity turns out to be fairly non-trivial [4, 5, 12].

One direct way to improve proof search is to reduce the *proof search space*, which lets a search procedure make fewer choices to get farther. Over the past two decades, the *focusing* technique, originally developed for (linear) logic programming [14, 1] has turned out to be a generally applicable method of reducing the proof search space that remains complete (*i.e.*, every theorem has a focused proof). It has been transplanted from its origin in the sequent calculus for linear logic [1] to a wide variety of logics [8, 11, 17] and proof systems [6, 3, 7], and it is empirically a very "high impact" optimization to standard proof search procedures [8, 13]. This generality suggests that the ability to transform a proof system into a focused form is a good indication of its syntactic quality, in a manner similar to how admissibility of cut shows that a proof system is syntactically consistent.

We have recently shown how to adapt the focusing technique to the classical nested proof systems [7]. Now, nested proof systems also exist for the intuitionistic versions of these modal systems [21, 12], so it is natural to ask if our focusing technique applies here as well. The intuitionistic restriction in these systems is achieved by means of an *input/output annotation* on the formulas that corresponds to whether that formula is a hypothesis or a conclusion [9]. As long as there is exactly one output in a sequent, its semantic meaning is intuitionistic, and hence the inference rules of the system are designed to preserve this singular occurrence of output formulas. These annotations cause every rule corresponding to connectives and the modal axioms to have two incarnations, one for an input-annotated and the other for an output-annotated formula. Section 3 summarizes the system NIK from [21] that we use as the basis of our focused systems.

Our starting point, therefore, was a focused version of the proof system containing annotated formulas. However, we were surprised to discover that: (1) the input/output annotations turn out to be redundant, as they can always be uniquely inferred; and that (2) in the *synthetic* form (Section 5) of the system, which elides the details of the focused logical rules and records only the *phase transitions*, there is only a single modal *structural* rule that is needed for every axiom. It turns out that the synthetic version of the system has *fewer* structural rules than the non-focused version, and the same number of structural rules as the classical system, which we did not expect would be the case. The input/output annotations are shown to be unnecessary by the use of *polarized* syntax that separates the classes of *positive* formulas, whose output rules are non-invertible (therefore requiring non-deterministic choices during search) and *negative* formulas, whose input rules are non-invertible. We can show that intuitionistically meaningful polarized sequents are exactly those sequents with a single negative formula interpreted as the output.

Like in the classical case [7], our main technical contribution is the proof of completeness of the focused calculus by means of an internal cut-elimination proof. In the process of writing this proof, we discovered a further simplification of the synthetic version of the focused system: the so called *store* rule of focused calculi [7, Figure 3] that we used earlier in the classical system is also unnecessary. Indeed, removing the store rule makes the *decision* and *release* rules of the system correspond exactly to the introduction rules for the two shift connectives $\downarrow$ and $\uparrow$, respectively, that inject each polarized class into the other. This simplification in turn makes the three cuts that were required in the classical cut-elimination argument [7, Figures 6 and 9] merely variants of a single cut rule. Moreover, since this simplification was effectively independent of the classical or intuitionistic flavor of the logic, we observe exactly the same reduction of the number of cut rules to just a single rule in the intuitionistic synthetic system described in Section 5. We then obtain a cut-elimination proof (Theorem 6.6) – and its corollary, the completeness of the synthetic system – that is considerably *shorter* and *simpler*, using a more standard induction measure, than the corresponding proof in [7].

Besides these technical contributions, we would like to stress the following conceptual point: focusing, written in a synthetic form, is not a complication one *adds* to a proof system and its associated meta-theory, but a *simplification* of both. Such a simplification has already been observed for ordinary intuitionistic logic by Zeilberger [22]. As we add more features to a logic, the effect of this simplification becomes more noticeable.

2 Preliminaries on Intuitionistic Modal Logic

We will work with the following grammar of *formulas* (written $A, B, \ldots$), which are built from a collection of *atomic formulas* (written $a, b, \ldots$).

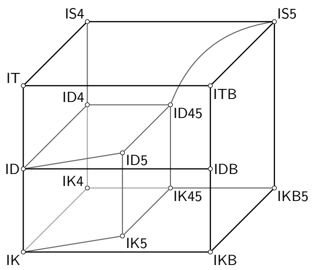
$$A, B, \ldots ::= a \mid A \wedge B \mid \top \mid A \vee B \mid \bot \mid A \supset B \mid \Box A \mid \Diamond A$$

This grammar is slightly redundant because $\top$ can be defined as $a \supset a$ for some atom a. We nevertheless keep it in the syntax because one of the polarized versions of $\top$, which we will encounter in Section 4, will turn out to be non-redundant. Recall that classical modal logic K is obtained from classical propositional logic by adding to any standard formulation, such as Hilbert's axiomatization,

- a *necessitation* rule that says that $\Box A$ is a theorem of K if A is a theorem; and
- the axiom of *distributivity*, commonly called k: $\Box(A \supset B) \supset (\Box A \supset \Box B)$.

Obtaining the intuitionistic variant of K is more involved. Lacking De Morgan duality, there are several variants of k that are classically but not intuitionistically equivalent. In this paper, we consider the intuitionistic variant of the modal logic K, called IK, that is obtained from ordinary intuitionistic propositional logic (IPL) by

- adding the *necessitation rule*: $\Box A$ is a theorem of IK if A is a theorem; and



$$\begin{array}{rll}
\mathsf{d}\colon & \Box A \supset \Diamond A & \text{(Seriality)}\\
\mathsf{t}\colon & (A \supset \Diamond A) \wedge (\Box A \supset A) & \text{(Reflexivity)}\\
\mathsf{b}\colon & (A \supset \Box\Diamond A) \wedge (\Diamond\Box A \supset A) & \text{(Symmetry)}\\
\mathsf{4}\colon & (\Diamond\Diamond A \supset \Diamond A) \wedge (\Box A \supset \Box\Box A) & \text{(Transitivity)}\\
\mathsf{5}\colon & (\Diamond A \supset \Box\Diamond A) \wedge (\Diamond\Box A \supset \Box A) & \text{(Euclideanness)}
\end{array}$$

Figure 1 The intuitionistic modal S5 cube and the five constituent axioms.

- adding the following five variants of the k axiom.

$$\begin{array}{lll}
\mathsf{k}_1\colon \Box(A \supset B) \supset (\Box A \supset \Box B) & \mathsf{k}_3\colon \Diamond(A \vee B) \supset (\Diamond A \vee \Diamond B) & \mathsf{k}_5\colon \Diamond\bot \supset \bot\\
\mathsf{k}_2\colon \Box(A \supset B) \supset (\Diamond A \supset \Diamond B) & \mathsf{k}_4\colon (\Diamond A \supset \Box B) \supset \Box(A \supset B) &
\end{array} \tag{1}$$

This logic IK was first studied in [18] and [16], and then was investigated in detail in [20], particularly its standard Kripke semantics based on birelational models.

In this paper, we will also examine the intuitionistic variants of the axioms d, t, b, 4, and 5 that are shown on the right in Figure 1. As in the classical case, they give rise to 15 different distinct logics that can be arranged in a cube, the so-called *S5-cube*. (There are fewer than 32 logics because of redundant sets such as $\{\mathsf{t}, \mathsf{5}\}$ and $\{\mathsf{b}, \mathsf{4}\}$ that both yield the logic IS5.) The intuitionistic variant of the cube is shown on the left in Figure 1.

For a given set $\mathsf{X} \subseteq \{\mathsf{t}, \mathsf{d}, \mathsf{4}, \mathsf{b}, \mathsf{5}\}$, we write IK+X for the logic that is obtained from IK by adding the axioms in X. A formula A is said to be X-*valid* iff it is a theorem of IK+X.[1] In addition, we define the *45-closure* of X, denoted by $\hat{\mathsf{X}}$, as follows:

$$\hat{\mathsf{X}} = \begin{cases} \mathsf{X}{+}\mathsf{4} & \text{if } \{\mathsf{b}, \mathsf{5}\} \subseteq \mathsf{X} \text{ or if } \{\mathsf{t}, \mathsf{5}\} \subseteq \mathsf{X}\\ \mathsf{X}{+}\mathsf{5} & \text{if } \{\mathsf{b}, \mathsf{4}\} \subseteq \mathsf{X}\\ \mathsf{X} & \text{otherwise} \end{cases}$$

If $\mathsf{X} = \hat{\mathsf{X}}$ we also say that X is *45-closed*. In this case we have that whenever the 4 axiom (or the 5 axiom) is derivable in IK+X, then 4 (or 5 resp.) is already contained in X. Every logic in the cube in Figure 1 can be defined by at least one 45-closed set of axioms [4].

3 Intuitionistic Modal Logic in Nested Sequents

This section is a summary of the nested sequent system NIK from [21]. The standard formulation of NIK is based closely on the classical system KN [7, 4]. A *nested sequent* is a finite tree where each node contains a multiset of formulas. In the classical case, this tree is then endowed with an *interpretation* where, at each node, the interpretation of each child subtree is *boxed* (using □) and considered to be disjunctively related to that of the other child subtrees and to the formulas at the node. This interpretation is purely symmetric. To move to the intuitionistic case, we need to introduce an essential asymmetry between the *input* (*i.e.*, the *left*) formulas, which constitute the hypotheses, and the singleton *output* (or the *right*) that constitutes the conclusion. Exactly one of the formulas in the tree will

[1] We slightly abuse the term *valid* as we do not refer to semantics in this paper.

therefore be annotated with a special mark, depicted with a superscript °, to signify that it is the output; all other formulas will then be interpreted as inputs.

To be concrete, we will present nested sequents in terms of a grammar of *input sequents* (written Λ) where the output formula does not occur, and *full sequents* (written Γ) where the output formula does occur. When the distinction between input and full sequents is not essential, we will use Δ to stand for either case. The relationship between parent and child in the tree will be represented using *bracketing* ([ ]).

$$\Lambda ::= \emptyset \mid A, \Lambda \mid [\Lambda_1], \Lambda_2 \qquad \Gamma ::= \Lambda, A^\circ \mid \Lambda, [\Gamma] \qquad \Delta ::= \Lambda \mid \Gamma$$

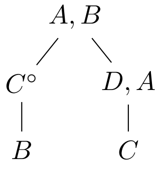
Every full sequent Γ therefore has the shape $\Lambda_1, [\Lambda_2, \cdots [\Lambda_n, A^\circ] \cdots]$. As usual, we consider sequents to be identical up to a reordering of the comma-separated elements. Observe that removing the output formula from a full sequent yields an input sequent. We write Λ, Δ to stand for the concatenation of Λ and Δ, given inductively by $\emptyset, \Delta = \Delta$; $(A, \Lambda), \Delta = A, (\Lambda, \Delta)$; and $([\Lambda_1], \Lambda_2), \Delta = [\Lambda_1], (\Lambda_2, \Delta)$.

▶ **Definition 3.1** (Meaning). The *meaning* of a NIK sequent Δ is a formula, written $\mathrm{fm}(\Delta)$, that obeys the following equations.

$$\mathrm{fm}(\emptyset) = \top \qquad \mathrm{fm}(A, \Lambda) = A \wedge \mathrm{fm}(\Lambda) \qquad \mathrm{fm}([\Lambda_1], \Lambda_2) = \Diamond\mathrm{fm}(\Lambda_1) \wedge \mathrm{fm}(\Lambda_2)$$
$$\mathrm{fm}(\Lambda, A^\circ) = \mathrm{fm}(\Lambda) \supset A \qquad \mathrm{fm}(\Lambda, [\Gamma]) = \mathrm{fm}(\Lambda) \supset \Box\mathrm{fm}(\Gamma)$$

We assume that any occurrences of $A \wedge \top$ and $\top \supset A$ in the meaning are simplified to A.

▶ **Example 3.2.** Consider the following full sequent: $\Gamma = A, B, [C^\circ, [B]], [D, A, [C]]$. It is considered identical to $A, B, [D, A, [C]], [[B], C^\circ]$ and represents this tree:



We also have $\mathrm{fm}(\Gamma) = A \wedge B \wedge \Diamond(D \wedge A \wedge \Diamond C) \supset \Box(\Diamond B \supset C)$.

The inference rules for nested sequents will operate on subtrees of such sequents. To identify such subtrees, we use the notions of *contexts* and *substitutions*.

▶ **Definition 3.3** (Context). An *n-holed context* is like a sequent but contains n pairwise distinct numbered *holes* of the form $\{\,\}_i$ (for $i \in 1..n$) wherever an input formula may otherwise occur. We depict such a context as $\Delta\{\,\}_1 \cdots \{\,\}_n$. Given n sequents $\Delta_1, \ldots, \Delta_n$ (called the *arguments*), we write $\Delta\{\Delta_1\} \cdots \{\Delta_n\}$, called a *substitution*, to stand for the sequent where the hole $\{\,\}_i$ in $\Delta\{\,\}_1 \cdots \{\,\}_n$ has been replaced by Δ_i (for $i \in 1..n$), assuming that the result is well-formed, *i.e.*, there is at most one °-annotated formula. Note that if $\Delta_i = \emptyset$ we simply remove the hole $\{\,\}_i$. A *full context* is a context of the form $\Gamma\{\,\}_1 \cdots \{\,\}_n$, which means that there is an output formula in $\Gamma\{\emptyset\}_1 \cdots \{\emptyset\}_n$. Thus, all the arguments to this context must be input sequents. On the other hand, an *input context* is of the form $\Lambda\{\,\}_1 \cdots \{\,\}_n$ and contains only input formulas, so when it is used to build a sequent at most one of its arguments can itself be a full sequent.

In the rest of this paper, we will omit the hole index subscripts (except when there is some ambiguity) to keep the notation light. Note that a 0-holed context is the same as a sequent. Given a 1-holed context that contains no output formulas, *i.e.*, of the form $\Lambda\{\,\}$, it

NIK

$$\mathsf{id}\,\frac{}{\Lambda\{a, a^\circ\}} \qquad \wedge_{\mathsf{R}}\,\frac{\Lambda\{A^\circ\} \quad \Lambda\{B^\circ\}}{\Lambda\{A \wedge B^\circ\}} \qquad \wedge_{\mathsf{L}}\,\frac{\Gamma\{A, B\}}{\Gamma\{A \wedge B\}} \qquad \top_{\mathsf{R}}\,\frac{}{\Lambda\{\top^\circ\}}$$

$$\vee_{\mathsf{R1}}\,\frac{\Lambda\{A^\circ\}}{\Lambda\{A \vee B^\circ\}} \qquad \vee_{\mathsf{R2}}\,\frac{\Lambda\{B^\circ\}}{\Lambda\{A \vee B^\circ\}} \qquad \vee_{\mathsf{L}}\,\frac{\Gamma\{A\} \quad \Gamma\{B\}}{\Gamma\{A \vee B\}} \qquad \bot_{\mathsf{L}}\,\frac{}{\Gamma\{\bot\}}$$

$$\supset_{\mathsf{R}}\,\frac{\Lambda\{A, B^\circ\}}{\Lambda\{A \supset B^\circ\}} \qquad \supset_{\mathsf{L}}\,\frac{\Gamma^*\{A \supset B, A^\circ\} \quad \Gamma\{B\}}{\Gamma\{A \supset B\}}$$

$$\Diamond_{\mathsf{Rk}}\,\frac{\Lambda_1\{[\Lambda_2, A^\circ]\}}{\Lambda_1\{[\Lambda_2], \Diamond A^\circ\}} \qquad \Diamond_{\mathsf{L}}\,\frac{\Gamma\{[A]\}}{\Gamma\{\Diamond A\}} \qquad \Box_{\mathsf{R}}\,\frac{\Lambda\{[A^\circ]\}}{\Lambda\{\Box A^\circ\}} \qquad \Box_{\mathsf{Lk}}\,\frac{\Delta_1\{\Box A, [A, \Delta_2]\}}{\Delta_1\{\Box A, [\Delta_2]\}}$$

X

$$\Diamond_{\mathsf{Rt}}\,\frac{\Lambda\{A^\circ\}}{\Lambda\{\Diamond A^\circ\}} \qquad \Box_{\mathsf{Lt}}\,\frac{\Gamma\{\Box A, A\}}{\Gamma\{\Box A\}} \qquad \Diamond_{\mathsf{R4}}\,\frac{\Lambda_1\{[\Lambda_2, \Diamond A^\circ]\}}{\Lambda_1\{[\Lambda_2], \Diamond A^\circ\}} \qquad \Box_{\mathsf{L4}}\,\frac{\Delta_1\{\Box A, [\Box A, \Delta_2]\}}{\Delta_1\{\Box A, [\Delta_2]\}}$$

$$\Diamond_{\mathsf{Rd}}\,\frac{\Lambda\{[A^\circ]\}}{\Lambda\{\Diamond A^\circ\}} \qquad \Box_{\mathsf{Ld}}\,\frac{\Gamma\{\Box A, [A]\}}{\Gamma\{\Box A\}} \qquad \Diamond_{\mathsf{Rb}}\,\frac{\Lambda_1\{[\Lambda_2], A^\circ\}}{\Lambda_1\{[\Lambda_2, \Diamond A^\circ]\}} \qquad \Box_{\mathsf{Lb}}\,\frac{\Delta_1\{[\Box A, \Delta_2], A\}}{\Delta_1\{[\Box A, \Delta_2]\}}$$

$$\Diamond_{\mathsf{R5}}\,\frac{\Lambda\{\emptyset\}\{\Diamond A^\circ\}}{\Lambda\{\Diamond A^\circ\}\{\emptyset\}}\ \mathrm{dp}(\Lambda\{\ \}\{\emptyset\}) > 0 \qquad \Box_{\mathsf{L5}}\,\frac{\Gamma\{\Box A\}\{\Box A\}}{\Gamma\{\Box A\}\{\emptyset\}}\ \mathrm{dp}(\Gamma\{\ \}\{\emptyset\}) > 0$$

Figure 2 The NIK+X family of nested sequent systems for intuitionistic modal logics.

is permissible to replace the hole with a full sequent Γ, in which case the substitution $\Lambda\{\Gamma\}$ is also a full sequent. If the context contains an output formula, however, then this formula must be removed before such a substitution is syntactically well-formed.

▶ **Definition 3.4** (Output Deletion). We write $\Delta^*\{\ \}_1 \cdots \{\ \}_n$ for the result of deleting any output formulas from an n-holed context $\Delta\{\ \}_1 \cdots \{\ \}_n$.

▶ **Example 3.5.** Consider $\Lambda\{\ \} = [[B, C], \{\ \}], C$; $\Gamma_1\{\ \} = C, [\{\ \}, [B, C^\circ]]$; and $\Gamma_2 = A, [B^\circ]$. Then, $\Lambda\{\Gamma_2\} = C, [[B, C], A, [B^\circ]]$ and $\Gamma_1\{\Lambda\{\emptyset\}\} = C, [[[B, C]], C, [B, C^\circ]]$. $\Gamma_1\{\Gamma_2\}$ is not well-formed because it would contain both C° and B°, but $\Gamma_1^*\{\Gamma_2\} = C, [[B], A, [B^\circ]]$.

We now have enough ingredients to define the inference rules for NIK, which are displayed in Figure 2. The rules in the upper box are common to every logic in the modal cube and so we call just this core system NIK. For every collection of axioms $\mathsf{X} \subseteq \{\mathsf{t}, \mathsf{d}, \mathsf{4}, \mathsf{b}, \mathsf{5}\}$, we define the system NIK+X by adding to NIK the rules $\Diamond_{\mathsf{Rx}}$ and $\Box_{\mathsf{Lx}}$ for every $\mathsf{x} \in \mathsf{X}$. Note that in the rules $\Box_{\mathsf{Lk}}$, $\Box_{\mathsf{L4}}$, $\Box_{\mathsf{Lb}}$ exactly one of the $\Delta_1\{\ \}$ and Δ_2 is a full sequent (context), and the other is an input sequent (context), as only one of them can contain the unique output formula.

The $\Diamond_{\mathsf{R5}}$ and $\Box_{\mathsf{L5}}$ rules have a side condition on the *depth* of the occurrence of the principal formula, which must not be in the root of the tree representation of the sequent. This is a direct consequence of the fact that the 5 axiom implies that $\Diamond \cdots \Diamond A \supset \Box\Diamond A$, so a bracketed $\Diamond A^\circ$ in the conclusion can be derived from any $\Diamond A^\circ$ under a prefix of n $\Diamond$s, which can then be moved into any other bracket at depth n in the premise using $\Diamond_{\mathsf{Rk}}$.

▶ **Definition 3.6** (Depth). The *depth* of a 1-holed context $\Delta\{\ \}$, written $\mathrm{dp}(\Delta\{\ \})$, is given inductively by $\mathrm{dp}(\{\ \}) = 0$; $\mathrm{dp}(\Delta_1, \Delta_2\{\ \}) = \mathrm{dp}(\Delta_2\{\ \})$; $\mathrm{dp}(\Delta_1, [\Delta_2\{\ \}]) = 1 + \mathrm{dp}(\Delta_2\{\ \})$.

▶ **Example 3.7.** We give as an example the proof of $\mathsf{k}_4 : (\Diamond p \supset \Box n) \supset \Box(p \supset n)$ in NIK.

$$\supset_{\mathsf{R}}\,\dfrac{\Box_{\mathsf{R}}\,\dfrac{\supset_{\mathsf{R}}\,\dfrac{\supset_{\mathsf{L}}\,\dfrac{\Diamond_{\mathsf{R}}\,\dfrac{\mathsf{id}\,\dfrac{}{\Diamond p \supset \Box n, [p^\circ, p]}}{\Diamond p \supset \Box n, \Diamond p^\circ, [p]} \qquad \Box_{\mathsf{L}}\,\dfrac{\mathsf{id}\,\dfrac{}{\Box n, [p, n, n^\circ]}}{\Box n, [p, n^\circ]}}{\Diamond p \supset \Box n, [p, n^\circ]}\ (\dagger)}{\Diamond p \supset \Box n, [p \supset n^\circ]}}{\Diamond p \supset \Box n, \Box(p \supset n)^\circ}}{(\Diamond p \supset \Box n) \supset \Box(p \supset n)^\circ}$$

Observe how n° is deleted from the first premise of (†).

The following theorem summarizes the main results of [21].

▶ **Theorem 3.8.** *Let* $\mathsf{X} \subseteq \{\mathsf{t}, \mathsf{d}, 4, \mathsf{b}, 5\}$ *be 45-closed and let* Γ *be a sequent. The following are equivalent:*

1. $\mathrm{fm}(\Gamma)$ *is* X*-valid.*
2. Γ *is provable in* $\mathsf{NIK}{+}\mathsf{X}{+}\mathsf{cut}$*, where* cut *is:*
3. Γ *is provable in* $\mathsf{NIK}{+}\mathsf{X}$.

$$\boxed{\mathsf{cut}\ \frac{\Gamma^*\{A^\circ\} \quad \Gamma\{A\}}{\Gamma\{\emptyset\}}}$$

Proof. See [21, Theorems 4.1, 5.1, 5.3, and 6.7]. 45-closure is only needed for cut-elimination and cut-free completeness. It is not necessary for soundness and completeness with cut. ◀

4 Focused Nested Sequents

In the previous section, input and output formulas were differentiated using annotations, but without any particular restrictions on which kinds of formulas may receive which annotations. It turns out that certain connectives are endowed with inherent affinities for one or the other annotation. For instance, $\Diamond$-formulas in the output tend to remain as side formulas until the sequent has the adequate bracketing structure, but input $\Diamond$-formulas can be decomposed eagerly since $\Diamond_\mathsf{L}$ is an invertible rule. For example, the sequent $\Diamond a, \Diamond a^\circ$ can only be proved by applying the $\Diamond_\mathsf{L}$ rule below $\Diamond_{\mathsf{Rk}}$. In the terminology of polarities and focusing, $\Diamond$-formulas are *synchronous* or *positive*.

It turns out that we can classify every formula – not just $\Diamond$ – into either a *positive formula*, whose right rules are non-invertible, or a *negative formula*, whose left rules are non-invertible. For nearly every kind of formula, this classification is canonically determined; the exceptions are the atoms, where the choice of polarity is free as long as each atom is assigned exactly one polarity, and the $\wedge$ and $\top$ connectives, which are ambiguous in the sense that it is possible to design inference rules for them that give them a positive or a negative interpretation. Following [7, 11], we divide $\wedge$ and $\top$ into their polarized incarnations as separate connectives; $\wedge$ into its positive and negative polarizations, $\overset{+}{\wedge}$ and $\overset{-}{\wedge}$, and $\top$ into $\overset{+}{\top}$ and $\overset{-}{\top}$. Formulas are therefore divided into the positive (written with P, Q) and negative (written with N, M) classes as follows.

$$P, Q ::= L \mid P \overset{+}{\wedge} Q \mid \overset{+}{\top} \mid P \vee Q \mid \bot \mid \Diamond P \qquad\qquad L ::= p \mid {\downarrow} N$$
$$N, M ::= R \mid M \overset{-}{\wedge} N \mid \overset{-}{\top} \mid P \supset N \mid \Box N \qquad\qquad R ::= n \mid {\uparrow} P$$

We write L for particular positive formulas that we call *left-neutral formulas*, and R for particular negative formulas that we call *right-neutral formulas*. They can be atoms or built from the polarity *shifts* $\downarrow$ and $\uparrow$, which are used to move between the two polarized classes.

Polarized sequents are similar to NIK sequents, but instead of using annotations, we force input formulas to be positive and output formulas to be negative. The resulting grammar for *polarized input sequents* (written Ω) and *polarized full sequents* (written Σ) is then:

$$\Omega ::= \emptyset \mid P, \Omega \mid [\Omega_1], \Omega_2 \qquad\qquad \Sigma ::= \Omega, N \mid \Omega, [\Sigma] \qquad\qquad \Theta ::= \Omega \mid \Sigma$$

Observe that in any polarized full sequent there is always exactly one negative formula. In building the focused proof system, we will largely confine ourselves to *neutral input sequents* (written Λ) and *neutral full sequents* (written Γ), which are those subclasses of polarized

FoNIK

$$\mathsf{id}^{\mathsf f}_{\mathsf R}\,\dfrac{}{\Gamma\{p,\langle p\rangle\}} \qquad \wedge{+}^{\mathsf f}_{\mathsf R}\,\dfrac{\Gamma\{\langle P\rangle\}\quad\Gamma\{\langle Q\rangle\}}{\Gamma\{\langle P\overset{+}{\wedge}Q\rangle\}} \qquad \top{+}^{\mathsf f}_{\mathsf R}\,\dfrac{}{\Gamma\{\langle\overset{+}{\top}\rangle\}} \qquad \vee^{\mathsf f}_{\mathsf{R1}}\,\dfrac{\Gamma\{\langle P\rangle\}}{\Gamma\{\langle P\vee Q\rangle\}} \qquad \vee^{\mathsf f}_{\mathsf{R2}}\,\dfrac{\Gamma\{\langle Q\rangle\}}{\Gamma\{\langle P\vee Q\rangle\}}$$

$$\mathsf{id}^{\mathsf f}_{\mathsf L}\,\dfrac{}{\Lambda\{n,\langle n\rangle\}} \qquad \wedge{-}^{\mathsf f}_{\mathsf{L1}}\,\dfrac{\Gamma\{\langle M\rangle\}}{\Gamma\{\langle M\overset{-}{\wedge}N\rangle\}} \qquad \wedge{-}^{\mathsf f}_{\mathsf{L2}}\,\dfrac{\Gamma\{\langle N\rangle\}}{\Gamma\{\langle M\overset{-}{\wedge}N\rangle\}} \qquad \supset^{\mathsf f}_{\mathsf L}\,\dfrac{\Gamma\{\langle P\rangle\}\quad\Gamma\{\langle N\rangle\}}{\Gamma\{\langle P\supset N\rangle\}}$$

$$\Diamond^{\mathsf f}_{\mathsf{Rk}}\,\dfrac{\Delta_1\{[\Delta_2,\langle P\rangle]\}}{\Delta_1\{[\Delta_2],\langle\Diamond P\rangle\}} \qquad \Box^{\mathsf f}_{\mathsf{Lk}}\,\dfrac{\Delta_1\{[\Delta_2,\langle N\rangle]\}}{\Delta_1\{[\Delta_2],\langle\Box N\rangle\}}$$

$$\uparrow^{\mathsf f}_{\mathsf R}\,\dfrac{\Lambda\{\uparrow P,\langle P\rangle\}}{\Lambda\{\uparrow P\}} \qquad \uparrow^{\mathsf f}_{\mathsf L}\,\dfrac{\Gamma\{P\}}{\Gamma\{\langle\uparrow P\rangle\}} \qquad \downarrow^{\mathsf f}_{\mathsf L}\,\dfrac{\Gamma\{\downarrow N,\langle N\rangle\}}{\Gamma\{\downarrow N\}} \qquad \downarrow^{\mathsf f}_{\mathsf R}\,\dfrac{\Gamma^*\{N\}}{\Gamma\{\langle\downarrow N\rangle\}}$$

$$\wedge{+}^{\mathsf f}_{\mathsf L}\,\dfrac{\Sigma\{P,Q\}}{\Sigma\{P\overset{+}{\wedge}Q\}} \qquad \top{+}^{\mathsf f}_{\mathsf L}\,\dfrac{\Sigma\{\emptyset\}}{\Sigma\{\overset{+}{\top}\}} \qquad \vee^{\mathsf f}_{\mathsf L}\,\dfrac{\Sigma\{P\}\quad\Sigma\{Q\}}{\Sigma\{P\vee Q\}} \qquad \bot^{\mathsf f}_{\mathsf L}\,\dfrac{}{\Sigma\{\bot\}} \qquad \Diamond^{\mathsf f}_{\mathsf L}\,\dfrac{\Sigma\{[P]\}}{\Sigma\{\Diamond P\}}$$

$$\wedge{-}^{\mathsf f}_{\mathsf R}\,\dfrac{\Omega\{M\}\quad\Omega\{N\}}{\Omega\{M\overset{-}{\wedge}N\}} \qquad \top{-}^{\mathsf f}_{\mathsf R}\,\dfrac{}{\Omega\{\overset{-}{\top}\}} \qquad \supset^{\mathsf f}_{\mathsf R}\,\dfrac{\Omega\{P,N\}}{\Omega\{P\supset N\}} \qquad \Box^{\mathsf f}_{\mathsf R}\,\dfrac{\Omega\{[N]\}}{\Omega\{\Box N\}}$$

$\mathsf{X}^{\mathsf f}$

$$\Diamond^{\mathsf f}_{\mathsf{Rt}}\,\dfrac{\Gamma\{\langle P\rangle\}}{\Gamma\{\langle\Diamond P\rangle\}} \qquad \Box^{\mathsf f}_{\mathsf{Lt}}\,\dfrac{\Gamma\{\langle N\rangle\}}{\Gamma\{\langle\Box N\rangle\}} \qquad \Diamond^{\mathsf f}_{\mathsf{R4}}\,\dfrac{\Delta_1\{[\Delta_2,\langle\Diamond P\rangle]\}}{\Delta_1\{[\Delta_2],\langle\Diamond P\rangle\}} \qquad \Box^{\mathsf f}_{\mathsf{L4}}\,\dfrac{\Delta_1\{[\Delta_2,\langle\Box N\rangle]\}}{\Delta_1\{[\Delta_2],\langle\Box N\rangle\}}$$

$$\Diamond^{\mathsf f}_{\mathsf{Rd}}\,\dfrac{\Gamma\{[\langle P\rangle]\}}{\Gamma\{\langle\Diamond P\rangle\}} \qquad \Box^{\mathsf f}_{\mathsf{Ld}}\,\dfrac{\Gamma\{[\langle N\rangle]\}}{\Gamma\{\langle\Box N\rangle\}} \qquad \Diamond^{\mathsf f}_{\mathsf{Rb}}\,\dfrac{\Delta_1\{[\Delta_2],\langle P\rangle\}}{\Delta_1\{[\Delta_2,\langle\Diamond P\rangle]\}} \qquad \Box^{\mathsf f}_{\mathsf{Lb}}\,\dfrac{\Delta_1\{[\Delta_2],\langle N\rangle\}}{\Delta_1\{[\Delta_2,\langle\Box N\rangle]\}}$$

$$\Diamond^{\mathsf f}_{\mathsf{R5}}\,\dfrac{\Gamma\{\emptyset\}\{\langle\Diamond P\rangle\}}{\Gamma\{\langle\Diamond P\rangle\}\{\emptyset\}}\ \mathrm{dp}(\Gamma\{\ \}\{\emptyset\})>1 \qquad \Box^{\mathsf f}_{\mathsf{L5}}\,\dfrac{\Gamma\{\emptyset\}\{\langle\Box N\rangle\}}{\Gamma\{\langle\Box N\rangle\}\{\emptyset\}}\ \mathrm{dp}(\Gamma\{\ \}\{\emptyset\})>1$$

Figure 3 The FoNIK+X$^{\mathsf f}$ family, focused versions of NIK+X from Figure 2.

input sequents and polarized full sequents that are built up of neutral formulas. In other words, they have the following grammar.

$$\Lambda ::= \emptyset \mid L,\Lambda \mid [\Lambda_1],\Lambda_2 \qquad\qquad \Gamma ::= \Lambda, R \mid \Lambda,[\Gamma] \qquad\qquad \Delta ::= \Lambda \mid \Gamma$$

Let us now give the meanings of these polarized sequents.

▶ **Definition 4.1** (Depolarization). Every polarized formula P or N is related to an unpolarized formula by a *depolarization* map $\lfloor\ \rfloor$ with the following inductive definition.

$$\lfloor p\rfloor = p \quad \lfloor\downarrow N\rfloor = \lfloor N\rfloor \qquad \lfloor n\rfloor = n \quad \lfloor\uparrow P\rfloor = \lfloor P\rfloor$$

$$\lfloor P\overset{+}{\wedge}Q\rfloor = \lfloor P\rfloor\wedge\lfloor Q\rfloor \quad \lfloor\overset{+}{\top}\rfloor = \top \quad \lfloor P\vee Q\rfloor = \lfloor P\rfloor\vee\lfloor Q\rfloor \quad \lfloor\bot\rfloor = \bot \quad \lfloor\Diamond P\rfloor = \Diamond\lfloor P\rfloor$$

$$\lfloor M\overset{-}{\wedge}N\rfloor = \lfloor M\rfloor\wedge\lfloor N\rfloor \quad \lfloor\overset{-}{\top}\rfloor = \top \quad \lfloor P\supset N\rfloor = \lfloor P\rfloor\supset\lfloor N\rfloor \quad \lfloor\Box N\rfloor = \Box\lfloor N\rfloor$$

We say that P or N is X-*valid* iff $\lfloor P\rfloor$ or $\lfloor N\rfloor$ is X-valid, respectively.

▶ **Definition 4.2** (Meaning). The *meaning* of a polarized sequent Θ, written $\mathrm{fm}(\Theta)$, is a positive or a negative formula (respectively) obeying:

$$\mathrm{fm}(\emptyset) = \overset{+}{\top} \qquad \mathrm{fm}(P,\Omega) = P\overset{+}{\wedge}\mathrm{fm}(\Omega) \qquad \mathrm{fm}([\Omega_1],\Omega_2) = \Diamond\mathrm{fm}(\Omega_1)\overset{+}{\wedge}\mathrm{fm}(\Omega_2)$$

$$\mathrm{fm}(\Omega,N) = \mathrm{fm}(\Omega)\supset N \qquad \mathrm{fm}(\Omega,[\Sigma]) = \mathrm{fm}(\Omega)\supset\Box\mathrm{fm}(\Sigma)$$

▶ **Definition 4.3** (Polarized Context). An *n-holed polarized context* is like a polarized sequent but contains n pairwise distinct numbered *holes* of the form $\{\ \}_i$ (for $i\in 1..n$) wherever a positive formula may otherwise occur. We depict such a context as $\Theta\{\ \}_1\cdots\{\ \}_n$. Given n polarized sequents $\Theta_1,\ldots,\Theta_n$ (the *arguments*), we write the *substitution* $\Theta\{\Theta_1\}\cdots\{\Theta_n\}$ to mean the sequent where the hole $\{\ \}_i$ in $\Theta\{\ \}_1\cdots\{\ \}_n$ is replaced by Θ_i (or removed if $\Theta_i=\emptyset$), for $i\in 1..n$, assuming that the result is well-formed, *i.e.*, that there is at most one negative formula in the result. We write $\Theta^*\{\ \}_1 \cdots \{\ \}_n$ for the context formed by deleting any negative formula from $\Theta\{\ \}_1 \cdots \{\ \}_n$.[2]

The inference system for polarized sequents will be *focused* [1]. A focused proof is a proof where the *decision* to apply a non-invertible rule to a neutral formula must be explicitly taken, and this decision commits the proof to retaining focus on its transitive subformulas until there is a polarity change. This focusing protocol drastically reduces the space of proofs, since rules can only be applied to the focused formula when one exists. Nevertheless, every derivable polarized sequent has a focused proof, as we will see in Section 6.

▶ **Definition 4.4** (Focused Sequent)**.** A *focused sequent* is like a neutral sequent but contains an additional single occurrence of $\langle P\rangle$ or $\langle N\rangle$ wherever a positive formula may otherwise occur, called its *focus*. We depict such sequents as $\Gamma\{\langle P\rangle\}$ or $\Gamma\{\langle N\rangle\}$ where $\Gamma\{\ \}$ is a neutral context (*i.e.*, $\Gamma\{\emptyset\}$ is a neutral sequent). The meaning of a focused sequent is written by extending $\mathrm{fm}(\ )$, which now obeys $\mathrm{fm}(\Gamma\{\langle P\rangle\}) = \mathrm{fm}(\Gamma^*\{\uparrow P\})$ and $\mathrm{fm}(\Gamma\{\langle N\rangle\}) = \mathrm{fm}(\Gamma\{\downarrow N\})$.

The inference rules of the family of focused sequent systems $\mathsf{FoNIK{+}X^f}$ are given in Figure 3. Like with $\mathsf{NIK{+}X}$ earlier, for any $\mathsf{X} \subseteq \{\mathsf{t}, \mathsf{d}, \mathsf{4}, \mathsf{b}, \mathsf{5}\}$, we define $\mathsf{FoNIK{+}X^f}$ to be the system FoNIK, consisting of the rules in the upper section of Figure 3, extended with $\Box^{\mathsf f}_{\mathsf{Lx}}$ and $\Diamond^{\mathsf f}_{\mathsf{Rx}}$ (for each $\mathsf{x} \in \mathsf{X}$) in the lower section of the figure.

A focused proof of a neutral sequent begins – reading from conclusion upwards – with a neutral end-sequent, to which only the two rules $\downarrow^{\mathsf f}_{\mathsf L}$ and $\uparrow^{\mathsf f}_{\mathsf R}$ may be applied. In each case a neutral shifted formula is selected for focus, at which point the proof enters the *focused* phase, which persists until the focus again becomes neutral. At this point, the proof either finishes with $\mathsf{id}^{\mathsf f}_{\mathsf R}$ or $\mathsf{id}^{\mathsf f}_{\mathsf L}$ if the focus is atomic, or it enters the *active* phase using the rules $\uparrow^{\mathsf f}_{\mathsf L}$ or $\downarrow^{\mathsf f}_{\mathsf R}$. Note that, because the $\downarrow^{\mathsf f}_{\mathsf R}$ rule introduces a negative formula to the premise sequent, any other existing negative formulas must be deleted. In the active phase, positive and negative formulas are decomposed, in an arbitrary order, using left and right rules respectively, until eventually the sequent becomes neutral again.

▶ **Example 4.5.** Let $R = \uparrow\downarrow(\downarrow(\Diamond p \supset \Box n) \supset \Box(p \supset n))$ which is a right-neutral polarized form of $\mathsf{k_4}$ (see (1)) with a minimal number of shifts. Below is the derivation of R in FoNIK, and therefore the focused version of the derivation in Example 3.7:

$$
\uparrow^{\mathsf f}_{\mathsf R}\,\dfrac{\downarrow^{\mathsf f}_{\mathsf R}\,\dfrac{\supset^{\mathsf f}_{\mathsf R}\,\dfrac{\Box^{\mathsf f}_{\mathsf R}\,\dfrac{\supset^{\mathsf f}_{\mathsf R}\,\dfrac{\downarrow^{\mathsf f}_{\mathsf L}\,\dfrac{\supset^{\mathsf f}_{\mathsf L}\,\dfrac{\Diamond^{\mathsf f}_{\mathsf{Rk}}\,\dfrac{\mathsf{id}^{\mathsf f}_{\mathsf R}\,\dfrac{}{R, \downarrow(\Diamond p \supset \Box n), [\langle p\rangle, p, n]}}{R, \downarrow(\Diamond p \supset \Box n), \langle\Diamond p\rangle, [p, n]} \qquad \Box^{\mathsf f}_{\mathsf{Lk}}\,\dfrac{\mathsf{id}^{\mathsf f}_{\mathsf L}\,\dfrac{}{R, \downarrow(\Diamond p \supset \Box n), [\langle n\rangle, p, n]}}{R, \downarrow(\Diamond p \supset \Box n), \langle\Box n\rangle, [p, n]}}{R, \downarrow(\Diamond p \supset \Box n), \langle\Diamond p \supset \Box n\rangle, [p, n]}\;(\dagger)}{R, \downarrow(\Diamond p \supset \Box n), [p, n]}}{R, \downarrow(\Diamond p \supset \Box n), [p \supset n]}}{R, \downarrow(\Diamond p \supset \Box n), \Box(p \supset n)}}{R, \downarrow(\Diamond p \supset \Box n) \supset \Box(p \supset n)}}{R, \langle\downarrow(\downarrow(\Diamond p \supset \Box n) \supset \Box(p \supset n))\rangle}}{R}
$$

Observe that the instance of $\downarrow^{\mathsf f}_{\mathsf L}$ marked (†) cannot be applied any lower in the derivation, since its conclusion would not then be neutral.

▶ **Lemma 4.6** (Soundness)**.** *Let* $\mathsf{X} \subseteq \{\mathsf{t}, \mathsf{d}, \mathsf{4}, \mathsf{b}, \mathsf{5}\}$. *If* Σ *is provable in* $\mathsf{FoNIK{+}X^f}$ *then it is* X*-valid.*

[2] We reuse the right-deletion notation from Definition 3.4 in the polarized case since the concepts are similar, replacing "output formula" with "negative formula".

Proof. The proof makes use of the following two inference rules

$$\mathsf{weak}\,\frac{\Gamma\{\emptyset\}}{\Gamma\{\Lambda\}} \qquad \text{and} \qquad \mathsf{cont}\,\frac{\Gamma\{\Lambda,\Lambda\}}{\Gamma\{\Lambda\}}$$

defined on unpolarized sequents, and the fact that they are admissible for NIK+X (see [21, Lemma 6.4]). Now, any polarized sequent Σ can be transformed into an unpolarized sequent $\lfloor\Sigma\rfloor$ with the same meaning by replacing every formula P in Σ by $\lfloor P\rfloor$ and the unique formula N in Σ by $\lfloor N\rfloor^{\circ}$, and similarly for contexts $\Sigma\{\ \}$. Then we can define $\lfloor\Sigma\{\langle N\rangle\}\rfloor = \lfloor\Sigma\rfloor\{\lfloor N\rfloor\}$ and $\lfloor\Sigma\{\langle P\rangle\}\rfloor = \lfloor\Sigma\rfloor^{*}\{\lfloor P\rfloor^{\circ}\}$. Every rule in FoNIK+X$^{\mathsf{f}}$ then either becomes trivial or can be simulated by a derivation consisting of an instance of a rule in NIK+X and an instance of weak, except for $\uparrow^{\mathsf{f}}_{\mathsf{R}}$ and $\downarrow^{\mathsf{f}}_{\mathsf{L}}$, which become instances of cont. Thus, a proof of Σ in FoNIK+X$^{\mathsf{f}}$ is transformed into a proof of $\lfloor\Sigma\rfloor$ in NIK+X + weak + cont. The lemma now follows from admissibility of weak and cont for NIK+X and Theorem 3.8. ◀

5 Synthetic Nested Sequents

Ultimately, we wish to establish the following relation between NIK+X and FoNIK+X$^{\mathsf{f}}$.

▶ **Theorem 5.1** (Soundness and Completeness of FoNIK+X$^{\mathsf{f}}$). *The neutral sequent Γ is derivable in* FoNIK+X$^{\mathsf{f}}$ *if and only if $\lfloor\Gamma\rfloor$ is derivable in* NIK+X*.*

However, directly showing this statement is rather complicated because of the number of rules in FoNIK+X$^{\mathsf{f}}$. The issue is actually worse than it appears since we would like to have the completeness of focusing be a consequence of cut-elimination and identity reduction in FoNIK+X$^{\mathsf{f}}$, following a strategy initially described by Laurent [10] that has turned out to be remarkably versatile [8, 11, 19, 7]. To retrace this meta-theory directly in FoNIK+X$^{\mathsf{f}}$ would require a carefully managed collection of cuts with a lengthy and intricate argument.

Can the system be simplified? Since the boundary rules $\downarrow^{\mathsf{f}}_{\mathsf{L}}$, $\uparrow^{\mathsf{f}}_{\mathsf{R}}$, $\uparrow^{\mathsf{f}}_{\mathsf{L}}$ and $\downarrow^{\mathsf{f}}_{\mathsf{R}}$ are limited to conclusions that are either neutral or focused, we can see a FoNIK+X$^{\mathsf{f}}$ derivation as progressing in large *synthetic* steps where the rest of the rules are elided. In this section, we give a presentation of FoNIK+X$^{\mathsf{f}}$ that formally builds only such synthetic derivations. Importantly, the synthetic system has far fewer rules than FoNIK+X$^{\mathsf{f}}$. In particular, there is no longer a duplication of the modal rules into $\Diamond^{\mathsf{f}}_{\mathsf{R}}$ and $\Box^{\mathsf{f}}_{\mathsf{L}}$ versions. Nevertheless, this system will be sound and complete with respect to both NIK+X and FoNIK+X$^{\mathsf{f}}$, giving us Theorem 5.1 as a corollary.

The basis of the synthetic system is to isolate the subformula relation and generalize it into an inductively defined *substructure relation*, written $\Subset$, that determines, for a given focus, what formulas would be present in the fringe of the focused phase rooted on it. Since only neutral formulas occur at the fringes, these substructures would be made up of neutral formulas. The inductive definition of the subformula relation is given in the uppermost part of Figure 4. When $\Lambda \Subset P$ or $\Gamma \Subset N$, we say that Λ or Γ is, respectively, a *synthetic substructure* of P or N. Intuitively, each substructure defines a particular collection of *disjunctive* choices available in a corresponding focused phase. The focused phase is launched from a neutral sequent by picking a suitable neutral formula for focus, selecting one of its substructures, and then *contextualizing* the substructure using a generalization of focused sequents (Definition 4.4).

▶ **Definition 5.2** (Contextualizing Sequent). A *contextualizing sequent* is like a neutral sequent but contains a single occurrence of a *focus* of the form $\langle\Delta\rangle$ (where Δ is a neutral sequent) where a positive neutral formula may otherwise occur. Such sequents are written as $\Gamma\{\langle\Delta\rangle\}$

SyNIK

$$\frac{}{L \Subset L} \qquad \frac{\Lambda_1 \Subset P \quad \Lambda_2 \Subset Q}{\Lambda_1, \Lambda_2 \Subset P \overset{+}{\wedge} Q} \qquad \frac{}{\emptyset \Subset \overset{+}{\top}} \qquad \frac{\Lambda \Subset P}{\Lambda \Subset P \vee Q} \qquad \frac{\Lambda \Subset Q}{\Lambda \Subset P \vee Q} \qquad \frac{\Lambda \Subset P}{[\Lambda] \Subset \Diamond P}$$

$$\frac{}{R \Subset R} \qquad \frac{\Gamma \Subset M}{\Gamma \Subset M \overset{-}{\wedge} N} \qquad \frac{\Gamma \Subset N}{\Gamma \Subset M \overset{-}{\wedge} N} \qquad \frac{\Lambda \Subset P \quad \Gamma \Subset N}{\Lambda, \Gamma \Subset P \supset N} \qquad \frac{\Gamma \Subset N}{[\Gamma] \Subset \Box N}$$

$$\mathsf{id}^{\langle\rangle}\,\frac{}{\Delta\{a, \langle a \rangle\}} \qquad {\uparrow}_{\mathsf{L}}^{\langle\rangle}\,\frac{\big\{\Gamma\{\Lambda\} : \Lambda \Subset P\big\}}{\Gamma\{\langle {\uparrow} P \rangle\}} \qquad {\uparrow}_{\mathsf{R}}^{\langle\rangle}\,\frac{\Lambda_2 \Subset P \quad \Lambda_1\{{\uparrow} P, \langle \Lambda_2 \rangle\}}{\Lambda_1\{{\uparrow} P\}}$$

$$\mathsf{fin}^{\langle\rangle}\,\frac{}{\Gamma\{\langle \emptyset \rangle\}} \qquad {\downarrow}_{\mathsf{R}}^{\langle\rangle}\,\frac{\big\{\Gamma_1^*\{\Gamma_2\} : \Gamma_2 \Subset N\big\}}{\Gamma_1\{\langle {\downarrow} N \rangle\}} \qquad {\downarrow}_{\mathsf{L}}^{\langle\rangle}\,\frac{\Gamma_2 \Subset N \quad \Gamma_1\{{\downarrow} N, \langle \Gamma_2 \rangle\}}{\Gamma_1\{{\downarrow} N\}}$$

$$\mathsf{k}^{\langle\rangle}\,\frac{\Delta_1\{[\Delta_2, \langle \Delta_3 \rangle]\}}{\Delta_1\{[\Delta_2], \langle [\Delta_3] \rangle\}} \qquad \mathsf{spl}^{\langle\rangle}\,\frac{\Gamma\{\langle \Delta \rangle\} \quad \Gamma\{\langle \Lambda \rangle\}}{\Gamma\{\langle \Delta, \Lambda \rangle\}}\ \#\Delta > 0,\ \#\Lambda > 0$$

$\mathsf{X}^{\langle\rangle}$

$$\mathsf{t}^{\langle\rangle}\,\frac{\Gamma\{\langle \Delta \rangle\}}{\Gamma\{\langle [\Delta] \rangle\}} \quad \mathsf{4}^{\langle\rangle}\,\frac{\Delta_1\{[\Delta_2, \langle [\Delta_3] \rangle]\}}{\Delta_1\{[\Delta_2], \langle [\Delta_3] \rangle\}} \quad \mathsf{d}^{\langle\rangle}\,\frac{\Gamma\{[\langle \Delta \rangle]\}}{\Gamma\{\langle [\Delta] \rangle\}} \quad \mathsf{b}^{\langle\rangle}\,\frac{\Delta_1\{[\Delta_2], \langle \Delta_3 \rangle\}}{\Delta_1\{[\Delta_2, \langle [\Delta_3] \rangle]\}} \quad \mathsf{5}^{\langle\rangle}\,\frac{\Gamma\{\langle [\Delta] \rangle\}\{\emptyset\}}{\Gamma\{\emptyset\}\{\langle [\Delta] \rangle\}}$$

Figure 4 The SyNIK+X$^{\langle\rangle}$ family, synthetic versions of FoNIK+X$^{\mathsf{f}}$ (Figure 3).

where $\Gamma\{\emptyset\}$ is a neutral sequent. The *meaning* of a contextualizing sequent is written using $\mathrm{fm}(\,)$ obeying: $\mathrm{fm}(\Gamma\{\langle \Lambda \rangle\}) = \mathrm{fm}(\Gamma^*\{{\uparrow}\mathrm{fm}(\Lambda)\})$ and $\mathrm{fm}(\Gamma_1\{\langle \Gamma_2 \rangle\}) = \mathrm{fm}(\Gamma_1\{{\downarrow}\mathrm{fm}(\Gamma_2)\})$.

The synthetic system SyNIK will be built using neutral and contextualizing sequents. The rules of SyNIK+X$^{\langle\rangle}$ (for any $\mathsf{X} \subseteq \{\mathsf{t}, \mathsf{d}, \mathsf{4}, \mathsf{b}, \mathsf{5}\}$) are shown in Figure 4. As before for NIK+X and FoNIK+X$^{\mathsf{f}}$, we define SyNIK+X$^{\langle\rangle}$ to be SyNIK extended with $\mathsf{x}^{\langle\rangle}$ for every $\mathsf{x} \in \mathsf{X}$. The ${\downarrow}_{\mathsf{L}}^{\langle\rangle}$ and ${\uparrow}_{\mathsf{R}}^{\langle\rangle}$ rules are similar to the ${\downarrow}_{\mathsf{L}}^{\mathsf{f}}$ and ${\uparrow}_{\mathsf{R}}^{\mathsf{f}}$ rules from FoNIK, except that, instead of granting focus to the P or N (respectively), one of its substructures is selected for contextualization. The contextualization rules consist of the rules $\{\mathsf{spl}^{\langle\rangle}, \mathsf{fin}^{\langle\rangle}, \mathsf{k}^{\langle\rangle}, \mathsf{t}^{\langle\rangle}, \mathsf{4}^{\langle\rangle}, \mathsf{d}^{\langle\rangle}, \mathsf{b}^{\langle\rangle}, \mathsf{5}^{\langle\rangle}\}$ that serve to divide up or move the focus among the premises of the rule. To prevent needless looping, the $\mathsf{spl}^{\langle\rangle}$ rule has a side condition that neither of the foci in the premises is empty; the $\mathsf{fin}^{\langle\rangle}$ rule handles the empty focus case instead. The modal rules require the focus in the conclusion to be bracketed. Observe that there is exactly one modal rule for every modal axiom, unlike FoNIK+X$^{\mathsf{f}}$ that needed both left and right versions. The $\mathsf{5}^{\langle\rangle}$ rule has the usual side condition that $\mathrm{dp}(\Gamma\{\ \}\{\emptyset\}) > 0$.

Once the focus has been reduced to a single formula by the other rules, it must either be atomic or a shifted formula. In the former case, we apply the $\mathsf{id}^{\langle\rangle}$ rule, which is the common form of $\mathsf{id}_{\mathsf{R}}^{\mathsf{f}}$ and $\mathsf{id}_{\mathsf{L}}^{\mathsf{f}}$ from FoNIK. When the focus is a shifted formula, we use ${\uparrow}_{\mathsf{L}}^{\langle\rangle}$ or ${\downarrow}_{\mathsf{R}}^{\langle\rangle}$; in each case, we iterate over the substructures of the principal formula, producing one premise per substructure. When the substructure is a neutral sequent, we need to remove the right-neutral formula from the surrounding context; this only happens in the ${\downarrow}_{\mathsf{R}}^{\langle\rangle}$ rule.

▶ **Example 5.3.** Here is the synthetic version of the derivation in Example 4.5.

$$\dfrac{\mathsf{spl}^{\langle\rangle}\;\dfrac{\mathsf{k}^{\langle\rangle}\;\dfrac{\mathsf{id}^{\langle\rangle}\;\dfrac{}{R, {\downarrow}(\Diamond p \supset \Box n), [\langle p \rangle, p, n]}}{R, {\downarrow}(\Diamond p \supset \Box n), \langle [p] \rangle, [p, n]} \qquad \mathsf{k}^{\langle\rangle}\;\dfrac{\mathsf{id}^{\langle\rangle}\;\dfrac{}{R, {\downarrow}(\Diamond p \supset \Box n), [\langle n \rangle, p, n]}}{R, {\downarrow}(\Diamond p \supset \Box n), \langle [n] \rangle, [p, n]}}{{\downarrow}_{\mathsf{L}}^{\langle\rangle}\;\dfrac{R, {\downarrow}(\Diamond p \supset \Box n), \langle [p], [n] \rangle, [p, n]}{{\downarrow}_{\mathsf{R}}^{\langle\rangle}\;\dfrac{R, {\downarrow}(\Diamond p \supset \Box n), [p, n]}{R, \langle {\downarrow}({\downarrow}(\Diamond p \supset \Box n) \supset \Box(p \supset n)) \rangle}}\,(\ddagger)}}{R}\,{\uparrow}_{\mathsf{R}}^{\langle\rangle}\,(\dagger)$$

The instance $(\dagger)$ of ${\uparrow}_{\mathsf{R}}^{\langle\rangle}$ is applicable as $L = {\downarrow}(\Diamond p \supset \Box n) \supset \Box(p \supset n)$ is left-neutral, so $L \Subset L$. Likewise, the instance $(\ddagger)$ of ${\downarrow}_{\mathsf{L}}^{\langle\rangle}$ is applicable since $[p], [n] \Subset \Diamond p \supset \Box n$.

▶ **Lemma 5.4** (Soundness). *Let* $\mathsf{X} \subseteq \{\mathsf{t}, \mathsf{d}, \mathsf{4}, \mathsf{b}, \mathsf{5}\}$ *and let* Γ *be a neutral sequent. If* Γ *is provable in* $\mathsf{SyNIK}{+}\mathsf{X}^{\langle\rangle}$*, then it is also provable in* $\mathsf{FoNIK}{+}\mathsf{X}^{\mathsf{f}}$*.*

Proof (Sketch). The essential idea is to interpret the SyNIK contextualizing sequent $\Gamma\{\langle\Theta\rangle\}$ as the FoNIK focused sequent $\Gamma\{\langle \mathrm{fm}(\Theta)\rangle\}$. The rules of the former can be simulated by the latter because $\Theta \Subset \mathrm{fm}(\Theta)$. Examples 4.5 and 5.3 illustrates this interpretation. The whole proof then works by induction on the given $\mathsf{SyNIK}{+}\mathsf{X}^{\langle\rangle}$ derivation; the $\downarrow^{\langle\rangle}_{\mathsf{L}}$ and $\uparrow^{\langle\rangle}_{\mathsf{R}}$ rules are simulated by repeating the derivation of the substructure in the FoNIK sequent rather than as a side premise. The $\uparrow^{\langle\rangle}_{\mathsf{L}}$ and $\downarrow^{\langle\rangle}_{\mathsf{R}}$ rules are easily simulated since the active rules of FoNIK are precisely matched by the $\Subset$ inferences. The $\mathsf{x}^{\langle\rangle}$ rules are simulated by $\Diamond^{\mathsf{f}}_{\mathsf{Rx}}$ or $\Box^{\mathsf{f}}_{\mathsf{Lx}}$ rules, respectively depending on whether the focus contains a negative formula or not. Finally, $\mathsf{spl}^{\langle\rangle}$ and $\mathsf{fin}^{\langle\rangle}$ are simulated by $\wedge{+}^{\mathsf{f}}_{\mathsf{R}}$ and $\top{+}^{\mathsf{f}}_{\mathsf{R}}$ respectively. ◀

The less trivial converse of Lemma 5.4 will follow from cut-elimination in the next section.

6 Synthetic Meta-Theory

In this section we will show that $\mathsf{SyNIK}{+}\mathsf{X}^{\langle\rangle}$ extended with a cut rule can simulate NIK+X derivations under a certain interpretation of the annotations. We will then show that the cut rule is admissible in $\mathsf{SyNIK}{+}\mathsf{X}^{\langle\rangle}$, thereby concluding that NIK+X rules under that interpretation are admissible in $\mathsf{SyNIK}{+}\mathsf{X}^{\langle\rangle}$, *i.e.*, $\mathsf{SyNIK}{+}\mathsf{X}^{\langle\rangle}$ is complete with respect to NIK+X. To formulate the cut rule with a minimum of redundancy, we will need to slightly enlarge our notion of contexts and define a pair of *pruning* operations for such contexts.

▶ **Definition 6.1** (Enlarged Contexts and Pruning). In this section, we allow neutral sequents to contain at most one occurrence of a focus $\langle\Delta\rangle$. For a sequent Δ, we write $\Delta^{\cancel{\langle\rangle}}$ to prune its focus if there is one (*i.e.*, if $\Delta = \Delta_1\{\langle\Delta_2\rangle\}$ for some $\Delta_1\{\,\}$, then $\Delta^{\cancel{\langle\rangle}} = \Delta_1\{\emptyset\}$; otherwise $\Delta^{\cancel{\langle\rangle}} = \Delta$). This definition extends straightforwardly to contexts $\Delta\{\,\}$. For a context $\Delta_1\{\,\}$, we write $\Delta_1^{\star}\{\Delta_2\}$ to mean $\Delta_1\{\Delta_2\}$ if Δ_2 is an input sequent, and $\Delta_1^{*}\{\Delta_2\}$ if Δ_2 is a full sequent (see Definition 3.4).

The synthetic cut rule for $\mathsf{SyNIK}{+}\mathsf{X}^{\langle\rangle}$ can then be written concisely as follows:

$$\mathsf{cut}^{\langle\rangle}\ \frac{\Gamma^{\star}\{\Delta\} \quad \Gamma^{\cancel{\langle\rangle}}\{\langle\Delta\rangle\}}{\Gamma\{\emptyset\}}$$

Before we can show that $\mathsf{cut}^{\langle\rangle}$ is admissible, we need to show the admissibility of the structural rules shown in Figure 5 that are used in the cut-elimination proof. Note that the weakening rule weak can be applied only to input contexts and the contraction rule cont only to positive atoms. While the contraction rule could be generalized to any input context, only this instance is needed in the proof. Of course, the notion of contexts and sequents in these structural rules are enlarged in the sense of Definition 6.1.

▶ **Definition 6.2.** The *height* of a derivation $\mathcal{D}$, denoted by $\mathrm{ht}(\mathcal{D})$, is the height of $\mathcal{D}$ when seen as a tree, *i.e.*, the length of the longest branch from the root to a leaf. We say that a rule $\mathsf{r}\,\dfrac{\Gamma_1}{\Gamma_2}$ is *admissible* for a system S if for every proof $\mathcal{D}_1$ of Γ_1 in S there is a proof $\mathcal{D}_2$ of Γ_2 in S. We say that it is *height-preserving admissible* if additionally $\mathrm{ht}(\mathcal{D}_2) \le \mathrm{ht}(\mathcal{D}_1)$.

▶ **Lemma 6.3** (Admissible Rules). *Let* $\mathsf{X} \subseteq \{\mathsf{t}, \mathsf{d}, \mathsf{4}, \mathsf{b}, \mathsf{5}\}$ *be* 45*-closed. The rules* weak*,* cont*, and* $\mathsf{k}^{[]}$ *are height-preserving admissible in* $\mathsf{SyNIK}{+}\mathsf{X}^{\langle\rangle}$*, and for every* $\mathsf{x} \in \mathsf{X}$*, the rule* $\mathsf{x}^{[]}$ *is admissible in* $\mathsf{SyNIK}{+}\mathsf{X}^{\langle\rangle}$*.*

$$
\mathsf{weak}\,\dfrac{\Gamma\{\emptyset\}}{\Gamma\{\Lambda\}}
\qquad
\mathsf{k}^{[\,]}\,\dfrac{\Delta_1\{[\Delta_2],[\Delta_3]\}}{\Delta_1\{[\Delta_2,\Delta_3]\}}
\qquad
\mathsf{d}^{[\,]}\,\dfrac{\Delta_1\{[\,]\}}{\Delta_1\{\emptyset\}}
\qquad
\mathsf{b}^{[\,]}\,\dfrac{\Delta_1\{[\Delta_2,[\Delta_3]]\}}{\Delta_1\{[\Delta_2],\Delta_3\}}
$$

$$
\mathsf{cont}\,\dfrac{\Gamma\{p,p\}}{\Gamma\{p\}}
\qquad
\mathsf{4}^{[\,]}\,\dfrac{\Delta_1\{[\Delta_2],[\Delta_3]\}}{\Delta_1\{[[\Delta_2],\Delta_3]\}}
\qquad
\mathsf{t}^{[\,]}\,\dfrac{\Delta_1\{[\Delta_2]\}}{\Delta_1\{\Delta_2\}}
\qquad
\mathsf{5}^{[\,]}\,\dfrac{\Delta_1\{\emptyset\}\{[\Delta_2]\}}{\Delta_1\{[\Delta_2]\}\{\emptyset\}}\;\mathrm{dp}(\Delta_1\{\,\}\{\emptyset\})>0
$$

Figure 5 Structural rules admissible in $\mathsf{SyNIK}{+}\mathsf{X}^{\langle\rangle}$.

Proof. The first part is by straightforward induction on the height of the derivation. The second part is less straightforward, but also by induction on the height of the derivation, following [4, Lemma 9]. Below we show the translation of one case from the case analysis in [4] into our setting. The others are similar.

$$
\mathsf{4}^{[\,]}\,\dfrac{\mathsf{b}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}}{\Delta_1\{\langle\Delta_2\rangle,[\Delta_3],[\Delta_4]\}}}{\Delta_1\{[\langle[\Delta_2]\rangle,\Delta_3],[\Delta_4]\}}}{\Delta_1\{[[\langle[\Delta_2]\rangle,\Delta_3],\Delta_4]\}}
\quad\rightsquigarrow\quad
\mathsf{5}^{\langle\rangle}\,\dfrac{\mathsf{b}^{\langle\rangle}\,\dfrac{\mathsf{4}^{[\,]}\,\dfrac{\overset{\mathcal{D}}{\Delta_1\{\langle\Delta_2\rangle,[\Delta_3],[\Delta_4]\}}}{\Delta_1\{\langle\Delta_2\rangle,[[\Delta_3],\Delta_4]\}}}{\Delta_1\{[\langle[\Delta_2]\rangle,[\Delta_3],\Delta_4]\}}}{\Delta_1\{[[\langle[\Delta_2]\rangle,\Delta_3],\Delta_4]\}}
$$

We rely on the crucial fact that $\mathsf{5}\in\mathsf{X}$ when $\{\mathsf{b},\mathsf{4}\}\subseteq\mathsf{X}$, as X is 45-closed. So, this case illustrates why 45-closure is needed, and also shows that the height of the proof can increase for the admissibility of the $\mathsf{x}^{[\,]}$ rules. ◀

▶ **Definition 6.4.** The *depth* of a polarized formula P or N, denoted by $\mathrm{dp}(P)$ or $\mathrm{dp}(N)$, is inductively given as follows:

$$
\begin{aligned}
\mathrm{dp}(p)=\mathrm{dp}(n)&=1 & \mathrm{dp}(P\mathbin{\overset{+}{\wedge}}Q)&=\max(\mathrm{dp}(P),\mathrm{dp}(Q))+1\\
\mathrm{dp}(\bot)=\mathrm{dp}(\overset{+}{\top})=\mathrm{dp}(\overset{-}{\top})&=1 & \mathrm{dp}(P\vee Q)&=\max(\mathrm{dp}(P),\mathrm{dp}(Q))+1\\
\mathrm{dp}(\uparrow P)=\mathrm{dp}(\Diamond P)&=\mathrm{dp}(P)+1 & \mathrm{dp}(P\supset N)&=\max(\mathrm{dp}(P),\mathrm{dp}(N))+1\\
\mathrm{dp}(\downarrow N)=\mathrm{dp}(\Box N)&=\mathrm{dp}(N)+1 & \mathrm{dp}(M\mathbin{\overset{-}{\wedge}}N)&=\max(\mathrm{dp}(M),\mathrm{dp}(N))+1
\end{aligned}
$$

The *rank* of a neutral sequent Δ, denoted by $\mathrm{rk}(\Delta)$, is the multiset of the depths of the formulas in Δ. Formally, it can be defined inductively as follows:

$$
\mathrm{rk}(L,\Lambda)=\{\!\!\{\mathrm{dp}(L)\}\!\!\}\uplus\mathrm{rk}(\Lambda)\quad \mathrm{rk}(\Lambda,R)=\mathrm{rk}(\Lambda)\uplus\{\!\!\{\mathrm{dp}(R)\}\!\!\}\quad \mathrm{rk}(\Lambda,[\Delta])=\mathrm{rk}(\Lambda)\uplus\mathrm{rk}(\Delta)
$$

▶ **Lemma 6.5** (Cut Reduction). *Let* $\mathsf{X}\subseteq\{\mathsf{t},\mathsf{d},\mathsf{4},\mathsf{b},\mathsf{5}\}$ *be 45-closed. Given a proof that ends:*

$$
\mathsf{cut}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_1}{\Gamma^{\star}\{\Delta\}}\qquad\overset{\mathcal{D}_2}{\Gamma^{\langle\!/\!\rangle}\{\langle\Delta\rangle\}}}{\Gamma\{\emptyset\}}
$$

where $\mathcal{D}_1$ *and* $\mathcal{D}_2$ *are in* $\mathsf{SyNIK}{+}\mathsf{X}^{\langle\rangle}$, *there is a proof of* $\Gamma\{\emptyset\}$ *in* $\mathsf{SyNIK}{+}\mathsf{X}^{\langle\rangle}$.

Proof. By lexicographic induction on the tuple $\langle\mathrm{rk}(\Delta),\mathrm{ht}(\mathcal{D}_2),\mathrm{ht}(\mathcal{D}_1)\rangle$, splitting cases on the last rule instances in $\mathcal{D}_1$ and $\mathcal{D}_2$. Note that the last rule in $\mathcal{D}_2$ always applies to the focus $\langle\Delta\rangle$. We will rewrite the derivation, written using $\rightsquigarrow$, by moving the instance of $\mathsf{cut}^{\langle\rangle}$ to a position of strictly lower measure or eliminating it entirely.

- First, let us consider the cases where $\mathcal{D}_2$ ends with a structural rule:

$$
\mathsf{cut}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_1}{\Gamma^{\star}\{\Delta,\Lambda\}}\qquad \mathsf{spl}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_2'}{\Gamma^{\langle\!/\!\rangle}\{\langle\Delta\rangle\}}\quad\overset{\mathcal{D}_2''}{\Gamma^{\langle\!/\!\rangle}\{\langle\Lambda\rangle\}}}{\Gamma^{\langle\!/\!\rangle}\{\langle\Delta,\Lambda\rangle\}}}{\Gamma\{\emptyset\}}
\quad\rightsquigarrow\quad
\mathsf{cut}^{\langle\rangle}\,\dfrac{\mathsf{cut}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_1}{\Gamma^{\star}\{\Delta,\Lambda\}}\qquad \mathsf{weak}\,\dfrac{\overset{\mathcal{D}_2'}{\Gamma^{\langle\!/\!\rangle}\{\langle\Delta\rangle\}}}{\Gamma^{\langle\!/\!\rangle}\{\langle\Delta\rangle,\Lambda\}}}{\Gamma\{\Lambda\}}\qquad\overset{\mathcal{D}_2''}{\Gamma^{\langle\!/\!\rangle}\{\langle\Lambda\rangle\}}}{\Gamma\{\emptyset\}}
$$

$$\mathsf{cut}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_1}{\Delta_1^\star\{[\Delta_2^\star],[\Delta_3]\}} \qquad \mathsf{k}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_2'}{\Delta_1^{\not\langle\rangle}\big\{\big[\Delta_2^{\not\langle\rangle},\langle\Delta_3\rangle\big]\big\}}}{\Delta_1^{\not\langle\rangle}\big\{\big[\Delta_2^{\not\langle\rangle}\big],\langle[\Delta_3]\rangle\big\}}}{\Delta_1\{[\Delta_2]\}} \rightsquigarrow \mathsf{cut}^{\langle\rangle}\,\dfrac{\mathsf{k}^{[]}\,\dfrac{\overset{\mathcal{D}_1}{\Delta_1^\star\{[\Delta_2^\star],[\Delta_3]\}}}{\Delta_1^\star\{[\Delta_2^\star,\Delta_3]\}} \qquad \overset{\mathcal{D}_2'}{\Delta_1^{\not\langle\rangle}\big\{\big[\Delta_2^{\not\langle\rangle},\langle\Delta_3\rangle\big]\big\}}}{\Delta_1\{[\Delta_2]\}}$$

$$\mathsf{cut}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_1}{\Gamma^\star\{[\Delta]\}} \qquad \mathsf{d}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_2'}{\Gamma^{\not\langle\rangle}\{[\langle\Delta\rangle]\}}}{\Gamma^{\not\langle\rangle}\{\langle[\Delta]\rangle\}}}{\Gamma\{\emptyset\}} \rightsquigarrow \mathsf{d}^{[]}\,\dfrac{\mathsf{cut}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_1}{\Gamma^\star\{[\Delta]\}} \qquad \overset{\mathcal{D}_2'}{\Gamma^{\not\langle\rangle}\{[\langle\Delta\rangle]\}}}{\Gamma\{[\emptyset]\}}}{\Gamma\{\emptyset\}}$$

$$\mathsf{cut}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_1}{\Gamma^\star\{[\Delta]\}} \qquad \mathsf{t}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_2'}{\Gamma^{\not\langle\rangle}\{\langle\Delta\rangle\}}}{\Gamma^{\not\langle\rangle}\{\langle[\Delta]\rangle\}}}{\Gamma\{\emptyset\}} \rightsquigarrow \mathsf{cut}^{\langle\rangle}\,\dfrac{\mathsf{t}^{[]}\,\dfrac{\overset{\mathcal{D}_1}{\Gamma^\star\{[\Delta]\}}}{\Gamma^\star\{\Delta\}} \qquad \overset{\mathcal{D}_2'}{\Gamma^{\not\langle\rangle}\{\langle\Delta\rangle\}}}{\Gamma\{\emptyset\}}$$

$$\mathsf{cut}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_1}{\Delta_1^\star\{[\Delta_2^\star,[\Delta_3]]\}} \qquad \mathsf{b}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_2'}{\Delta_1^{\not\langle\rangle}\big\{\big[\Delta_2^{\not\langle\rangle}\big],\langle\Delta_3\rangle\big\}}}{\Delta_1^{\not\langle\rangle}\big\{\big[\Delta_2^{\not\langle\rangle},\langle[\Delta_3]\rangle\big]\big\}}}{\Delta_1\{[\Delta_2]\}} \rightsquigarrow \mathsf{cut}^{\langle\rangle}\,\dfrac{\mathsf{b}^{[]}\,\dfrac{\overset{\mathcal{D}_1}{\Delta_1^\star\{[\Delta_2^\star,[\Delta_3]]\}}}{\Delta_1^\star\{[\Delta_2^\star],\Delta_3\}} \qquad \overset{\mathcal{D}_2'}{\Delta_1^{\not\langle\rangle}\big\{\big[\Delta_2^{\not\langle\rangle}\big],\langle\Delta_3\rangle\big\}}}{\Delta_1\{[\Delta_2]\}}$$

$$\mathsf{cut}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_1}{\Delta_1^\star\{[\Delta_2^\star],[\Delta_3]\}} \qquad \mathsf{4}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_2'}{\Delta_1^{\not\langle\rangle}\big\{\big[\Delta_2^{\not\langle\rangle},\langle[\Delta_3]\rangle\big]\big\}}}{\Delta_1^{\not\langle\rangle}\big\{\big[\Delta_2^{\not\langle\rangle}\big],\langle[\Delta_3]\rangle\big\}}}{\Delta_1\{[\Delta_2]\}} \rightsquigarrow \mathsf{cut}^{\langle\rangle}\,\dfrac{\mathsf{4}^{[]}\,\dfrac{\overset{\mathcal{D}_1}{\Delta_1^\star\{[\Delta_2^\star],[\Delta_3]\}}}{\Delta_1^\star\{[\Delta_2^\star,[\Delta_3]]\}} \qquad \overset{\mathcal{D}_2'}{\Delta_1^{\not\langle\rangle}\big\{\big[\Delta_2^{\not\langle\rangle},\langle[\Delta_3]\rangle\big]\big\}}}{\Delta_1\{[\Delta_2]\}}$$

$$\mathsf{cut}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_1}{\Gamma^\star\{[\Delta]\}\{\emptyset\}} \qquad \mathsf{5}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_2'}{\Gamma^{\not\langle\rangle}\{\emptyset\}\{\langle[\Delta]\rangle\}}}{\Gamma^{\not\langle\rangle}\{\langle[\Delta]\rangle\}\{\emptyset\}}}{\Gamma\{\emptyset\}\{\emptyset\}} \rightsquigarrow \mathsf{cut}^{\langle\rangle}\,\dfrac{\mathsf{5}^{[]}\,\dfrac{\overset{\mathcal{D}_1}{\Gamma^\star\{[\Delta]\}\{\emptyset\}}}{\Gamma^\star\{\emptyset\}\{[\Delta]\}} \qquad \overset{\mathcal{D}_2'}{\Gamma^{\not\langle\rangle}\{\emptyset\}\{\langle[\Delta]\rangle\}}}{\Gamma\{\emptyset\}\{\emptyset\}}$$

In each case we can apply the induction hypothesis because $\mathrm{ht}(\mathcal{D}_2') < \mathrm{ht}(\mathcal{D}_2)$ and in the first case also $\mathrm{ht}(\mathcal{D}_2'') < \mathrm{ht}(\mathcal{D}_2)$.[3] Note the use of Lemma 6.3.

- If the last rule in $\mathcal{D}_2$ is an axiom, we have one of the following three cases:

$$\mathsf{cut}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_1}{\Gamma\{\emptyset\}} \qquad \mathsf{fin}^{\langle\rangle}\,\dfrac{}{\Gamma^{\not\langle\rangle}\{\langle\emptyset\rangle\}}}{\Gamma\{\emptyset\}} \rightsquigarrow \overset{\mathcal{D}_1}{\Gamma\{\emptyset\}} \qquad\qquad \mathsf{cut}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_1}{\Lambda\{n\}} \qquad \mathsf{id}^{\langle\rangle}\,\dfrac{}{\Lambda^{\not\langle\rangle}\{n,\langle n\rangle\}}}{\Lambda\{n\}} \rightsquigarrow \overset{\mathcal{D}_1}{\Lambda\{n\}}$$

$$\mathsf{cut}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_1}{\Gamma\{p,p\}} \qquad \mathsf{id}^{\langle\rangle}\,\dfrac{}{\Gamma^{\not\langle\rangle}\{p,\langle p\rangle\}}}{\Gamma\{p\}} \rightsquigarrow \mathsf{cont}\,\dfrac{\overset{\mathcal{D}_1}{\Gamma\{p,p\}}}{\Gamma\{p\}}$$

For the third case we use the admissibility of atomic contraction.

- Finally, the last rule in $\mathcal{D}_2$ can be $\uparrow_{\mathsf{L}}^{\langle\rangle}$ or $\downarrow_{\mathsf{R}}^{\langle\rangle}$, and if at the same time the last rule in $\mathcal{D}_1$ is the corresponding $\downarrow_{\mathsf{L}}^{\langle\rangle}$ or $\uparrow_{\mathsf{R}}^{\langle\rangle}$ on the cut formula (the cut sequent has to be a singleton in that case), we have one of the two principal cases:

$$\mathsf{cut}^{\langle\rangle}\,\dfrac{\downarrow_{\mathsf{L}}^{\langle\rangle}\,\dfrac{\Gamma_2 \Subset N \quad \overset{\mathcal{D}_1'}{\Gamma_1\{\downarrow N,\langle\Gamma_2\rangle\}}}{\Gamma_1\{\downarrow N\}} \qquad \downarrow_{\mathsf{R}}^{\langle\rangle}\,\dfrac{\left\{\begin{matrix}\mathcal{D}_{\Gamma_2}\\ \Gamma_1^*\{\Gamma_2\}\end{matrix}\right\}_{\Gamma_2 \Subset N}}{\Gamma_1\{\langle\downarrow N\rangle\}}}{\Gamma_1\{\emptyset\}} \rightsquigarrow \mathsf{cut}^{\langle\rangle}\,\dfrac{\mathsf{cut}^{\langle\rangle}\,\dfrac{\overset{\mathcal{D}_1'}{\Gamma_1\{\downarrow N,\langle\Gamma_2\rangle\}} \qquad \overset{\mathcal{D}_2}{\Gamma_1\{\langle\downarrow N\rangle\}}}{\Gamma_1\{\langle\Gamma_2\rangle\}} \qquad \overset{\mathcal{D}_{\Gamma_2}}{\Gamma_1^*\{\Gamma_2\}}}{\Gamma_1\{\emptyset\}}$$

[3] We abuse the pruning notation in the cases for $\mathsf{k}^{\langle\rangle}$, $\mathsf{b}^{\langle\rangle}$ and $\mathsf{4}^{\langle\rangle}$ by writing $\Delta_1^\star\{[\Delta_2^\star],\{\ \}\}$ or $\Delta_1^\star\{[\Delta_2^\star,\{\ \}]\}$ to denote the pruned context $\Delta^\star\{\ \}$ when $\Delta\{\ \} = \Delta_1\{[\Delta_2],\{\ \}\}$ or $\Delta\{\ \} = \Delta_1\{[\Delta_2,\{\ \}]\}$ respectively.

$$
{\scriptstyle\mathsf{cut}^{\langle\rangle}}\dfrac{{\scriptstyle\uparrow_{\mathsf{R}}^{\langle\rangle}}\dfrac{\Lambda\Subset P \qquad \begin{matrix}\mathcal{D}_1'\\ \Gamma^*\{\uparrow P,\langle\Lambda\rangle\}\end{matrix}}{\Gamma^*\{\uparrow P\}} \qquad {\scriptstyle\uparrow_{\mathsf{L}}^{\langle\rangle}}\dfrac{\left\{\begin{matrix}\mathcal{D}_\Lambda\\ \Gamma\{\Lambda\}\end{matrix}\right\}\Lambda\Subset P}{\Gamma\{\langle\uparrow P\rangle\}}}{\Gamma\{\emptyset\}}
\quad\leadsto\quad
{\scriptstyle\mathsf{cut}^{\langle\rangle}}\dfrac{{\scriptstyle\mathsf{cut}^{\langle\rangle}}\dfrac{\begin{matrix}\mathcal{D}_1'\\ \Gamma^*\{\uparrow P,\langle\Lambda\rangle\}\end{matrix}\qquad\begin{matrix}\mathcal{D}_2\\ \Gamma\{\langle\uparrow P\rangle\}\end{matrix}}{\Gamma\{\langle\Lambda\rangle\}}\qquad\begin{matrix}\mathcal{D}_\Lambda\\ \Gamma\{\Lambda\}\end{matrix}}{\Gamma\{\emptyset\}}
$$

In both cases we have to apply the induction hypothesis twice: first to the upper cut because $\mathrm{ht}(\mathcal{D}_1') < \mathrm{ht}(\mathcal{D}_1)$, and then to the lower cut because $\mathrm{rk}(\Gamma_2) < \mathrm{rk}(\downarrow N)$ and $\mathrm{rk}(\Lambda) < \mathrm{rk}(\uparrow P)$. After the reduction step the focus is not in the same branch any more, so which branch is considered to be $\mathcal{D}_1$ or $\mathcal{D}_2$ may change, but since the rank has decreased strictly this does not affect the inductive argument.

- Of course, when the last rule in $\mathcal{D}_2$ is $\uparrow_{\mathsf{L}}^{\langle\rangle}$ or $\downarrow_{\mathsf{R}}^{\langle\rangle}$, the last rule in $\mathcal{D}_1$ does not need to be the corresponding $\uparrow_{\mathsf{R}}^{\langle\rangle}$ or $\downarrow_{\mathsf{L}}^{\langle\rangle}$ rule. In that case we have a commutative case: the last rule in $\mathcal{D}_1$ is permuted under the cut:

$$
{\scriptstyle\mathsf{cut}^{\langle\rangle}}\dfrac{{\scriptstyle\mathsf{r}}\dfrac{\begin{matrix}\mathcal{D}_1'\\ \Gamma_1^\star\{\Delta\}\end{matrix}}{\Gamma^\star\{\Delta\}}\qquad\begin{matrix}\mathcal{D}_2\\ \Gamma^{\not{\langle\rangle}}\{\langle\Delta\rangle\}\end{matrix}}{\Gamma\{\emptyset\}}
\quad\leadsto\quad
{\scriptstyle\mathsf{r}}\dfrac{{\scriptstyle\mathsf{cut}^{\langle\rangle}}\dfrac{\begin{matrix}\mathcal{D}_1'\\ \Gamma_1^\star\{\Delta\}\end{matrix}\qquad\begin{matrix}\mathcal{D}_2\\ \Gamma_1^{\not{\langle\rangle}}\{\langle\Delta\rangle\}\end{matrix}}{\Gamma_1\{\emptyset\}}}{\Gamma\{\emptyset\}}
$$

The situation above applies if r is $\mathsf{k}^{\langle\rangle}$ or any of the $\mathsf{x}^{\langle\rangle}$ rules, because then there is a focus in $\Gamma\{\ \}$ which is moved by r, and we have $\Gamma_1^{\not{\langle\rangle}}\{\ \} = \Gamma^{\not{\langle\rangle}}\{\ \}$. It also applies if r is one of $\downarrow_{\mathsf{L}}^{\langle\rangle}$ or $\uparrow_{\mathsf{R}}^{\langle\rangle}$ because then $\Gamma\{\ \}$ contains no focus and therefore $\Gamma_1^{\not{\langle\rangle}}\{\ \} = \Gamma^{\not{\langle\rangle}}\{\ \}$. If the last rule in $\mathcal{D}_1$ is $\mathsf{spl}^{\langle\rangle}$ the situation is similar, and if it is one of $\mathsf{id}^{\langle\rangle}$ or $\mathsf{fin}^{\langle\rangle}$, then the cut disappears trivially. Note that the last rule in $\mathcal{D}_1$ is not applying to Δ (which is a singleton) because otherwise it would be a principal case. The only nontrivial commutative cases are when the focus in $\Gamma\{\ \}$ is released by the last rule in $\mathcal{D}_1$ which can be either a $\uparrow_{\mathsf{L}}^{\langle\rangle}$ or a $\downarrow_{\mathsf{R}}^{\langle\rangle}$. In the $\uparrow_{\mathsf{L}}^{\langle\rangle}$-case, we can reduce as follows:

$$
{\scriptstyle\mathsf{cut}^{\langle\rangle}}\dfrac{{\scriptstyle\uparrow_{\mathsf{L}}^{\langle\rangle}}\dfrac{\left\{\begin{matrix}\mathcal{D}_\Lambda\\ \Gamma^\star\{\Lambda\}\{\Delta\}\end{matrix}\right\}\Lambda\Subset P}{\Gamma^\star\{\langle\uparrow P\rangle\}\{\Delta\}}\qquad\begin{matrix}\mathcal{D}_2\\ \Gamma\{\emptyset\}\{\langle\Delta\rangle\}\end{matrix}}{\Gamma\{\langle\uparrow P\rangle\}\{\emptyset\}}
\quad\leadsto\quad
{\scriptstyle\uparrow_{\mathsf{L}}^{\langle\rangle}}\dfrac{\left\{{\scriptstyle\mathsf{cut}^{\langle\rangle}}\dfrac{\begin{matrix}\mathcal{D}_\Lambda\\ \Gamma^\star\{\Lambda\}\{\Delta\}\end{matrix}\qquad{\scriptstyle\mathsf{weak}}\dfrac{\begin{matrix}\mathcal{D}_2\\ \Gamma\{\emptyset\}\{\langle\Delta\rangle\}\end{matrix}}{\Gamma\{\Lambda\}\{\langle\Delta\rangle\}}}{\Gamma\{\Lambda\}\{\emptyset\}}\right\}\Lambda\Subset P}{\Gamma\{\langle\uparrow P\rangle\}\{\emptyset\}}
$$

and we only need height-preserving admissibility of weakening in order to apply the induction hypothesis, using $\mathrm{ht}(\mathcal{D}_\Lambda) < \mathrm{ht}(\mathcal{D}_1)$. In the $\downarrow_{\mathsf{R}}^{\langle\rangle}$-case we need to distinguish whether Δ is of the form $\uparrow P$ or $\downarrow N$. In the first case the cut disappears:

$$
{\scriptstyle\mathsf{cut}^{\langle\rangle}}\dfrac{{\scriptstyle\downarrow_{\mathsf{R}}^{\langle\rangle}}\dfrac{\left\{\begin{matrix}\mathcal{D}_{\Gamma_2}\\ \Gamma_1^*\{\Gamma_2\}\{\emptyset\}\end{matrix}\right\}\Gamma_2\Subset N}{\Gamma_1^*\{\langle\downarrow N\rangle\}\{\uparrow P\}}\qquad\begin{matrix}\mathcal{D}_2\\ \Gamma_1\{\emptyset\}\{\langle\uparrow P\rangle\}\end{matrix}}{\Gamma_1\{\langle\downarrow N\rangle\}\{\emptyset\}}
\quad\leadsto\quad
{\scriptstyle\downarrow_{\mathsf{R}}^{\langle\rangle}}\dfrac{\left\{\begin{matrix}\mathcal{D}_{\Gamma_2}\\ \Gamma_1^*\{\Gamma_2\}\{\emptyset\}\end{matrix}\right\}\Gamma_2\Subset N}{\Gamma_1\{\langle\downarrow N\rangle\}\{\emptyset\}}
$$

and in the second we again use height-preserving admissibility of weakening in order to apply the induction hypothesis, as $\mathrm{ht}(\mathcal{D}_{\Gamma_2}) < \mathrm{ht}(\mathcal{D}_1)$:

$$
{\scriptstyle\mathsf{cut}^{\langle\rangle}}\dfrac{{\scriptstyle\downarrow_{\mathsf{R}}^{\langle\rangle}}\dfrac{\left\{\begin{matrix}\mathcal{D}_{\Gamma_2}\\ \Gamma_1^*\{\Gamma_2\}\{\downarrow M\}\end{matrix}\right\}\Gamma_2\Subset N}{\Gamma_1\{\langle\downarrow N\rangle\}\{\downarrow M\}}\qquad{\scriptstyle\downarrow_{\mathsf{R}}^{\langle\rangle}}\dfrac{\left\{\begin{matrix}\mathcal{D}_{\Gamma_3}\\ \Gamma_1^*\{\emptyset\}\{\Gamma_3\}\end{matrix}\right\}\Gamma_3\Subset M}{\Gamma_1\{\emptyset\}\{\langle\downarrow M\rangle\}}}{\Gamma_1\{\langle\downarrow N\rangle\}\{\emptyset\}}
$$

$$
\leadsto\quad
{\scriptstyle\downarrow_{\mathsf{R}}^{\langle\rangle}}\dfrac{\left\{{\scriptstyle\mathsf{cut}^{\langle\rangle}}\dfrac{\begin{matrix}\mathcal{D}_{\Gamma_2}\\ \Gamma_1^*\{\Gamma_2\}\{\downarrow M\}\end{matrix}\qquad{\scriptstyle\downarrow_{\mathsf{R}}^{\langle\rangle}}\dfrac{\left\{{\scriptstyle\mathsf{weak}}\dfrac{\begin{matrix}\mathcal{D}_{\Gamma_3}\\ \Gamma_1^*\{\emptyset\}\{\Gamma_3\}\end{matrix}}{\Gamma_1^*\{\Gamma_2^*\}\{\Gamma_3\}}\right\}\Gamma_3\Subset M}{\Gamma_1^*\{\Gamma_2\}\{\langle\downarrow M\rangle\}}}{\Gamma_1^*\{\Gamma_2\}\{\emptyset\}}\right\}\Gamma_2\Subset N}{\Gamma_1\{\langle\downarrow N\rangle\}\{\emptyset\}}
\qquad\blacktriangleleft
$$

▶ **Theorem 6.6** (Cut-Elimination). *Let* $\mathsf{X} \subseteq \{\mathsf{t},\mathsf{d},\mathsf{4},\mathsf{b},\mathsf{5}\}$ *be* 45*-closed. If a sequent* Γ *is provable in* $\mathsf{SyNIK{+}X^{\langle\rangle}{+}cut}$*, then it is also provable in* $\mathsf{SyNIK{+}X^{\langle\rangle}}$*.*

Proof. By induction on the number of cuts in the proof, by repeatedly applying Lemma 6.5, always starting with a topmost cut. ◀

▶ **Lemma 6.7** (Identity). *The following rule is derivable in* SyNIK*:* $\mathsf{sid}^{\langle\rangle}\,\dfrac{}{\Delta_1\{\Delta_2,\langle\Delta_2\rangle\}}$.

Proof. The proof, by induction on the structure of the focus, is similar to the one for classical modal logics in [7, Lemma 4.6]. ◀

▶ **Lemma 6.8** (Simulation). *Let* A° *be provable in* $\mathsf{NIK{+}X}$*, and let* R *be a right-neutral formula with* $\lfloor R \rfloor = A$*. Then* R *is provable in* $\mathsf{SyNIK{+}X^{\langle\rangle}{+}cut}$*.*

Proof. First, any NIK sequent can be transformed into a neutral polarized sequent with the same meaning. The connectives are turned into their polarized variant and in particular, a polarity is arbitrarily chosen for every atom, every $\top$, and every $\wedge$; then shifts are added as needed to produce well-formed polarized formulas. Once the formulas are polarized, one can obtain neutrality, and remove the $^\circ$-annotation, by adding extra shifts in front of each formula in the sequent as follows: if P is a positive formula, $P \mapsto {\downarrow}{\uparrow}P$ and $P^\circ \mapsto {\uparrow}P$, and if N is a negative formula, $N \mapsto {\downarrow}N$ and $N^\circ \mapsto {\uparrow}{\downarrow}N$. Each rule of $\mathsf{NIK{+}X}$ can therefore be considered as a rule between neutral polarized sequents. As such, it can be shown to be derivable in $\mathsf{SyNIK{+}X^{\langle\rangle}{+}cut}$. We show the cases for the rules id, $\wedge_{\mathsf{L}}$ and $\Diamond_{\mathsf{Rd}}$. The other cases are similar.

$$\mathsf{id}\,\frac{}{\Lambda\{a,a^\circ\}} \qquad\text{becomes}\qquad {\downarrow}^{\langle\rangle}_{\mathsf{L}}\,\cfrac{\mathsf{sid}^{\langle\rangle}\,\cfrac{}{\Lambda\{{\downarrow}{\uparrow}p,\langle{\uparrow}p\rangle,{\uparrow}p\}}}{\Lambda\{{\downarrow}{\uparrow}p,{\uparrow}p\}} \qquad\text{or}\qquad {\uparrow}^{\langle\rangle}_{\mathsf{R}}\,\cfrac{\mathsf{sid}^{\langle\rangle}\,\cfrac{}{\Lambda\{{\downarrow}n,{\uparrow}{\downarrow}n,\langle{\downarrow}n\rangle\}}}{\Lambda\{{\downarrow}n,{\uparrow}{\downarrow}n\}}$$

$$\wedge_{\mathsf{L}}\,\frac{\Gamma\{A,B\}}{\Gamma\{A\wedge B\}} \qquad\text{becomes}$$

$$\mathsf{cut}^{\langle\rangle}\,\cfrac{\mathsf{weak}\,\cfrac{\Gamma\{{\downarrow}{\uparrow}P,{\downarrow}{\uparrow}Q\}}{\Gamma\{{\downarrow}{\uparrow}(P\overset{+}{\wedge}Q),{\downarrow}{\uparrow}P,{\downarrow}{\uparrow}Q\}} \qquad \mathsf{spl}^{\langle\rangle}\,\cfrac{{\downarrow}^{\langle\rangle}_{\mathsf{R}}\,\cfrac{{\downarrow}^{\langle\rangle}_{\mathsf{L}}\,\cfrac{{\downarrow}^{\langle\rangle}_{\mathsf{R}}\,\cfrac{\left\{\ {\uparrow}^{\langle\rangle}_{\mathsf{R}}\,\cfrac{\mathsf{sid}^{\langle\rangle}\,\cfrac{}{\Gamma^*\{{\downarrow}{\uparrow}(P\overset{+}{\wedge}Q),\Delta_P,\Delta_Q,{\uparrow}Q,\langle\Delta_Q\rangle\}}}{\Gamma^*\{{\downarrow}{\uparrow}(P\overset{+}{\wedge}Q),\Delta_P,\Delta_Q,{\uparrow}Q\}}\ \right\}_{\Delta_P\Subset P,\Delta_Q\Subset Q}}{\Gamma^*\{{\downarrow}{\uparrow}(P\overset{+}{\wedge}Q),\langle{\uparrow}(P\overset{+}{\wedge}Q)\rangle,{\uparrow}Q\}}}{\Gamma^*\{{\downarrow}{\uparrow}(P\overset{+}{\wedge}Q),{\uparrow}Q\}}}{\Gamma\{{\downarrow}{\uparrow}(P\overset{+}{\wedge}Q),\langle{\downarrow}{\uparrow}Q\rangle\}} \qquad \vdots}{\Gamma\{{\downarrow}{\uparrow}(P\overset{+}{\wedge}Q),\langle{\downarrow}{\uparrow}P,{\downarrow}{\uparrow}Q\rangle\}}}{\Gamma\{{\downarrow}{\uparrow}(P\overset{+}{\wedge}Q)\}}$$

(where the omitted third premise derivation is the similar branch for $\langle{\downarrow}{\uparrow}P\rangle$), or it becomes

$$\mathsf{cut}^{\langle\rangle}\,\cfrac{\mathsf{weak}\,\cfrac{\Gamma\{{\downarrow}N,{\downarrow}M\}}{\Gamma\{{\downarrow}(N\bar{\wedge}M),{\downarrow}N,{\downarrow}M\}} \qquad \mathsf{spl}^{\langle\rangle}\,\cfrac{{\downarrow}^{\langle\rangle}_{\mathsf{R}}\,\cfrac{\left\{\ {\downarrow}^{\langle\rangle}_{\mathsf{L}}\,\cfrac{\mathsf{sid}^{\langle\rangle}\,\cfrac{}{\Gamma^*\{{\downarrow}(N\bar{\wedge}M),\langle\Delta_N\rangle,\Delta_N\}}}{\Gamma^*\{{\downarrow}(N\bar{\wedge}M),\Delta_N\}}\ \right\}_{\Delta_N\Subset N}}{\Gamma\{{\downarrow}(N\bar{\wedge}M),\langle{\downarrow}N\rangle\}} \qquad \vdots}{\Gamma\{{\downarrow}(N\bar{\wedge}M),\langle{\downarrow}N,{\downarrow}M\rangle\}}}{\Gamma\{{\downarrow}(N\bar{\wedge}M)\}}$$

(where the omitted third premise derivation is the similar branch for $\langle{\downarrow}M\rangle$). Finally,

$$\Diamond_{\mathsf{Rd}}\,\frac{\Lambda\{[A^\circ]\}}{\Lambda\{\Diamond A^\circ\}} \qquad\text{becomes}\qquad \mathsf{cut}^{\langle\rangle}\,\cfrac{\Lambda\{[{\uparrow}P]\} \qquad \mathsf{d}^{\langle\rangle}\,\cfrac{\left\{\ {\uparrow}^{\langle\rangle}_{\mathsf{L}}\,\cfrac{{\uparrow}^{\langle\rangle}_{\mathsf{R}}\,\cfrac{\mathsf{k}^{\langle\rangle}\,\cfrac{\mathsf{sid}^{\langle\rangle}\,\cfrac{}{\Lambda\{{\uparrow}\Diamond P,[\langle\Delta_P\rangle,\Delta_P]\}}}{\Lambda\{{\uparrow}\Diamond P,\langle[\Delta_P]\rangle,[\Delta_P]\}}}{\Lambda\{{\uparrow}\Diamond P,[\Delta_P]\}}}{\Lambda\{{\uparrow}\Diamond P,[\langle{\uparrow}P\rangle]\}}\ \right\}_{\Delta_P\Subset P}}{\Lambda\{{\uparrow}\Diamond P,\langle[{\uparrow}P]\rangle\}}}{\Lambda\{{\uparrow}\Diamond P\}}$$

The lemma then follows by replacing in the proof of P° (or N°) in $\mathsf{NIK{+}X}$ each instance of a rule by the corresponding derivation in $\mathsf{SyNIK{+}X^{\langle\rangle}{+}cut}$, which builds a proof of ${\uparrow}P$ (or ${\uparrow}{\downarrow}N$ resp.) in $\mathsf{SyNIK{+}X^{\langle\rangle}{+}cut}$. ◀

We can now summarize the results of this paper in the following theorem:

▶ **Theorem 6.9.** *Let* $\mathsf{X} \subseteq \{\mathsf{t}, \mathsf{d}, \mathsf{4}, \mathsf{b}, \mathsf{5}\}$ *be 45-closed and let* Γ *be a neutral sequent. The following are equivalent:*

1. $\mathrm{fm}(\lfloor\Gamma\rfloor)$ *is* X*-valid.*
2. Γ *is provable in* $\mathsf{FoNIK}{+}\mathsf{X}^{\mathsf{f}}$*.*
3. Γ *is provable in* $\mathsf{SyNIK}{+}\mathsf{X}^{\langle\rangle}$*.*
4. Γ *is provable in* $\mathsf{SyNIK}{+}\mathsf{X}^{\langle\rangle}{+}\mathsf{cut}$*.*

Proof. 4 → 3 is just Theorem 6.6; 3 → 2 follows from Lemma 5.4; 2 → 1 follows from Lemma 4.6; and finally 1 → 4 follows from Lemma 6.8 with the use of Theorem 3.8. Observe that the proof of Lemma 6.8 applies to unpolarized sequents and neutral sequents in general, and not just output formulas and right-neutral formulas. ◀

References

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

2 Arnon Avron. The method of hypersequents in the proof theory of propositional non-classical logics. In *Logic: from foundations to applications: European logic colloquium*, pages 1–32. Clarendon Press, 1996.

3 Taus Brock-Nannestad and Carsten Schürmann. Focused natural deduction. In *LPAR 17*, volume 6397 of *LNCS*, pages 157–171, Yogyakarta, Indonesia, 2010. Springer. `doi:10.1007/978-3-642-16242-8_12`.

4 Kai Brünnler. Deep sequent systems for modal logic. *Archive for Mathematical Logic*, 48(6):551–577, 2009.

5 Kai Brünnler and Lutz Straßburger. Modular sequent systems for modal logic. In Martin Giese and Arild Waaler, editors, *Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX'09*, volume 5607 of *LNCS*, pages 152–166. Springer, 2009. `doi:10.1007/978-3-642-02716-1_12`.

6 Kaustuv Chaudhuri, Nicolas Guenot, and Lutz Straßburger. The focused calculus of structures. In *CSL*, (LIPIcs), pages 159–173. Schloss Dagstuhl–LZI, 2011. `doi:10.4230/LIPIcs.CSL.2011.159`.

7 Kaustuv Chaudhuri, Sonia Marin, and Lutz Straßburger. Focused and synthetic nested sequents. In Bart Jacobs and Christof Löding, editors, *FoSSaCS*, April 2016. To appear.

8 Kaustuv Chaudhuri, Frank Pfenning, and Greg Price. A logical characterization of forward and backward chaining in the inverse method. *J. of Autom. Reasoning*, 40(2-3):133–177, 2008. `doi:10.1007/s10817-007-9091-0`.

9 François Lamarche. On the algebra of structural contexts. Technical report, INRIA, 2006. Available at `https://hal.inria.fr/inria-00099461`; to appear in MSCS.

10 Olivier Laurent. A proof of the focalization property in linear logic. Unpublished note, 2004.

11 Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, 2009. `doi:10.1016/j.tcs.2009.07.041`.

12 Sonia Marin and Lutz Straßburger. Label-free modular systems for classical and intuitionistic modal logics. In *Advances in Modal Logic (AIML-10)*, 2014.

13 Sean McLaughlin and Frank Pfenning. Imogen: Focusing the polarized focused inverse method for intuitionistic propositional logic. In *15th LPAR*, volume 5330 of *LNCS*, pages 174–181, 2008.

14 Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

15 Sara Negri. Proof analysis in modal logic. *Journal of Philosophical Logic*, 34(5-6):507–544, 2005. doi:10.1007/s10992-005-2267-3.

16 Gordon D. Plotkin and Colin Stirling. A framework for intuitionistic modal logics. In *1st Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 399–406. Morgan Kaufmann, 1986.

17 Jason Reed and Frank Pfenning. Focus-preserving embeddings of substructural logics in intuitionistic logic. Draft manuscript, January 2010.

18 Gisèle Fischer Servi. Axiomatizations for some intuitionistic modal logics. *Rendiconti del Seminario Matematico dell' Università Politecnica di Torino*, 42(3):179–194, 1984.

19 Robert J. Simmons. Structural focalization. *ACM Trans. Comput. Log.*, 15(3):21:1–21:33, 2014. doi:10.1145/2629678.

20 Alex Simpson. *The Proof Theory and Semantics of Intuitionistic Modal Logic*. PhD thesis, University of Edinburgh, 1994.

21 Lutz Straßburger. Cut elimination in nested sequents for intuitionistic modal logics. In *16th International Conference on Foundations of Software Science and Computation Structures (FOSSACS)*, volume 7794 of *LNCS*, pages 209–224. Springer, 2013. doi:10.1007/978-3-642-37075-5_14.

22 Noam Zeilberger. Focusing and higher-order abstract syntax. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 359–369. ACM, 2008.

The Independence of Markov's Principle in Type Theory

Thierry Coquand[1] **and Bassel Mannaa**[2]

1 **Department of Computer Science and Engineering, University of Gothenburg, Gothenburg, Sweden**
`thierry.coquand@cse.gu.se`

2 **Department of Computer Science and Engineering, University of Gothenburg, Gothenburg, Sweden**
`bassel.mannaa@cse.gu.se`

Abstract

In this paper, we show that Markov's principle is not derivable in dependent type theory with natural numbers and one universe. One tentative way to prove this would be to remark that Markov's principle does not hold in a sheaf model of type theory over Cantor space, since Markov's principle does not hold for the generic point of this model. It is however not clear how to interpret the universe in a sheaf model [9, 17, 21]. Instead we design an extension of type theory, which intuitively extends type theory by the addition of a generic point of Cantor space. We then show the consistency of this extension by a normalization argument. Markov's principle does not hold in this extension, and it follows that it cannot be proved in type theory.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Forcing, Dependent type theory, Markov's Principle, Cantor Space

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.17

1 Introduction

Markov's principle has a special status in constructive mathematics. One way to formulate this principle is that if it is impossible that a given algorithm does not terminate, then it does terminate. It is equivalent to the fact that if a set of natural number and its complement are both computably enumerable, then this set is decidable. This form is often used in recursivity theory. This principle was first formulated by Markov, who called it "Leningrad's principle", and founded a branch of constructive mathematics around this principle [14].

This principle is also equivalent to the fact that if a given real number is *not* equal to 0 then this number is *apart* from 0 (that is this number is $< -r$ or $> r$ for some rational number $r > 0$). On this form, it was explicitly *refuted* by Brouwer in intuitionistic mathematics, who gave an example of a real number (well defined intuitionistically) which is not equal to 0, but also not apart from 0. (The motivation of Brouwer for this example was to show the necessity of using *negation* in intuitionistic mathematics [4].) The idea of Brouwer can be represented formally using topological models [19].

In a neutral approach to mathematics, such as Bishop's [3], Markov's principle is simply left undecided. We also expect to be able to prove that Markov's principle is *not* provable in formal system in which we can express Bishop's mathematics. For instance, Kreisel [12] introduced *modified realizability* to show that Markov's principle is not derivable in the formal system HA^ω. Similarly, one would expect that Markov's principle is *not* derivable in

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 17; pp. 17:1–17:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Martin-Löf type theory [15], but, as far as we know, such a result has not been established yet. [1]

We say that a statement A is *independent* of some formal system if A cannot be derived in that system. A statement in the formal system of Martin-Löf type theory (MLTT) is represented by a closed type. A statement/type A is derivable if it is inhabited by some term t (written $\mathrm{MLTT} \vdash t : A$). This is the so-called propositions-as-types principle. Correspondingly we say that a statement A (represented as a type) is independent of MLTT if there is no term t such that $\mathrm{MLTT} \vdash t : A$.

The main result of this paper is to show that Markov's principle is independent of Martin-Löf type theory.[2]

The main idea for proving this independence is to follow Brouwer's argument. We want to extend type theory with a "generic" infinite sequence of 0 and 1 and establish that it is both absurd that this generic sequence is never 0, but also that we cannot show that it *has to* take the value 0. To add such a generic sequence is exactly like adding a *Cohen real* [5] in forcing extension of set theory. A natural attempt for doing this will be to consider a *topological model* of type theory (sheaf model over Cantor space), extending the work [19] to type theory. However, while it is well understood how to represent universes in *presheaf* model [9], it has turned out to be surprisingly difficult to represent universes in *sheaf* models, as we learnt from works of Chuangjie Xu and Martin Escardo [21] and works of Thomas Streicher [17]. Our approach is here instead a purely *syntactical* description of a forcing extension of type theory (refining previous work of [7]), which contains a formal symbol for the generic sequence and a proof that it is absurd that this generic sequence is never 0, together with a *normalization* theorem, from which we can deduce that we *cannot* prove that this generic sequence has to take the value 0. Since this formal system is an extension of type theory, the independence of Markov's principle follows.

As stated in [11], which describes an elegant generalization of this principle in type theory, Markov's principle is an important technical tool for proving termination of computations, and thus can play a crucial role if type theory is extended with general recursion as in [6].

This paper is organized as follows. We first describe the rules of the version of type theory we are considering. This version can be seen as a simplified version of type theory as represented in the system Agda [16], and in particular, contrary to the work [7], we allow η-conversion, and we express conversion as *judgment.* Markov's principle can be formulated in a natural way in this formal system. We describe then the forcing extension of type theory, where we add a Cohen real. For proving normalization, we follow Tait's computability method [18, 15], but we have to consider an extension of this with a computability *relation* in order to interpret the conversion judgment. This can be seen as a forcing extension of the technique used in [1]. Using this computability argument, it is then possible to show that we cannot show that the generic sequence has to take the value 0. We end by a refinement of this method, giving a consistent extension of type theory where the *negation* of Markov's principle is provable.

[1] The paper [10] presents a model of the calculus of constructions using the idea of modified realizability, and it seems possible to use also this technique to interpret the type theory we consider and prove in this way the independence of Markov's principle.

[2] Some authors define independence in the stronger sense "A statement is independent of a formal system if neither the statement nor its negation is provable in the system", e.g. [13]. We will establish the independence of Markov's principle in this stronger sense with the help of known results from the literature.

2 Type theory and forcing extension

A dependent type theory is given by: A syntax describing the set objects of discourse, forms of judgments, and rules of inference for deriving valid judgments.

The syntax of our type theory is given by the grammar:

$$\begin{aligned} t, u, A, B \coloneqq\ & x \mid \bot\mathsf{rec}\ (\lambda x.A) \mid \mathsf{unitrec}\ (\lambda x.A)\, t \mid \mathsf{boolrec}\ (\lambda x.A)\, t\, u \mid \mathsf{natrec}\ (\lambda x.A)\, t\, u \\ & \mid U \mid N \mid N_0 \mid N_1 \mid N_2 \mid 0 \mid 1 \mid \mathsf{S}\, t \\ & \mid \Pi(x\!:\!A)B \mid \lambda x.t \mid t\, u \mid \Sigma(x\!:\!A)B \mid (t, u) \mid t.1 \mid t.2 \end{aligned}$$

The terms N_0, N_1, N_2, and N will denote , respectively, the empty type, the unit type, the type of booleans, and the type of natural numbers. The term U will denote the universe, i.e. the type of small types. We use the notation $\bar{n}$ as a short hand for the term $\mathsf{S}^n\, 0$, where S is the successor constructor of natural numbers.

2.1 Type system

We describe a type theory with one universe à la Russell, natural numbers, functional extensionality and surjective pairing, hereafter referred to as MLTT.[3] The type theory has the following judgment forms: 1. $\Gamma \vdash$. 2. $\Gamma \vdash A$. 3. $\Gamma \vdash t\!:\!A$. 4. $\Gamma \vdash A = B$. 5. $\Gamma \vdash t = u\!:\!A$. The first expresses that Γ is a well-formed contexts, the second that A is a type in the context Γ, and the third that t is a term of type A in the context Γ. The fourth and fifth express type and term equality respectively. Below we outline the inference rules of this type theory. We use the notation $F \to G$ for $\Pi(x : F)G$ when G doesn't depend on F and $\neg A$ for $A \to N_0$.

Natural numbers:

$$\frac{\Gamma \vdash}{\Gamma \vdash N} \qquad \frac{\Gamma \vdash}{\Gamma \vdash 0\!:\!N} \qquad \textsc{nat-suc}\ \frac{\Gamma \vdash n\!:\!N}{\Gamma \vdash \mathsf{S}\, n\!:\!N}$$

$$\textsc{natrec-I}\ \frac{\Gamma, x\!:\!N \vdash F \quad \Gamma \vdash a_0\!:\!F[0] \quad \Gamma \vdash g\!:\!\Pi(x\!:\!N)(F[x] \to F[\mathsf{S}\, x])}{\Gamma \vdash \mathsf{natrec}\ (\lambda x.F)\, a_0\, g\!:\!\Pi(x\!:\!N)F}$$

$$\textsc{natrec-0}\ \frac{\Gamma, x\!:\!N \vdash F \quad \Gamma \vdash a_0\!:\!F[0] \quad \Gamma \vdash g\!:\!\Pi(x\!:\!N)(F[x] \to F[\mathsf{S}\, x])}{\Gamma \vdash \mathsf{natrec}\ (\lambda x.F)\, a_0\, g\, 0 = a_0\!:\!F[0]}$$

$$\textsc{natrec-suc}\ \frac{\Gamma, x\!:\!N \vdash F \quad \Gamma \vdash a_0\!:\!F[0] \quad \Gamma \vdash n\!:\!N \quad \Gamma \vdash g\!:\!\Pi(x\!:\!N)(F[x] \to F[\mathsf{S}\, x])}{\Gamma \vdash \mathsf{natrec}\ (\lambda x.F)\, a_0\, g\, (\mathsf{S}\, n) = g\, n\, (\mathsf{natrec}\ (\lambda x.F)\, a_0\, g\, n)\!:\!F[\mathsf{S}\, n]}$$

$$\textsc{natrec-eq}\ \frac{\Gamma, x\!:\!N \vdash F = G \quad \Gamma \vdash a_0\!:\!F[0] \quad \Gamma \vdash g\!:\!\Pi(x\!:\!N)(F[x] \to F[\mathsf{S}\, x])}{\Gamma \vdash \mathsf{natrec}\ (\lambda x.F)\, a_0\, g = \mathsf{natrec}\ (\lambda x.G)\, a_0\, g\!:\!\Pi(x\!:\!N)F}$$

Booleans:

$$\frac{\Gamma \vdash}{\Gamma \vdash N_2} \qquad \frac{\Gamma \vdash}{\Gamma \vdash 0\!:\!N_2} \qquad \frac{\Gamma \vdash}{\Gamma \vdash 1\!:\!N_2}$$

$$\textsc{boolrec-I}\ \frac{\Gamma, x\!:\!N_2 \vdash F \quad \Gamma \vdash a_0\!:\!F[0] \quad \Gamma \vdash a_1\!:\!F[1]}{\Gamma \vdash \mathsf{boolrec}\ (\lambda x.F)\, a_0\, a_1\!:\!\Pi(x\!:\!N_2)F}$$

$$\textsc{boolrec-0}\ \frac{\Gamma, x\!:\!N_2 \vdash F \quad \Gamma \vdash a_0\!:\!F[0] \quad \Gamma \vdash a_1\!:\!F[1]}{\Gamma \vdash \mathsf{boolrec}\ (\lambda x.F)\, a_0\, a_1\, 0 = a_0\!:\!F[0]}$$

[3] This is a type system similar to Martin-löf's [15] except that we have η-conversion and surjective pairing.

$$\textsc{boolrec-1}\ \frac{\Gamma, x{:}N_2 \vdash F \quad \Gamma \vdash a_0 : F[0] \quad \Gamma \vdash a_1 : F[1]}{\Gamma \vdash \mathsf{boolrec}\ (\lambda x.F)\ a_0\ a_1\ 1 = a_1 : F[1]}$$

$$\textsc{boolrec-eq}\ \frac{\Gamma, x{:}N_2 \vdash F = G \quad \Gamma \vdash a_0 : F[0] \quad \Gamma \vdash a_1 : F[1]}{\Gamma \vdash \mathsf{natrec}\ (\lambda x.F)\ a_0\ a_1 = \mathsf{natrec}\ (\lambda x.G)\ a_0\ a_1 : \Pi(x{:}N_2)F}$$

Dependent functions:

$$\Pi\text{-I}\ \frac{\Gamma \vdash F \quad \Gamma, x{:}F \vdash G}{\Gamma \vdash \Pi(x{:}F)G} \qquad \Pi\text{-}\textsc{eq}\ \frac{\Gamma \vdash F = H \quad \Gamma, x{:}F \vdash G = E}{\Gamma \vdash \Pi(x{:}F)G = \Pi(x{:}H)E}$$

$$\lambda\text{-I}\ \frac{\Gamma, x{:}F \vdash t{:}G}{\Gamma \vdash \lambda x.t{:}\Pi(x{:}F)G} \qquad \textsc{fun-ap}\ \frac{\Gamma \vdash g{:}\Pi(x{:}F)G \quad \Gamma \vdash a{:}F}{\Gamma \vdash g\,a{:}G[a]} \qquad \beta\ \frac{\Gamma, x{:}F \vdash t{:}G \quad \Gamma \vdash a{:}F}{\Gamma \vdash (\lambda x.t)a = t[a] : G[a]}$$

$$\textsc{fun}\ \frac{\Gamma \vdash g{:}\Pi(x{:}F)G \quad \Gamma \vdash u = v{:}F}{\Gamma \vdash g\,u = g\,v{:}G[u]} \qquad \textsc{fun-eq}\ \frac{\Gamma \vdash h = g{:}\Pi(x{:}F)G \quad \Gamma \vdash u{:}F}{\Gamma \vdash h\,u = g\,u{:}G[u]}$$

$$\textsc{fun-ext}\ \frac{\Gamma \vdash h{:}\Pi(x{:}F)G \quad \Gamma \vdash g{:}\Pi(x{:}F)G \quad \Gamma, x{:}F \vdash h\,x = g\,x{:}G[x]}{\Gamma \vdash h = g{:}\Pi(x{:}F)G}$$

Dependent product:

$$\Sigma\text{-I}\ \frac{\Gamma \vdash F \quad \Gamma, x{:}F \vdash G}{\Gamma \vdash \Sigma(x{:}F)G} \qquad \Sigma\text{-}\textsc{eq}\ \frac{\Gamma \vdash F = H \quad \Gamma, x{:}F \vdash G = E}{\Gamma \vdash \Sigma(x{:}F)G = \Sigma(x{:}H)E}$$

$$\textsc{pr-I}\ \frac{\Gamma, x{:}F \vdash G \quad \Gamma \vdash a{:}F \quad \Gamma \vdash b{:}G[a]}{\Gamma \vdash (a,b){:}\Sigma(x{:}F)G} \qquad \textsc{pr-e-1}\ \frac{\Gamma \vdash t{:}\Sigma(x{:}F)G}{\Gamma \vdash t.1{:}F} \qquad \textsc{pr-e-2}\ \frac{\Gamma \vdash t{:}\Sigma(x{:}F)G}{\Gamma \vdash t.2{:}G[t.1]}$$

$$\mathsf{pr}_1\ \frac{\Gamma \vdash (t,u){:}\Sigma(x{:}F)G}{\Gamma \vdash (t,u).1 = t{:}F} \qquad \mathsf{pr}_2\ \frac{\Gamma \vdash (t,u){:}\Sigma(x{:}F)G}{\Gamma \vdash (t,u).2 = u{:}G[t]}$$

$$\textsc{pr-eq-1}\ \frac{\Gamma \vdash t = u{:}\Sigma(x{:}F)G}{\Gamma \vdash t.1 = u.1{:}F} \qquad \textsc{pr-eq-1}\ \frac{\Gamma \vdash t = u{:}\Sigma(x{:}F)G}{\Gamma \vdash t.2 = u.2{:}G[t.1]}$$

$$\textsc{pr-ext}\ \frac{\Gamma \vdash t{:}\Sigma(x{:}F)G \quad \Gamma \vdash u{:}\Sigma(x{:}F)G \quad \Gamma \vdash t.1 = u.1{:}F \quad \Gamma \vdash t.2 = u.2{:}G[t.1]}{\Gamma \vdash t = u{:}\Sigma(x{:}F)G}$$

Universe:

$$\frac{\Gamma \vdash}{\Gamma \vdash U} \qquad \frac{\Gamma \vdash F{:}U}{\Gamma \vdash F} \qquad \frac{\Gamma \vdash F = G{:}U}{\Gamma \vdash F = U} \qquad \frac{\Gamma \vdash}{\Gamma \vdash N{:}U} \qquad \frac{\Gamma \vdash}{\Gamma \vdash N_2{:}U}$$

$$\frac{\Gamma \vdash F : U \quad \Gamma, x{:}F \vdash G : U}{\Gamma \vdash \Pi(x{:}F)G : U} \qquad \frac{\Gamma \vdash F = H : U \quad \Gamma, x{:}F \vdash G = E : U}{\Gamma \vdash \Pi(x{:}F)G = \Pi(x{:}H)E{:}U}$$

$$\frac{\Gamma \vdash F : U \quad \Gamma, x{:}F \vdash G : U}{\Gamma \vdash \Sigma(x{:}F)G : U} \qquad \frac{\Gamma \vdash F = H : U \quad \Gamma, x{:}F \vdash G = E : U}{\Gamma \vdash \Sigma(x{:}F)G = \Sigma(x{:}H)E{:}U}$$

Congruence:

$$\frac{\Gamma\vdash t:F \quad \Gamma\vdash F=G}{\Gamma\vdash t:G} \qquad \frac{\Gamma\vdash t=u:F \quad \Gamma\vdash F=G}{\Gamma\vdash t=u:G}$$

$$\frac{\Gamma\vdash F}{\Gamma\vdash F=F} \qquad \frac{\Gamma\vdash F=G}{\Gamma\vdash G=F} \qquad \frac{\Gamma\vdash F=G \quad \Gamma\vdash G=H}{\Gamma\vdash F=H}$$

$$\frac{\Gamma\vdash t:F}{\Gamma\vdash t=t:F} \qquad \frac{\Gamma\vdash t=u:F}{\Gamma\vdash u=t:F} \qquad \frac{\Gamma\vdash t=u:F \quad \Gamma\vdash u=v:F}{\Gamma\vdash t=v:F}$$

For brevity we omitted the rules for the types N_0 and N_1.

The following four rules are admissible in the this type system [1]:

$$\frac{\Gamma\vdash a:A}{\Gamma\vdash A} \qquad \frac{\Gamma\vdash a=b:A}{\Gamma\vdash a:A} \qquad \frac{\Gamma,x:F\vdash G \quad \Gamma\vdash a=b:F}{\Gamma\vdash G[a]=G[b]} \qquad \frac{\Gamma,x:F\vdash t:G \quad \Gamma\vdash a=b:F}{\Gamma\vdash t[a]=t[b]:G[a]}$$

2.2 Markov's principle

Markov's principle can be represented in type theory by the type

$$\mathrm{MP} := \Pi(h:N\to N_2)[\neg\neg(\Sigma(x:N)\,\mathsf{IsZero}\,(h\,x))\to\Sigma(x:N)\,\mathsf{IsZero}\,(h\,x)]$$

where $\mathsf{IsZero}:N_2\to U$ is defined by $\mathsf{IsZero} := \lambda y.\mathsf{boolrec}\,(\lambda x.U)\,N_1\,N_0\,y$.

Note that $\mathsf{IsZero}\,(h\,n)$ is inhabited when $h\,n = 0$ and empty when $h\,n = 1$. Thus $\Sigma(x:N)\,\mathsf{IsZero}\,(h\,x)$ is inhabited if there is n such that $h\,n = 0$.

The main result of this paper is the following:

▶ **Theorem 2.1.** *There is no term t such that* $\mathrm{MLTT}\vdash t:\mathrm{MP}$.

An *extension* of MLTT is given by introducing new objects, judgment forms and derivation rules. This means in particular that any judgment valid in MLTT is valid in the extension. A *consistent* extension is one in which the type N_0 is uninhabited.

To show Theorem 2.1 we will form a consistent extension of MLTT with a new consant f where $\vdash \mathsf{f}:N\to N_2$ and $\neg\neg(\Sigma(x:N)\,\mathsf{IsZero}\,(\mathsf{f}\,x))\to\Sigma(x:N)\,\mathsf{IsZero}\,(\mathsf{f}\,x)$ is not derivable. Thus MP is not derivable in this extension and consequently not derivable in MLTT.

While this is sufficient to establish independence in the sense of non-derivability of MP. To establish the independence of MP in the stronger sense one also needs to show that $\neg$MP is not derivable in MLTT. This can achieved by reference to the work of Aczel [2] where it is shown that MLTT extended with $\vdash \mathsf{dne}:\Pi(A:U)(\neg\neg A\to A)$ is consistent. Since $h:N\to N_2, x:N\vdash \mathsf{IsZero}\,(h\,x):U$ we have $h:N\to N_2\vdash\Sigma(x:N)\,\mathsf{IsZero}\,(h\,x):U$. Thus

$$h:N\to N_2\vdash \mathsf{dne}\,(\Sigma(x:N)\,\mathsf{IsZero}\,(h\,x)):\neg\neg(\Sigma(x:N)\,\mathsf{IsZero}\,(h\,x))\to\Sigma(x:N)\,\mathsf{IsZero}\,(h\,x)$$

By λ abstraction we have $\vdash \lambda h.\mathsf{dne}\,(\Sigma(x:N)\,\mathsf{IsZero}\,(h\,x)):\mathrm{MP}$. We can then conclude that there is no term t such that $\mathrm{MLTT}\vdash t:\neg\mathrm{MP}$.

Finally, we will refine the result of Theorem 2.1 by building a consistent extension of MLTT where $\neg$MP is derivable.

2.3 Forcing extension

A *condition* p is a graph of a partial finite function from $\mathbb{N}$ to $\{0,1\}$. We denote by $\langle\rangle$ the empty condition. We write $p(n) = b$ when $(n,b)\in p$. We say q *extends* p (written $q\leqslant p$) if p is a subset of q. A condition can be thought of as a compact open in Cantor space $2^{\mathbb{N}}$. Two conditions p and q are *compatible* if $p\cup q$ is a condition and we write pq for $p\cup q$, otherwise they are *incompatible.* If $n \notin \mathrm{dom}(p)$ we write $p(n,0)$ for $p \cup \{(n,0)\}$ and $p(n,1)$ for $p \cup \{(n,1)\}$. We define the notion of *partition* corresponding to the notion of finite covering of a compact open in Cantor space.

▶ **Definition 2.2** (Partition)**.** We write $p \lhd p_1,\dots,p_n$ to say that $p_1,\dots,p_n$ is a partition of p and we define it as follows:

1. $p \lhd p$.
2. If $n \notin \mathrm{dom}(p)$ and $p(n,0) \lhd \dots,q_i,\dots$ and $p(n,1) \lhd \dots,r_j,\dots$ then $p \lhd \dots,q_i,\dots,r_j,\dots$.

Note that if $p \lhd p_1,\dots,p_n$ then p_i and p_j are incompatible whenever $i \neq j$. If moreover $q \leqslant p$ then $q \lhd \dots, qp_j, \dots$ where p_j is compatible with q.

We extend the given type theory by annotating the judgments with conditions, i.e. replacing each judgment $\Gamma \vdash J$ in the given type system with a judgment $\Gamma \vdash_p J$.

In addition we add the locality rule: $\textsc{loc}\dfrac{\Gamma \vdash_{p_1} J \quad \dots \quad \Gamma \vdash_{p_n} J}{\Gamma \vdash_p J} p \lhd p_1 \dots p_n$.

We add a term f for the generic point along with the introduction and conversion rules:

$$\textsf{f-I}\dfrac{\Gamma \vdash_p}{\Gamma \vdash_p \mathsf{f} : N \to N_2} \qquad \textsf{f-}\textsc{eval}\dfrac{\Gamma \vdash_p}{\Gamma \vdash_p \mathsf{f}\,\overline{n} = p(n) : N_2} n \in \mathrm{dom}(p) \ .$$

We add a term w and the rule: $\textsf{w-}\textsc{term}\dfrac{\Gamma \vdash_p}{\Gamma \vdash_p \mathsf{w} : \neg\neg(\Sigma(x:N)\,\mathsf{IsZero}\,(\mathsf{f}\,x))}$.

Since w inhabits $\neg\neg(\Sigma(x:N)\,\mathsf{IsZero}\,(\mathsf{f}\,x))$, our goal is then to show that no term inhabits $\Sigma(x:N)\,\mathsf{IsZero}(\mathsf{f}\,x)$.

It follows directly from the description of the forcing extension that:

▶ **Lemma 2.3.** *If* $\Gamma \vdash J$ *then* $\Gamma \vdash_p J$ *for all* p*. In particular, if* $\vdash t:A$ *then* $\vdash_p t:A$ *for all* p*.*

Note that if $q \leqslant p$ and $\Gamma \vdash_p J$ then $\Gamma \vdash_q J$ (monotonicity). A statement A (represented as a closed type) is derivable in this extension if $\vdash_{\langle\rangle} t:A$ for some t, which in turn implies $\vdash_p t:A$ for all p.

Similarly to [7] we can state a conservativity result for this extension. Let $\vdash g:N \to N_2$ and $\vdash v : \neg\neg(\Sigma(x:N)\,\mathsf{IsZero}\,(g\,x))$ be two terms of standard type theory. We say that g is compatible with a condition p if g is such that $\vdash g\,\overline{n} = b : N_2$ whenever $(n,b) \in p$ and $\vdash g\,\overline{n} = 0 : N_2$ otherwise. We say that v is compatible with a condition p if g is compatible with p and v is given by $v \coloneqq \lambda x.x\,(\overline{n}_p, 0)$ where n_p is the smallest natural number such that $n_p \notin \mathrm{dom}(p)$. To see that v is well typed, note that by design $\Gamma \vdash g\,\overline{n}_p = 0 : N_2$ thus $\Gamma \vdash \mathsf{IsZero}\,(g\,\overline{n}_p) = N_1$ and $\Gamma \vdash (\overline{n}_p, 0) : \Sigma(x:N)\mathsf{IsZero}\,(g\,x)$. We have then $\Gamma, x : \neg(\Sigma(y:N)\,\mathsf{IsZero}\,(g\,y)) \vdash x\,(\overline{n}_p,0) : N_0$ thus $\Gamma \vdash \lambda x.x\,(\overline{n}_p,0) : \neg\neg(\Sigma(y:N)\,\mathsf{IsZero}\,(g\,y))$.

▶ **Lemma 2.4** (Conservativity)**.** *Let* $\vdash g : N \to N_2$ *and* $\vdash v : \neg\neg(\Sigma(x:N)\,\mathsf{IsZero}\,(g\,x))$ *be compatible with* p*. If* $\Gamma \vdash_p J$ *then* $\Gamma[g/\mathsf{f}, v/\mathsf{w}] \vdash J[g/\mathsf{f}, v/\mathsf{w}]$*, i.e. replacing* f *with* g *then* w *with* v *we obtain a valid judgment in standard type theory. In particular, if* $\Gamma \vdash_{\langle\rangle} J$ *where neither* f *nor* w *occur in* Γ *or* J *then* $\Gamma \vdash J$ *is a valid judgment in standard type theory.*

Proof. The proof is by induction on the type system and it is straightforward for all the standard rules. For (f-EVAL) we have $(\mathsf{f}\,\overline{n})[g/\mathsf{f}, v/\mathsf{w}] \coloneqq g\,\overline{n}$ and since g is compatible with p we have $\Gamma[g/\mathsf{f}, v/\mathsf{w}] \vdash g\,\overline{n} = p(n) : N_2$ whenever $n \in \mathrm{dom}(p)$. For (w-TERM) we have $(\mathsf{w} : \neg\neg(\Sigma(x:N)\,\mathsf{IsZero}\,(\mathsf{f}\,x)))[g/\mathsf{f}, v/\mathsf{w}] \coloneqq (\mathsf{w} : \neg\neg(\Sigma(x:N)\,\mathsf{IsZero}\,(g\,x)))[v/\mathsf{w}] \coloneqq v : \neg\neg(\Sigma(x:N)\,\mathsf{IsZero}\,(g\,x))$. For (LOC) the statement follows from the observation that when g is compatible with p and $p \lhd p_1,\dots,p_n$ then g is compatible with exactly one p_i for $1 \leqslant i \leqslant n$. ◀

3 A Semantics of the forcing extension

In this section we outline a semantics for the forcing extension given in the previous section. We will interpret the judgments of type theory by computability predicates and relations defined by reducibility to computable weak head normal forms.

3.1 Reduction rules

We extend the β, ι conversion with $\mathsf{f}\,\overline{n} \Rightarrow_p b$ whenever $(n, b) \in p$. In order to ease the presentation of the proofs and definitions we introduce *evaluation contexts* following [20].

$$\begin{aligned}\mathbb{E} ::= &[\,] \mid \mathbb{E}\,u \mid \mathbb{E}.1 \mid \mathbb{E}.2 \mid \mathsf{S}\,\mathbb{E} \mid \mathsf{f}\,\mathbb{E}\\ &\bot\mathsf{rec}\;(\lambda x.C)\,\mathbb{E} \mid \mathsf{unitrec}\;(\lambda x.C)\,a\,\mathbb{E} \mid \mathsf{boolrec}\;(\lambda x.C)\,a_0\,a_1\,\mathbb{E} \mid \mathsf{natrec}\;(\lambda x.C)\,c_z\,g\,\mathbb{E}\end{aligned}$$

An expression $\mathbb{E}[e]$ is then the expression resulting from replacing the hole $[\,]$ by e. We reserve the symbols $\mathbb{E}$ and $\mathbb{C}$ for evaluation contexts. We have the following reduction rules:

$$\frac{}{\mathsf{unitrec}\;(\lambda x.C)\;c\;0 \to c} \qquad \frac{}{\mathsf{boolrec}\;(\lambda x.C)\,c_0\,c_1\,0 \to c_0} \qquad \frac{}{\mathsf{boolrec}\;(\lambda x.C)\,c_0\,c_1\,1 \to c_1}$$

$$\frac{}{\mathsf{natrec}\;(\lambda x.C)\,c_z\,g\,0 \to c_z} \qquad \frac{}{\mathsf{natrec}\;(\lambda x.C)\,c_z\,g\,(\mathsf{S}\,\overline{k}) \to g\,\overline{k}\,(\mathsf{natrec}\;(\lambda x.C)\,c_z\,g\,\overline{k})}$$

$$\frac{}{(\lambda x.t)\,a \to t[a/x]} \qquad \frac{}{(u,v).1 \to u} \qquad \frac{}{(u,v).2 \to v}$$

$$\frac{e \to e'}{e \to_p e'} \qquad \textsf{f-RED}\,\frac{k \in \mathrm{dom}(p)}{\mathsf{f}\,\overline{k} \to_p p(k)} \qquad \frac{e \to_p e'}{\mathbb{E}[e] \Rightarrow_p \mathbb{E}[e']}$$

Note that we reduce under S.

The relation $\Rightarrow$ is monotone, that is if $q \leqslant p$ and $t \Rightarrow_p u$ then $t \Rightarrow_q u$. We will also need to show that the reduction is local, i.e. if $p \lhd p_1, \dots, p_n$ and $t \Rightarrow_{p_i} u$ then $t \Rightarrow_p u$.

▶ **Lemma 3.1.** *If $m \notin \mathrm{dom}(p)$ and $t \to_{p(m,0)} u$ and $t \to_{p(m,1)} u$ then $t \to_p u$.*

Proof. By induction on the derivation of $t \to_{p(m,0)} u$. If $t \to_{p(m,0)} u$ is derived by (f-RED) then $t := \mathsf{f}\,\overline{k}$ and $u := p(m,0)(k)$ for some $k \in \mathrm{dom}(p(m,0))$. But since we also have a reduction $\mathsf{f}\,\overline{k} \to_{p(m,1)} u$, we have $p(m,1)(k) := u := p(m,0)(k)$ which could only be the case if $k \in \mathrm{dom}(p)$. Thus we have a reduction $\mathsf{f}\,\overline{k} \to_p u := p(k)$. Alternatively, we have a derivation $t \to u$, in which case we have $t \to_p u$ directly. ◀

▶ **Lemma 3.2.** *If $m \notin \mathrm{dom}(p)$ and $t \Rightarrow_{p(m,0)} u$ and $t \Rightarrow_{p(m,1)} u$ then $t \Rightarrow_p u$.*

Proof. From the reduction $t \Rightarrow_{p(k,0)} u$ we have $t := \mathbb{E}[e]$, $u := \mathbb{E}[e']$ and $e \to_{p(m,0)} e'$ for some context $\mathbb{E}$. But then we also have a reduction $\mathbb{E}[e] \Rightarrow_{p(m,1)} \mathbb{E}[e']$, thus $e \to_{p(m,1)} e'$. By Lemma 3.1, we have $e \to_p e'$ and thus $\mathbb{E}[e] \Rightarrow_p \mathbb{E}[e']$. ◀

▶ **Lemma 3.3.** *Let $q \leqslant p$. If $t \to_q u$ then either $t \to_p u$ or t has the form $\mathbb{E}[\mathsf{f}\,\overline{m}]$ for some $m \in \mathrm{dom}(q) \setminus \mathrm{dom}(p)$.*

Proof. By induction on the derivation of $t \to_q u$. If the reduction $t \to_q u$ has the form $\mathsf{f}\,\overline{k} \to_q q(k)$ then either $k \notin \mathrm{dom}(p)$ and the statement follows or $k \in \mathrm{dom}(p)$ and we have $t \to_p u$. Alternatively, we have $t \to u$ and immediately $t \to_p u$. ◀

▶ **Lemma 3.4.** *Let $q \leqslant p$. If $t \Rightarrow_q u$ then either $t \Rightarrow_p u$ or t has the form $\mathbb{E}[\mathsf{f}\,\overline{m}]$ for some $m \in \mathrm{dom}(q) \setminus \mathrm{dom}(p)$.*

Proof. If $t \Rightarrow_q u$ then $t \coloneqq \mathbb{E}[e]$, $u \coloneqq \mathbb{E}[e']$ and $e \rightarrow_q e'$ for some context $\mathbb{E}$. By Lemma 3.3 either $e \coloneqq \mathbb{C}[\mathsf{f}\,\overline{m}]$ for $m \notin \mathrm{dom}(p)$ and the statement follows or $e \rightarrow_p e'$ in which case we have $t \Rightarrow_p u$. ◀

▶ **Corollary 3.5.** *For any condition p and $m \notin \mathrm{dom}(p)$. Let $t \Rightarrow_{p(m,0)} u$ and $t \Rightarrow_{p(m,1)} v$. If $u \coloneqq v$ then $t \Rightarrow_p u$; otherwise, t has the form $\mathbb{E}[\mathsf{f}\,\overline{m}]$.*

Proof. Follows by Lemma 3.2 and Lemma 3.4. ◀

Next we define the relation $p \vdash t \Rightarrow u : A$ to mean $t \Rightarrow_p u$ and $\vdash_p t = u : A$ and we write $p \vdash A \Rightarrow B$ for $p \vdash A \Rightarrow B : U$. We note that it holds that if $p \vdash t \Rightarrow u : \Pi(x:F)G$ and $\vdash a : F$ then $p \vdash t\,a \Rightarrow u\,a : G[a]$ and if $p \vdash t \Rightarrow u : \Sigma(x:F)G$ then $p \vdash t.1 \Rightarrow u.1 : F$ and $p \vdash t.2 \Rightarrow u.2 : G[t.1]$. We define a closure for this relation as follows:

$$\frac{\vdash_p t : A}{p \vdash t \Rightarrow^* t : A} \qquad \frac{p \vdash t \Rightarrow u : A}{p \vdash t \Rightarrow^* u : A} \qquad \frac{p \vdash t \Rightarrow u : A \quad p \vdash u \Rightarrow^* v : A}{p \vdash t \Rightarrow^* v : A}$$

$$\frac{\vdash_p A}{p \vdash A \Rightarrow^* A} \qquad \frac{p \vdash A \Rightarrow B}{p \vdash A \Rightarrow^* B} \qquad \frac{p \vdash A \Rightarrow B \quad p \vdash B \Rightarrow^* C}{p \vdash A \Rightarrow^* C}$$

A term t is *in* p-whnf if whenever $t \Rightarrow_p u$ then $t \coloneqq u$. A whnf is canonical if it has the form 0,1,$\overline{n}$, $\lambda x.t$, f, w, $\bot\mathsf{rec}\ (\lambda x.C)$, $\mathsf{unitrec}\ (\lambda x.C)\, a$, $\mathsf{boolrec}\ (\lambda x.C)\, a_0\, a_1$, $\mathsf{natrec}\ (\lambda x.C)\, c_z\, g$, N_0,N_1,N_2,N,U,$\Pi(x:F)G$, or $\Sigma(x:F)G$. A p-whnf is proper if it is canonical or it is of the form $\mathbb{E}[\mathsf{f}\,\overline{k}]$ for $k \notin \mathrm{dom}(p)$.

We have the following corollaries to Lemma 3.2 and Corollary 3.5.

▶ **Corollary 3.6.** *Let $m \notin \mathrm{dom}(p)$. Let $p(m,0) \vdash t \Rightarrow_{p(m,0)} u : A$ and $p(m,1) \vdash t \Rightarrow_{p(m,1)} v : A$. If $u \coloneqq v$ then $p \vdash t \Rightarrow u : A$; otherwise t has the form $\mathbb{E}[\mathsf{f}\,\overline{m}]$.*

▶ **Corollary 3.7.** *If $p \vdash t \Rightarrow u : A$ and $q \leqslant p$ then $q \vdash t \Rightarrow u : A$. If $p \lhd p_1, \ldots, p_n$ and $p_i \vdash t \Rightarrow u : A$ for all i then $p \vdash t \Rightarrow u : A$.*

Proof. Let $q \leqslant p$. If $t \Rightarrow_p u$ we have $t \Rightarrow_q u$ and if $\vdash_p t = u : A$ then $\vdash_q t = u : A$. Thus $q \vdash t \Rightarrow u : A$ whenever $p \vdash t \Rightarrow u : A$. Let $p \lhd p_1, \ldots, p_n$. If for all i, $t \Rightarrow_{p_i} u : A$ then from Lemma 3.2, by induction on the partition, we have $t \Rightarrow_p u : A$. If $\vdash_{p_i} t = u : A$ for all i, then $\vdash_p t = u : A$. Thus we have $p \vdash t \Rightarrow u : A$ whenever $p_i \vdash t \Rightarrow u : A$ for all i. ◀

From the above we can show that closure $\Rightarrow^*$ is monotone, it is not however local.

For a closed term $\vdash_p t : A$, we say that t *has* a p-whnf if $p \vdash t \Rightarrow^* u : A$ and u is in p-whnf. If moreover u is canonical, respectively proper, we say that t has a canonical, respectively proper, p-whnf. Since the reduction relation is deterministic we have

▶ **Lemma 3.8.** *A term $\vdash_p t : A$ has at most one p-whnf.*

▶ **Corollary 3.9.** *Let $\vdash_p t : A$ and $m \notin \mathrm{dom}(p)$. If t has proper $p(m,0)$-whnf and a proper $p(m,1)$-whnf then t has a proper p-whnf.*

Proof. Let $p(m,0) \vdash t \Rightarrow^* u : A$ and $p(m,1) \vdash t \Rightarrow^* v : A$ with u in proper $p(m,0)$-whnf and v in proper $p(m,1)$-whnf. If $t \coloneqq u$ or $t \coloneqq v$ then t is already in proper p-whnf. Alternatively we have reductions $p(m,0) \vdash t \Rightarrow u_1 : A$ and $p(m,1) \vdash t \Rightarrow v_1 : A$. By Corollary 3.6 either t is in proper p-whnf or $u_1 \coloneqq v_1$ and $p \vdash t \Rightarrow u_1 : A$. It then follows by induction that u_1, and thus t, has a proper p-whnf. ◀

3.2 Computability predicate and relation

We define inductively a forcing relation $p \Vdash A$ to express that a type A is computable at p. Mutually by recursion we define relations $p \Vdash a : A$, $p \Vdash A = B$, and $p \Vdash a = b : A$. The definition fits the generalized mutual induction-recursion schema [8][4].

▶ **Definition 3.10** (Computibility predicate and relation).

$(\mathbf{F_{N_0}})$ If $p \vdash A \Rightarrow^* N_0$ then $p \Vdash A$.

1. $p \Vdash t : A$ does not hold for all t.
2. $p \Vdash t = u : A$ does not hold for all t and u.
3. If $p \Vdash B$ then $p \Vdash A = B$ if
   - (i) $p \vdash B \Rightarrow^* N_0$.
   - (ii) $p \vdash B \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}]$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash A = B$ for all $i \in \{0,1\}$.

$(\mathbf{F_{N_1}})$ If $p \vdash A \Rightarrow^* N_1$ then $p \Vdash A$.

1. $p \Vdash t : A$ if
   - (i) $p \vdash t \Rightarrow^* 0 : A$.
   - (ii) $p \vdash t \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}] : A$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash t : A$ for all $i \in \{0,1\}$.
3. If $p \Vdash t : A$ and $p \Vdash u : A$ then $p \Vdash t = u : A$ if
   - (i) $p \vdash t \Rightarrow^* 0 : A$ and $p \vdash u \Rightarrow^* 0 : A$.
   - (ii) $p \vdash t \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}] : A$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash t = u : A$ for all $i \in \{0,1\}$.
   - (iii) $p \vdash u \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}] : A$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash t = u : A$ for all $i \in \{0,1\}$.
4. If $p \Vdash B$ then $p \Vdash A = B$ if
   - (i) $p \vdash B \Rightarrow^* N_1$.
   - (ii) $p \vdash B \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}]$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash A = B$ for all $i \in \{0,1\}$.

$(\mathbf{F_{N_2}})$ If $p \vdash A \Rightarrow^* N_2$ then $p \Vdash A$.

1. $p \Vdash t : A$ if
   - (i) $p \vdash t \Rightarrow^* b : A$ for some $b \in \{0,1\}$.
   - (ii) $p \vdash t \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}] : A$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash t : A$ for all $i \in \{0,1\}$.
3. If $p \Vdash t : A$ and $p \Vdash u : A$ then $p \Vdash t = u : A$ if
   - (i) $p \vdash t \Rightarrow^* b : A$ and $p \vdash u \Rightarrow^* b : A$ for some $b \in \{0,1\}$.
   - (ii) $p \vdash t \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}] : A$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash t = u : A$ for all $i \in \{0,1\}$.
   - (iii) $p \vdash u \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}] : A$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash t = u : A$ for all $i \in \{0,1\}$.
4. If $p \Vdash B$ then $p \Vdash A = B$ if
   - (i) $p \vdash B \Rightarrow^* N_2$.
   - (ii) $p \vdash B \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}]$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash A = B$ for all $i \in \{0,1\}$.

$(\mathbf{F_{N}})$ If $p \vdash A \Rightarrow^* N$ then $p \Vdash A$.

1. $p \Vdash t : A$ if
   - (i) $p \vdash t \Rightarrow^* \overline{n} : A$ for some $n \in \mathbb{N}$.
   - (ii) $p \vdash t \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}] : A$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash t : A$ for all $i \in \{0,1\}$.
3. If $p \Vdash t : A$ and $p \Vdash u : A$ then $p \Vdash t = u : A$ if
   - (i) $p \vdash t \Rightarrow^* \overline{n} : A$ and $p \vdash u \Rightarrow^* \overline{n} : A$ for some $n \in \mathbb{N}$.
   - (ii) $p \vdash t \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}] : A$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash t = u : A$ for all $i \in \{0,1\}$.
   - (iii) $p \vdash u \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}] : A$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash t = u : A$ for all $i \in \{0,1\}$.
4. If $p \Vdash B$ then $p \Vdash A = B$ if
   - (i) $p \vdash B \Rightarrow^* N$.
   - (ii) $p \vdash B \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}]$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash A = B$ for all $i \in \{0,1\}$.

[4] However, for the canonical proof below we actually need something weaker than an inductive-recursive definition (arbitrary fixed-point instead of *least* fixed-point), reflecting the fact that the universe is defined in an open way [15].

$(\mathbf{F_\Pi})$ If $p \vdash A \Rightarrow^* \Pi(x:F)G$ then $p \Vdash A$ if $p \Vdash F$ and for all $q \leqslant p$, $q \Vdash G[a]$ whenever $q \Vdash a:F$ and $q \Vdash G[a] = G[b]$ whenever $q \Vdash a = b:F$.

1. If $\vdash_p f:A$ then $p \Vdash f:A$ if for all $q \leqslant p$, $q \Vdash f\,a : G[a]$ whenever $q \Vdash a:F$ and $q \Vdash f\,a = f\,b : G[a]$ whenever $q \Vdash a = b:F$.
2. If $p \Vdash f:A$ and $p \Vdash g:A$ then $p \Vdash f = g:A$ if for all $q \leqslant p$, $q \Vdash f\,a = g\,a : G[a]$ whenever $q \Vdash a:F$.
3. If $p \Vdash B$ then $p \Vdash A = B$ if
   - **(i)** $\vdash_p A = B$ and $p \vdash B \Rightarrow^* \Pi(x:H)E$ and $p \Vdash F = H$ and for all $q \leqslant p$, $q \Vdash G[a] = E[a]$ whenever $q \Vdash a:F$.
   - **(ii)** $p \vdash B \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}]$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash A = B$ for all $i \in \{0,1\}$.

$(\mathbf{F_\Sigma})$ If $p \vdash A \Rightarrow^* \Sigma(x:F)G$ then $p \Vdash A$ if $p \Vdash F$ and for all $q \leqslant p$, $q \Vdash G[a]$ whenever $q \Vdash a:F$ and $q \Vdash G[a] = G[b]$ whenever $q \Vdash a = b:F$.

1. If $\vdash_p t:A$ then $p \Vdash t:A$ if $p \Vdash t.1:F$ and $p \Vdash t.2:G[t.1]$.
2. If $p \Vdash t:A$ and $p \Vdash u:A$ then $p \Vdash t = u:A$ if $p \Vdash t.1 = u.1:F$ and $p \Vdash t.2 = u.2:G[t.1]$.
3. If $p \Vdash B$ then $p \Vdash A = B$ if
   - **(i)** $\vdash_p A = B$ and $p \vdash B \Rightarrow^* \Sigma(x:H)E$ and $p \Vdash F = H$ and for all $q \leqslant p$, $q \Vdash G[a] = E[a]$ whenever $q \Vdash a:F$.
   - **(ii)** $p \vdash B \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}]$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash A = B$ for all $i \in \{0,1\}$.

$(\mathbf{F_U})$ If $p \vdash A \Rightarrow^* U$ then $p \Vdash A$.

1. $p \Vdash C:A$ if
   - **(i)** $p \vdash C \Rightarrow^* M:A$ for $M \in \{N_0, N_1, N_2, N\}$.
   - **(ii)** $p \vdash C \Rightarrow^* \Pi(x:F)G:A$ and $p \Vdash F:A$ and for all $q \leqslant p$, $q \Vdash G[a]:A$ whenever $q \Vdash a:F$ and $q \Vdash G[a] = G[b]:A$ whenever $q \Vdash a = b:F$.
   - **(iii)** $p \vdash C \Rightarrow^* \Sigma(x:F)G:A$ and $p \Vdash F:A$ and for all $q \leqslant p$, $q \Vdash G[a]:A$ whenever $q \Vdash a:F$ and $q \Vdash G[a] = G[b]:A$ whenever $q \Vdash a = b:F$.
   - **(iv)** $p \vdash C \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}]:A$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash C:A$ for all $i \in \{0,1\}$.
5. If $p \Vdash C:A$ and $p \Vdash D:A$ then $p \Vdash C = D:A$ if
   - **(i)** $p \vdash C \Rightarrow^* M:A$ and $D \Rightarrow^* M:A$ for $M \in \{N_0, N_1, N_2, N\}$.
   - **(ii)** $p \vdash C \Rightarrow^* \Pi(x:F)G:A$ and $p \vdash D \Rightarrow^* \Pi(x:H)E:A$ and $p \Vdash F = H:A$ and for all $q \leqslant p$, $q \Vdash G[a] = E[a]:A$ whenever $q \Vdash a:F$.
   - **(iii)** $p \vdash C \Rightarrow^* \Sigma(x:F)G:A$ and $p \vdash D \Rightarrow^* \Sigma(x:H)E:A$ and $p \Vdash F = H:A$ and for all $q \leqslant p$, $q \Vdash G[a] = E[a]:A$ whenever $q \Vdash a:F$.
   - **(iv)** $p \vdash C \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}]:A$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash C = D:A$ for all $i \in \{0,1\}$.
   - **(v)** $p \vdash D \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}]:A$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash C = D:A$ for all $i \in \{0,1\}$.
6. If $p \Vdash B$ then $p \Vdash A = B$ if $p \vdash B \Rightarrow^* U$.

$(\mathbf{F_{Loc}})$ If $p \vdash A \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}]$ for some $m \notin \mathrm{dom}(p)$ and $p(m,i) \Vdash A$ for all $i \in \{0,1\}$ then $p \Vdash A$.

1. If $p(m,i) \Vdash t:A$ for all $i \in \{0,1\}$ then $p \Vdash t:A$.
2. If $p \Vdash t:A$ and $p \Vdash u:A$ and $p(m,i) \Vdash t:A$ for all $i \in \{0,1\}$ then $p \Vdash t = u:A$.
3. If $p \Vdash B$ then $p \Vdash A = B$ if $p(m,i) \Vdash A = B$ for all $i \in \{0,1\}$.

We note from the definition that when $p \Vdash A = B$ then $p \Vdash A$ and $p \Vdash B$, when $p \Vdash a:A$ then $p \Vdash A$ and when $p \Vdash a = b:A$ then $p \Vdash a:A$ and $p \Vdash b:A$. We remark also if $p \vdash A \Rightarrow^* U$ then $A \coloneqq U$ since we have only one universe.

The clause $(\mathrm{F_{Loc}})$ gives semantics to *variable types.* For example, if $p \coloneqq \{(0,0)\}$ and $q \coloneqq \{(0,1)\}$ the type $R \coloneqq \mathsf{boolrec}\,(\lambda x.U)\,N_1\,N\,(\mathsf{f}\,0)$ has reductions $p \vdash R \Rightarrow^* N_1$ and $q \vdash R \Rightarrow^* N$. Thus $p \Vdash R$ and $q \Vdash R$ and since $\langle\rangle \lhd p, q$ we have $\langle\rangle \Vdash R$.

Immediately from Definition 3.10 we get:

▶ **Lemma 3.11.** *If $p \Vdash A$ then $\vdash_p A$. If $p \Vdash a\!:\!A$ then $\vdash_p a\!:\!A$.*

▶ **Lemma 3.12.** *If $p \Vdash A$ then there is a partition $p \lhd p_1, \dots, p_n$ where A has a canonical p_i-whnf for all i.*

Proof. The statement follows from the definition by induction on the derivation of $p \Vdash A$ ◀

▶ **Corollary 3.13.** *Let $p \lhd p_1, \dots, p_n$. If $p_i \Vdash A$ for all i then A has a proper p-whnf.*

Proof. Follows from Lemma 3.12 and Corollary 3.9 by induction on the partition. ◀

▶ **Lemma 3.14.** *If $p \Vdash A$ and $q \leqslant p$ then $q \Vdash A$.*

Proof. Let $p \Vdash A$ and $q \leqslant p$. By induction on the derivation of $p \Vdash A$:

($\mathrm{F_N}$) Since $p \vdash A \Rightarrow^* N$ and the reduction relation is monotone we have $q \vdash A \Rightarrow^* N$, thus $q \Vdash A$. The statement follows similarly for ($\mathrm{F_{N_0}}$), ($\mathrm{F_{N_1}}$), ($\mathrm{F_{N_2}}$) and ($\mathrm{F_U}$).

($\mathrm{F_\Pi}$) Let $p \vdash A \Rightarrow^* \Pi(x\!:\!F)G$. Since $p \Vdash F$, by induction $q \Vdash F$. Let $s \leqslant q$, we have then $s \leqslant p$. It then follows from $p \Vdash A$ that $s \Vdash G[a]$ whenever $s \Vdash a\!:\!F$ and $s \Vdash G[a] = G[b]$ whenever $s \Vdash a = b\!:\!F$. Thus $q \Vdash A$. The statement follows similarly for ($\mathrm{F_\Sigma}$).

($\mathrm{F_{Loc}}$) Let $p \vdash A \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}]$. If $m \in \mathrm{dom}(q)$ then $q \leqslant p(m,0)$ or $q \leqslant p(m,1)$ and since $p(m,i) \Vdash A$, by induction $q \Vdash A$. Alternatively, $q \vdash A \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}]$. But $q \lhd q(m,0), q(m,1)$ and $q(m,i) \leqslant p(m,i)$. By induction $q(m,i) \Vdash A$ for all $i \in \{0,1\}$ and thus $q \Vdash A$. ◀

▶ **Lemma 3.15.** *If $p \Vdash t\!:\!A$ and $q \leqslant p$ then $q \Vdash t\!:\!A$.*

Proof. Let $p \Vdash t\!:\!A$ and $q \leqslant p$. By induction on the derivation of $p \Vdash A$.

($\mathrm{F_N}$) Since $p \vdash A \Rightarrow^* N$ then $q \vdash A \Rightarrow^* N$. By induction on the derivation of $p \Vdash t\!:\!A$. If $p \vdash t \Rightarrow^* \overline{n} : A$ for $n \in \mathbb{N}$ then $q \vdash t \Rightarrow^* \overline{n} : A$, hence, $q \Vdash t : A$. Alternatively, $p \vdash t \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{k}]\!:\!A$ for some $k \notin \mathrm{dom}(p)$ and $p(k,b) \Vdash t\!:\!A$ for all $b \in \{0,1\}$. If $k \in \mathrm{dom}(q)$ then $q \leqslant p(k,1)$ or $q \leqslant p(k,0)$ and in either case, by induction, $q \Vdash t\!:\!A$. Otherwise, we have $q(k,b) \leqslant p(k,b)$ and by induction $q(k,b) \Vdash t\!:\!A$ for all b. By the definition $q \Vdash t\!:\!A$. The statement follows similarly for ($\mathrm{F_{N_0}}$), ($\mathrm{F_{N_1}}$), and ($\mathrm{F_{N_2}}$).

($\mathrm{F_U}$) We can show the statement by a proof similar to that of Lemma 3.14.

($\mathrm{F_\Pi}$) Let $p \vdash A \Rightarrow^* \Pi(x\!:\!F)G$. We have $q \vdash A \Rightarrow^* \Pi(x\!:\!F)G$. From $\vdash_p t\!:\!A$ we have $\vdash_q t\!:\!A$. Let $r \leqslant q$. If $r \Vdash a\!:\!F$ then since $r \leqslant p$ we have $r \Vdash t\,a\!:\!G[a]$. Similarly if $r \Vdash a = b\!:\!F$ then $r \Vdash t\,a = t\,b\!:\!G[a]$. Thus $q \Vdash t\!:\!A$.

($\mathrm{F_\Sigma}$) Let $p \vdash A \Rightarrow^* \Sigma(x\!:\!F)G$. We have $q \vdash A \Rightarrow^* \Sigma(x\!:\!F)G$. From $\vdash_p t\!:\!A$ we have $\vdash_q t\!:\!A$. Since $p \Vdash t : A$ we have $p \Vdash t.1 : F$ and $p \Vdash t.2 : G[t.1]$. By induction $q \Vdash t.1 : F$ and $q \Vdash t.2\!:\!G[t.1]$, thus $q \Vdash t\!:\!A$.

($\mathrm{F_{Loc}}$) Let $p \vdash A \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{k}]$ for some $k \notin \mathrm{dom}(p)$. Since $p \Vdash t\!:\!A$ we have $p(k,b) \Vdash t\!:\!A$ for all $b \in \{0,1\}$. If $k \in \mathrm{dom}(q)$ then $q \leqslant p(k,0)$ or $q \leqslant p(k,1)$ and by induction $q \Vdash t\!:\!A$. Otherwise, $q \vdash A \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{k}]$ and since $q(k,b) \leqslant p(k,b)$, by induction, $q(k,b) \Vdash t\!:\!A$ for all b. By definition $q \Vdash t\!:\!A$. ◀

Using similar arguments we can also show the following two statements:

▶ **Lemma 3.16.** *Let $p \Vdash A$. If $p \Vdash A = B$ and $q \leqslant p$ then $q \Vdash A = B$.*

▶ **Lemma 3.17.** *Let $p \Vdash A$. If $p \Vdash t = u\!:\!A$ and $q \leqslant p$ then $q \Vdash t = u\!:\!A$.*

We collect the results of Lemmas 3.14, Lemma 3.15, Lemma 3.17, and Lemma 3.16 in the following corollary.

▶ **Corollary 3.18** (Monotonicity). *If $p \Vdash J$ and $q \leqslant p$ then $q \Vdash J$.*

We write $\Vdash J$ when $\langle\rangle \Vdash J$. By monotonicity $\Vdash J$ iff $p \Vdash J$ for all p.

▶ **Lemma 3.19.** *If $p(m,0) \Vdash A$ and $p(m,1) \Vdash A$ for some $m \notin \mathrm{dom}(p)$ then $p \Vdash A$.*

Proof. By Corollary 3.13, either A has a canonical p-whnf or $p \vdash A \Rightarrow^* \mathbb{E}[\mathsf{f}\,\bar{k}]$ for $k \notin \mathrm{dom}(p)$.

- If $p \vdash A \Rightarrow^* M$ with $M \in \{N_0, N_1, N_2, N\}$ then we have immediately that $p \Vdash A$.
- If $p \vdash A \Rightarrow^* M$ with M of the form $\Pi(x:F)G$ or $\Sigma(x:F)G$ then $p(m,b) \vdash A \Rightarrow^* M$ for all $b \in \{0,1\}$. Since $p(m,b) \Vdash A$ we have $p(m,b) \Vdash F$ for all b and by induction $p \Vdash F$. Let $q \leqslant p$ and $q \Vdash a:F$. If $m \in \mathrm{dom}(q)$ then $q \leqslant p(m,b)$ for some $b \in \{0,1\}$. Assume, w.l.o.g, $q \leqslant p(m,0)$. Since $p(m,0) \Vdash A$ we have by the definition that $q \Vdash G[a]$. Alternatively, if $m \notin \mathrm{dom}(q)$ we have a partition $q \lhd q(m,0), q(m,1)$. By monotonicity $q(m,b) \Vdash a:F$, and since $q(m,b) \leqslant p(m,b)$, we have $q(m,b) \Vdash G[a]$ for all $b \in \{0,1\}$. By induction $q \Vdash G[a]$. Similarly we can show $q \Vdash G[a] = G[b]$ whenever $q \Vdash a = b:F$.
- Alternatively, $p \vdash A \Rightarrow^* \mathbb{E}[\mathsf{f}\,\bar{k}]$ for some $k \notin \mathrm{dom}(p)$. If $k = m$ then by the definition $p \Vdash A$. Otherwise, $p(m,0) \vdash A \Rightarrow^* \mathbb{E}[\mathsf{f}\,\bar{k}]$ and by the definition $p(m,0)(k,b) \Vdash A$ for all $b \in \{0,1\}$. Similarly, $p(m,1)(k,b) \Vdash A$ for all $b \in \{0,1\}$. But $p(k,b) \lhd p(m,0)(k,b), p(m,1)(k,b)$. By induction $p(k,b) \Vdash A$ for all $b \in \{0,1\}$ and thus $p \Vdash A$. ◀

Similarly we can show the following two statements:

▶ **Lemma 3.20.** *1. If $p(m,0) \Vdash t:A$ and $p(m,1) \Vdash t:A$ for some $m \notin \mathrm{dom}(p)$ then $p \Vdash t:A$.*
2. If $p(m,0) \Vdash t = u:A$ and $p(m,1) \Vdash t = u:A$ for some $m \notin \mathrm{dom}(p)$ then $p \Vdash t = u:A$.

▶ **Lemma 3.21.** *If $p(m,0) \Vdash A = B$ and $p(m,1) \Vdash A = B$ for some $m \notin \mathrm{dom}(p)$ then $p \Vdash A = B$.*

▶ **Corollary 3.22** (Local character). *If $p \lhd p_1, \ldots, p_n$ and $p_i \Vdash J$ for all i then $p \Vdash J$.*

Proof. Follows from Lemma 3.19, Lemma 3.20, and Lemma 3.21 by induction. ◀

▶ **Lemma 3.23.** *Let $p \vdash A \Rightarrow^* M$ where $M \in \{N_1, N_2, N\}$. If $p \Vdash a:A$ then there is a partition $p \lhd p_1, \ldots, p_n$ where a has a canonical p_i-whnf for all i. If $p \Vdash a = b:A$ then there is a partition $p \lhd q_1, \ldots, q_m$ where a and b have the same canonical q_j-whnf for each j.*

Proof. Follows by induction from the definition. ◀

▶ **Lemma 3.24.** *Let $p \Vdash A = B$.*
1. *If $p \Vdash t:A$ then $p \Vdash t:B$ and if $p \Vdash u:B$ then $p \Vdash u:A$.*
2. *If $p \Vdash t = u:A$ then $p \Vdash t = u:B$ and if $p \Vdash v = w:B$ then $p \Vdash v = w:A$.*

Proof. By induction on the derivation of $p \Vdash A$.

($\mathrm{F_N}$) By induction on the derivation of $p \Vdash A = B$. (i) Let $p \vdash B \Rightarrow^* N$ then the statement follows directly. (ii) Let $p \vdash B \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{m}]$ for $m \notin \mathrm{dom}(p)$ and $p(m,b) \Vdash A = B$ for all $b \in \{0,1\}$. Let $p \Vdash t:A$. By monotonicity $p(m,b) \Vdash t:A$ and by induction $p(m,b) \Vdash t:B$ for all b. By the definition $p \Vdash t:B$. Let $p \Vdash u:B$. By monotonicity $p(m,b) \Vdash u:B$ and $p(m,b) \Vdash A = B$. By induction $p(m,b) \Vdash u:A$ for all b. By local character $p \Vdash u:A$. Similarly we can show the second statement. The statement follows similarly for ($\mathrm{F_{N_1}}$) and ($\mathrm{F_{N_2}}$).

($\mathbf{F}_\Pi$) Let $p \vdash A \Rightarrow^* \Pi(x:F)G$. By induction on the derivation of $p \Vdash A = B$. (i) Let $\vdash_p A = B$ and $p \vdash B \Rightarrow^* \Pi(x:H)E$ and $p \Vdash F = H$ and for all $q \leqslant p$, $q \Vdash G[a] = E[a]$ whenever $q \Vdash a:F$. If $p \Vdash f:A$ then $\vdash_p f:A$, thus $\vdash_p f:B$. Let $q \leqslant p$ and $q \Vdash u:H$. By monotonicity $q \Vdash F = H$. By induction $q \Vdash u:F$, hence, $q \Vdash f\,u:G[u]$ and by induction $q \Vdash f\,u:E[u]$. Similarly, $q \Vdash f\,u = f\,v:E[u]$ whenever $q \Vdash u = v:H$. Thus $p \Vdash f:B$. Similarly, if $p \Vdash g:B$ we get $p \Vdash g:A$. (ii) Let $p \vdash B \Rightarrow^* \mathbb{E}[\mathsf{f}, \overline{k}]$ and $p(k,b) \Vdash A = B$ for all $b \in \{0,1\}$. If $p \Vdash f:A$ then by monotonicity $p(k,b) \Vdash f:A$ and by induction $p(k,b) \Vdash f:B$ for all b. By the definition $p \Vdash f:B$. If on the other hand $p \Vdash g:B$ then by definition $p(k,b) \Vdash g:B$ and by induction $p(k,b) \Vdash g:A$ for all b. By local character $p \Vdash g:A$. Similarly we can show the second statement.

($\mathbf{F}_\Sigma$) Let $p \vdash A \Rightarrow^* \Sigma(x:F)G$. By induction on the derivation of $p \Vdash A = B$. (i) Let $\vdash_p A = B$ and $p \vdash B \Rightarrow^* \Sigma(x:H)E$ and $p \Vdash F = H$ and for all $q \leqslant p$, $q \Vdash G[a] = E[a]$ whenever $q \Vdash a:F$. If $p \Vdash t:A$ then $\vdash_p t:A$, thus $\vdash_p t:B$. Since $p \Vdash t.1:F$, by induction $p \Vdash t.1:H$. Since $p \Vdash t.2:H[t.1]$, by induction $p \Vdash t.2:E[t.1]$. Thus $p \Vdash t:B$. Similarly if $p \Vdash u:B$ we have $p \Vdash u:A$. (ii) Let $p \vdash B \Rightarrow^* \mathbb{E}[\mathsf{f}, \overline{k}]$ and $p(k,b) \Vdash A = B$ for all $b \in \{0,1\}$. If $p \Vdash t:A$ then by monotonicity $p(k,b) \Vdash t:A$ and by induction $p(k,b) \Vdash t:B$ for all b. By the definition $p \Vdash f:B$. If on the other hand $p \Vdash g:B$ then by definition $p(k,b) \Vdash g:B$ and by induction $p(k,b) \Vdash g:A$ for all b. By local character $p \Vdash g:A$. Similarly we can show the second statement.

($\mathbf{F}_\mathbf{U}$) Since $p \Vdash A = B$, we have $p \vdash B \Rightarrow^* U$ and the statements follow directly.

($\mathbf{F}_\mathbf{Loc}$) Let $p \vdash A \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{k}]$ for some $k \notin \mathrm{dom}(p)$. Since $p \Vdash A = B$, we have $p(k,b) \Vdash A = B$ for all $b \in \{0,1\}$. If $p \Vdash t:A$ then $p(k,b) \Vdash t:A$ and by induction $p(k,b) \Vdash t:B$ for all b. By the definition $p \Vdash t:B$. If $p \Vdash u:B$ then $p(k,b) \Vdash u:B$ and by induction $p(k,b) \Vdash u:A$ for all b. By local character $p \Vdash u:A$. Similarly we can show the second statement. ◀

From the above results we can show that the relations $p \Vdash - = -$ and $p \Vdash - = -:A$ are equivalence relations. We omit the proof here.

▶ **Lemma 3.25.**
Reflexivity: *If $p \Vdash A$ then $p \Vdash A = A$ and if $p \Vdash t:A$ then $p \Vdash t = t:A$.*
Symmetry: *If $p \Vdash A = B$ then $p \Vdash B = A$ and if $p \Vdash t = u:A$ then $p \Vdash u = t:A$.*
Transitivity: *If $p \Vdash A = B$ and $p \Vdash B = C$ then $p \Vdash A = C$ and if $p \Vdash t = u:A$ and $p \Vdash u = v:A$ then $p \Vdash t = v:A$.*

4 Soundness

In this section we show that the type theory described in Section 2 is sound with respect to the semantics described in Section 3. That is, we aim to show that for any judgment J whenever $\vdash_p J$ then $p \Vdash J$.

▶ **Lemma 4.1.** *If $p \vdash A \Rightarrow^* B$ and $p \Vdash B$ then $p \Vdash A$ and $p \Vdash A = B$.*

Proof. Follows from the definition. ◀

▶ **Lemma 4.2.** *Let $p \Vdash A$. If $p \vdash t \Rightarrow u:A$ and $p \Vdash u:A$ then $p \Vdash t:A$ and $p \Vdash t = u:A$.*

Proof. Let $p \vdash t \Rightarrow u:A$ and $p \Vdash u:A$. By induction on the derivation of $p \Vdash A$.

($\mathbf{F}_\mathbf{U}$) That is, $p \vdash A \Rightarrow^* U$. The statement follows similarly to Lemma 4.1.

($\mathbf{F}_\mathbf{N}$) By induction on the derivation of $p \Vdash u:A$. If $p \vdash u \Rightarrow^* \overline{n}:N$ for some $n \in \mathbb{N}$ then $p \vdash t \Rightarrow^* \overline{n}:N$ and the statement follows by the definition. If $p \vdash u \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{k}]:A$ for $k \notin \mathrm{dom}(p)$ and $p(k,b) \Vdash u:A$ for all $b \in \{0,1\}$ then since $p(k,b) \vdash t \Rightarrow u:A$, by induction,

$p(k,b) \Vdash t : A$ and $p(k,b) \Vdash t = u : A$. By the definition $p \Vdash t : A$ and $p \Vdash t = u : A$. The statement follows similarly for $(\mathrm{F}_{\mathrm{N}_1})$, $(\mathrm{F}_{\mathrm{N}_2})$.

$(\mathbf{F}_\Pi)$ Let $p \vdash A \Rightarrow^* \Pi(x : F)G$. Since $p \vdash t \Rightarrow u : A$ we have $\vdash_p t : A$. Let $q \leqslant p$ and $q \Vdash a : F$. We have $q \vdash t\,a \Rightarrow u\,a : G[a]$. By induction $q \Vdash t\,a : G[a]$ and $q \Vdash t\,a = u\,a : G[a]$. If $q \Vdash a = b : F$ we similarly get $q \Vdash t\,b : G[b]$ and $q \Vdash t\,b = u\,b : G[b]$. Since $q \Vdash G[a] = G[b]$, by Lemma 3.24, $q \Vdash t\,b = u\,b : G[a]$. But $q \Vdash u\,a = u\,b : G[a]$. By symmetry and transitivity $q \Vdash t\,a = t\,b : G[a]$. Thus $p \Vdash t : A$ and $p \Vdash t = u : A$.

$(\mathbf{F}_\Sigma)$ Let $p \vdash A \Rightarrow^* \Sigma(x : F)G$. From $p \vdash t \Rightarrow u : A$ we have $\vdash_p t : A$ and we have $p \vdash t.1 \Rightarrow u.1 : F$ and $p \vdash t.2 \Rightarrow u.2 : G[u.1]$. By induction $p \Vdash t.1 : F$ and $p \Vdash t.1 = u.1 : F$. By induction $p \Vdash t.2 : G[u.1]$ and $p \Vdash t.2 = u.2 : G[u.1]$. But since $p \Vdash A$ and we have shown $p \Vdash t.1 = u.1 : F$ we get $p \Vdash G[t.1] = G[u.1]$. By Lemma 3.24, $p \Vdash t.2 : G[t.1]$ and $p \Vdash t.2 = u.2 : G[t.1]$. Thus $p \Vdash t : A$ and $p \Vdash t = u : A$

$(\mathbf{F}_{\mathbf{Loc}})$ Let $p \vdash A \Rightarrow^* \mathbb{E}[\mathsf{f}\,\overline{k}]$ for $k \notin \mathrm{dom}(p)$. Since $p \Vdash u : A$ we have $p(k,b) \Vdash u : A$ for all $b \in \{0,1\}$. But we have $p(k,b) \vdash t \Rightarrow u : A$. By induction $p(k,b) \Vdash t : A$ and $p(k,b) \Vdash t = u : A$. By the definition $p \Vdash t : A$ and $p \Vdash t = u : A$. ◀

▶ **Corollary 4.3.** *Let $p \vdash t \Rightarrow^* u : A$ and $p \Vdash A$. If $p \Vdash u : A$ then $p \Vdash t : A$ and $p \Vdash t = u : A$.*

▶ **Corollary 4.4.** $\Vdash \mathsf{f} : N \to N_2$.

Proof. It's direct to see that $\Vdash N \to N_2$. For an arbitrary condition p let $p \Vdash n : N$. By Lemma 3.23, we have a parition $p \lhd p_1, \ldots, p_m$ where for each i, $p_i \vdash n \Rightarrow^* \overline{m}_i : N$ for some $m_i \in \mathbb{N}$. We have thus a reduction $p_i \vdash \mathsf{f}\,n \Rightarrow^* \mathsf{f}\,\overline{m}_i : N_2$. If $\overline{m}_i \in \mathrm{dom}(p_i)$ then $p_i \vdash \mathsf{f}\,n \Rightarrow^* \mathsf{f}\,\overline{m}_i \Rightarrow b_i : N_2$ for some $b_i \in \{0,1\}$ and by definition $p_i \Vdash \mathsf{f}\,n : N_2$. If for any j, $\overline{m}_j \notin \mathrm{dom}(p_j)$ then $p_j(m_j, 0) \vdash \mathsf{f}\,n \Rightarrow^* \mathsf{f}\overline{m}_j \Rightarrow 0 : N_2$ and $p_j(m_j, 1) \vdash \mathsf{f}\,n \Rightarrow^* \mathsf{f}\overline{m}_j \Rightarrow 1 : N_2$. Thus $p_j(m_j, 0) \Vdash \mathsf{f}\,n : N_2$ and $p_j(m_j, 1) \Vdash \mathsf{f}\,n : N_2$. By the definition $p_j \Vdash \mathsf{f}\,n : N_2$. We thus have that $p_i \Vdash \mathsf{f}\,n : N_2$ for all i and by local character $p \Vdash \mathsf{f}\,n : N_2$. Similarly we can show $p \Vdash \mathsf{f}\,n_1 = \mathsf{f}\,n_2 : N_2$ whenever $p \Vdash n_1 = n_2 : N$. Hence $\vdash \mathsf{f} : N \to N_2$. ◀

▶ **Lemma 4.5.** *If $\vdash_p t : \neg A$ and $p \Vdash A$ then $p \Vdash t : \neg A$ iff for all $q \leqslant p$ there is no term u such that $q \Vdash u : A$.*

Proof. Let $p \Vdash A$ and $\vdash_p t : \neg A$. We have directly that $p \Vdash \neg A$. Let $p \Vdash t : \neg A$. If $q \Vdash u : A$ for some $q \leqslant p$, then $q \Vdash t\,u : N_0$ which is impossible. Conversely, assume it is the case that for all $q \leqslant p$ there is no u for which $q \Vdash u : A$. Since $r \Vdash a : A$ and $r \Vdash a = b : A$ never hold for any $r \leqslant p$, the statement "$r \Vdash t\,a : N_0$ whenever $r \Vdash a : A$ and $r \Vdash t\,a = t\,b : N_0$ whenever $r \Vdash a = b : A$" holds trivially. ◀

▶ **Lemma 4.6.** $\Vdash \mathsf{w} : \neg\neg(\Sigma(x : N)\mathsf{IsZero}(\mathsf{f}\,x))$.

Proof. By Lemma 4.5 it is enough to show that for all q there is no term u for which $q \Vdash u : \neg(\Sigma(x : N)\mathsf{IsZero}(\mathsf{f}\,x))$. Assume $q \Vdash u : \neg(\Sigma(x : N)\mathsf{IsZero}(\mathsf{f}\,x))$ for some u. Let $m \notin \mathrm{dom}(q)$ we have then $q(m,0) \Vdash (\overline{m}, 0) : \Sigma(x : N)\mathsf{IsZero}(\mathsf{f}\,x)$ thus $q(m,0) \Vdash u\,(\overline{m}, 0) : N_0$ which is impossible. ◀

Let $\Gamma \coloneqq x_1 : A_1 \ldots, x_n : A_n[x_1, \ldots, x_{n-1}]$ and $\rho \coloneqq a_1, \ldots, a_n$. We say $p \Vdash \rho : \Gamma$ if $p \Vdash a_1 : A, \ldots, p \Vdash a_n : A_n[a_1, \ldots, a_{n-1}]$. If moreover $\sigma \coloneqq b_1, \ldots, b_n$ and $p \Vdash \sigma : \Gamma$, we say $p \Vdash \rho = \sigma : \Gamma$ if $p \Vdash a_1 = b_1 : A_1, \ldots, p \Vdash a_n = b_n : A_n[a_1, \ldots, a_{n-1}]$.

▶ **Lemma 4.7.** *Let $\Gamma \vdash_p$. For all $q \leqslant p$, if $q \Vdash \rho : \Gamma$, $q \Vdash \sigma : \Gamma$ and $q \Vdash \rho = \sigma : \Gamma$ then*

- *If $\Gamma \vdash_p A$ then $q \Vdash A\rho = A\sigma$ and if $\Gamma \vdash_p A = B$ then $q \Vdash A\rho = B\rho$.*
- *If $\Gamma \vdash_p a : A$ then $q \Vdash a\rho = a\sigma : A\rho$ and if $\Gamma \vdash_p a = b : A$ then $q \Vdash a\rho = b\rho : A\rho$*

Proof. The proof is by induction on the rules of the type system. We show that if the statement holds for the premise of the rule it holds for the conclusion. For economy of presentation we only present the proof for few selected rules. For the rest of the rules the proof follows in a similar fashion.

- For the elimination rules (β), (UNITREC-0), (BOOLREC-0), (BOOLREC-1), (NATREC-0), (NATREC-SUC), (pr_1), (pr_2) and (f-EVAL) the statement follows from Corollary 4.3.
- For the congruence rules the statement follows from Lemma 3.24, Lemma 3.25.
- The statement follows for (f-I) by Corollary 4.4, for (w-TERM) by Lemma 4.6, and for (LOC) by Lemma 3.22.
- (NAT-SUC) By induction $q \Vdash n\rho = n\sigma : N$. By Lemma 3.23 there is a partition $q \lhd q_1, \dots, q_\ell$ where for each i, $q_i \vdash n\rho \Rightarrow^* \overline{m}_i : N$ and $q_i \vdash n\sigma \Rightarrow^* \overline{m}_i : N$ for some $m_i \in \mathbb{N}$. But then $q_i \vdash \mathsf{S}\, n\rho \Rightarrow^* \mathsf{S}\, \overline{m}_i : N$ and $q_i \vdash \mathsf{S}\, n\sigma \Rightarrow^* \mathsf{S}\, \overline{m}_i : N$ for all i. Thus $q_i \Vdash \mathsf{S}\, n\rho = \mathsf{S}\, n\sigma : N$ for all i and by local character $q \Vdash \mathsf{S}\, n\rho = \mathsf{S}\, n\sigma : N$.
- (Π-I) By induction $q \Vdash F\rho = F\sigma$. Let $r \leqslant q$. We have $r \Vdash (\rho, c) = (\rho, b) : (\Gamma, x : F)$ whenever $r \Vdash c = b : F$ and by induction $r \Vdash G\rho[c] = G\rho[b]$. We have then $q \Vdash \Pi(x : F\rho)G\rho$ and similarly $q \Vdash \Pi(x : F\sigma)G\sigma$. Whenever $r \Vdash a : F\rho$ then, by Lemma 3.24, $r \Vdash (\rho, a) = (\sigma, a) : (\Gamma, x : F)$ and by induction $r \Vdash G\rho[a] = G\sigma[a]$. Thus $q \Vdash \Pi(x : F\rho)G\rho = \Pi(x : F\sigma)G\sigma$.
- (λ-I) From $\Gamma, x : F \vdash_p t : G$ we have $\Gamma \vdash_p F$ and $\Gamma, x : F \vdash_p G$. Similarly to (Π-I) we can show $q \Vdash \Pi(x : F\rho)G\rho$, $q \Vdash \Pi(x : F\sigma)G\sigma$, and $q \Vdash \Pi(x : F\rho)G\rho = \Pi(x : F\sigma)G\sigma$. Let $r \leqslant q$ and $r \Vdash a : F\rho$. We have $r \Vdash (\rho, a) = (\sigma, a) : (\Gamma, x : F)$ and by induction $r \Vdash t\rho[a] = t\sigma[a] : G\rho[a]$. But $r \vdash (\lambda x.t\rho)\, a \Rightarrow t\rho[a] : G\rho[a]$ and $r \vdash (\lambda x.t\sigma)\, a \Rightarrow t\sigma[a] : G\sigma[a]$. By Lemma 4.2 one has $r \Vdash (\lambda x.t\rho)\, a = t\rho[a] : G\rho[a]$ and $r \Vdash (\lambda x.t\sigma)\, a = t\sigma[a] : G\sigma[a]$. Since by induction we have $r \Vdash G\rho[a] = G\sigma[a]$, by Lemma 3.24, $r \Vdash (\lambda x.t\sigma)\, a = t\sigma[a] : G\rho[a]$. By symmetry and transitivity $r \Vdash (\lambda x.t\rho)a = (\lambda x.t\sigma)a : G\rho[a]$. Similarly we can show $r \Vdash (\lambda x.t\rho)\, a = (\lambda x.t\rho)\, b : \Pi(x : F\rho)G\rho$ whenever $r \Vdash a = b : F\rho$ and $r \Vdash (\lambda x.t\sigma)\, a = (\lambda x.t\sigma)\, b : \Pi(x : F\sigma)G\sigma$ whenever $r \Vdash a = b : F\sigma$. Thus $q \Vdash (\lambda x.t\rho) = (\lambda x.t\sigma) : \Pi(x : F\rho)G\rho$
- ($\perp$REC-I-E) Follows trivially since $r \Vdash t : N_0$ never holds for any condition r.
- (NATREC-I) While we omit the proof here the basic idea is as follows: If for some $r \leqslant q$ we have $r \Vdash t : N$ then by Lemma 3.23, we have $r \lhd r_1, \dots, r_n$ and for each i, $r_i \vdash t \Rightarrow^* \mathsf{S}^{k_i} 0$ for some $k_i \in \mathbb{N}$. By induction on k_i we can show $r_i \Vdash (\mathsf{natrec}\ (\lambda x.F)\, a_0\, g)\rho\, t : F\rho[t]$ for all i. By local character we will then have $r \Vdash (\mathsf{natrec}\ (\lambda x.F)\, a_0\, g)\rho\, t : F\rho[t]$. Similarly we can show $r \Vdash (\mathsf{natrec}\ (\lambda x.F)\, a_0\, g)\rho\, t = (\mathsf{natrec}\ (\lambda x.F)\, a_0\, g)\rho\, u : F\rho[t]$ whenever $r \Vdash t = u : N$. By the definition we will have $q \Vdash (\mathsf{natrec}\ (\lambda x.F)\, a_0\, g)\rho : \Pi(x : N)F\rho$ and similarly we can show $q \Vdash (\mathsf{natrec}\ (\lambda x.F)\, a_0\, g)\rho = (\mathsf{natrec}\ (\lambda x.F)\, a_0\, g)\sigma : \Pi(x : N)F\rho$. ◀

▶ **Theorem 4.8** (Fundamental Theorem). *If $\vdash_p J$ then $p \Vdash J$.*

5 Markov's principle

Now we have enough machinery to show the independence of MP from type theory. The idea is that if a judgment J is derivable in type theory (i.e. $\vdash J$) then it is derivable in the forcing extension (i.e. $\vdash_{\langle\rangle} J$) and by Theorem 4.8 it holds in the interpretation (i.e. $\Vdash J$). It thus suffices to show that there no t such that $\Vdash t : \mathrm{MP}$ to establish the independence of MP from type theory. First we recall the formulation of MP.

$$\mathrm{MP} \coloneqq \Pi(h : N \to N_2)[\neg\neg(\Sigma(x : N)\, \mathsf{IsZero}\,(h\, x)) \to \Sigma(x : N)\, \mathsf{IsZero}\,(h\, x)]$$

where $\mathsf{IsZero} : N_2 \to U$ is given by $\mathsf{IsZero} \coloneqq \lambda y.\mathsf{boolrec}\ (\lambda x.U)\ N_1\ N_0\ y$.

▶ **Lemma 5.1.** *There is no term t such that* $\Vdash t : \Sigma(x : N)\, \mathsf{IsZero}\,(\mathsf{f}\, x)$.

Proof. Assume $\Vdash t : \Sigma(x : N)\, \mathsf{IsZero}\,(\mathsf{f}\, x)$ for some t. We then have $\Vdash t.1 : N$ and $\Vdash t.2 : \mathsf{IsZero}\,(\mathsf{f}\, t.1)$. By Lemma 3.23, one has a partition $\langle\rangle \lhd p_1, \dots, p_n$ where for each i, $p_i \vdash t.1 \Rightarrow^* \overline{m}_i$ for some $\overline{m}_i \in \mathbb{N}$. Hence $p_i \vdash \mathsf{IsZero}\,(\mathsf{f}\, t.1) \Rightarrow^* \mathsf{IsZero}\,(\mathsf{f}\, \overline{m}_i)$ and by Lemma 4.1, $p_i \Vdash \mathsf{IsZero}\,(\mathsf{f}\, t.1) = \mathsf{IsZero}\,(\mathsf{f}\, \overline{m}_i)$. But, by definition, a partition of $\langle\rangle$ must contain a condition, say p_j, such that $p_j(k) = 1$ whenever $k \in \mathrm{dom}(p_j)$ (this holds vacuously for $\langle\rangle \lhd \langle\rangle$). Assume $m_j \in \mathrm{dom}(p_j)$, then $p_j \vdash \mathsf{IsZero}\,(\mathsf{f}\, t.1) \Rightarrow^* \mathsf{IsZero}\,(\mathsf{f}\, m_j) \Rightarrow^* N_0$. By monotonicity, from $\Vdash t.2 : \mathsf{IsZero}\,(\mathsf{f}\, t.1)$ we get $p_j \Vdash t.2 : \mathsf{IsZero}\,(\mathsf{f}\, t.1)$. But $p_j \vdash \mathsf{IsZero}\,(\mathsf{f}\, t.1) \Rightarrow^* N_0$ thus $p_j \Vdash \mathsf{IsZero}\,(\mathsf{f}\, t.1) = N_0$. Hence, by Lemma 3.24, $p_j \Vdash t.2 : N_0$ which is impossible, thus contradicting our assumption. If on the other hand $m_j \notin \mathrm{dom}(p_j)$ then since $p_j \lhd p_j(m_j, 0), p_j(m_j, 1)$ we can apply the above argument with $p_j(m_j, 1)$ instead of p_j. ◀

▶ **Lemma 5.2.** *There is no term t such that* $\Vdash t : \mathrm{MP}$.

Proof. Assume $\Vdash t : \mathrm{MP}$ for some t. From the definition, whenever $\Vdash g : N \to N_2$ we have $\Vdash t\, g : \neg\neg(\Sigma(x : N)\, \mathsf{IsZero}\,(g\, x)) \to \Sigma(x : N)\, \mathsf{IsZero}\,(g\, x)$. Since by Corollary 4.4, $\Vdash \mathsf{f} : N \to N_2$ we have $\Vdash t\, \mathsf{f} : \neg\neg(\Sigma(x : N)\, \mathsf{IsZero}\,(\mathsf{f}\, x)) \to \Sigma(x : N)\, \mathsf{IsZero}\,(\mathsf{f}\, x)$. Since by Lemma 4.6, $\Vdash \mathsf{w} : \neg\neg(\Sigma(x : N)\, \mathsf{IsZero}\,(\mathsf{f}\, x))$ we have, $\Vdash (t\, \mathsf{f})\, \mathsf{w} : \Sigma(x : N)\, \mathsf{IsZero}\,(\mathsf{f}\, x)$ which is impossible by Lemma 5.1. ◀

From Theorem 4.8, Lemma 5.2, and Lemma 2.3 we can then conclude:

▶ **Theorem 2.1.** *There is no term t such that* $\mathrm{MLTT} \vdash t : \mathrm{MP}$.

5.1 Many Cohen reals

We extend the type system in Section 2 further by adding a generic point f_q for each condition q. The introduction and conversion rules for f_q are given by:

$$\frac{\Gamma \vdash_p}{\Gamma \vdash_p \mathsf{f}_q : N \to N_2} \quad \frac{\Gamma \vdash_p}{\Gamma \vdash_p \mathsf{f}_q\, \overline{n} = 1}\, n \in \mathrm{dom}(q) \quad \frac{\Gamma \vdash_p}{\Gamma \vdash_p \mathsf{f}_q\, \overline{n} = p(n)}\, n \notin \mathrm{dom}(q), n \in \mathrm{dom}(p)\,.$$

With the reduction rules: $\dfrac{n \in \mathrm{dom}(q)}{\mathsf{f}_q\, \overline{n} \to 1} \quad \dfrac{n \notin \mathrm{dom}(q), n \in \mathrm{dom}(p)}{\mathsf{f}_q\, \overline{n} \to_p p(n)}$.

We observe that the reduction relation is still monotone.

For each f_q we add a term $\dfrac{\Gamma \vdash_p}{\Gamma \vdash_p \mathsf{w}_q : \neg\neg(\Sigma(x : N)\, \mathsf{IsZero}\,(\mathsf{f}_q\, x))}$.

Finally we add a term mw witnessing the negation of MP $\dfrac{\Gamma \vdash_p}{\Gamma \vdash_p \mathsf{mw} : \neg\mathrm{MP}}$.

By analogy to Corollary 4.4 we have:

▶ **Lemma 5.3.** $\Vdash \mathsf{f}_q : N \to N_2$ *for all q.*

▶ **Lemma 5.4.** $\Vdash \mathsf{w}_q : \neg\neg(\Sigma(x : N)\, \mathsf{IsZero}\,(\mathsf{f}_q\, x))$ *for all q.*

Proof. Assume $p \Vdash t : \neg(\Sigma(x : N) \mathsf{IsZero}\,(\mathsf{f}_q\, x))$ for some p and t. Let $m \notin \mathrm{dom}(q) \cup \mathrm{dom}(p)$, we have $p(\overline{m}, 0) \Vdash \mathsf{f}_q\, m = 0$. Thus $p(\overline{m}, 0) \Vdash (\overline{m}, 0) : \Sigma(x : N)\, \mathsf{IsZero}\,(\mathsf{f}_q\, x)$ and $p(\overline{m}, 0) \Vdash t\, (\overline{m}, 0) : N_0$ which is impossible. ◀

▶ **Lemma 5.5.** *There is no term t for which* $q \Vdash t : \Sigma(x : N)\, \mathsf{IsZero}\,(\mathsf{f}_q\, x)$.

Proof. Assume $q \Vdash t : \Sigma(x : N)\,\mathsf{IsZero}\,(\mathsf{f}_q\, x)$ for some t. We then have $q \Vdash t.1 : N$ and $q \Vdash t.2 : \mathsf{IsZero}\,(\mathsf{f}_q\, t.1)$. By Lemma 3.23 one has a partition $q \lhd q_1, \dots, q_n$ where for each i, $t.1 \Rightarrow^*_{q_i} \overline{m}_i$ for some $\overline{m}_i \in \mathbb{N}$. Hence $q_i \vdash \mathsf{IsZero}\,(\mathsf{f}_q\, t.1) \Rightarrow^* \mathsf{IsZero}\,(\mathsf{f}_q\, \overline{m}_i)$. But any partition of q contain a condition, say q_j, where $q_j(k) = 1$ whenever $k \notin \mathrm{dom}(q)$ and $k \in \mathrm{dom}(q_j)$. Assume $m_j \in \mathrm{dom}(q_j)$. If $m_j \in \mathrm{dom}(q)$ then $q_j \vdash \mathsf{f}_q\, m_j \Rightarrow 1 : N_2$ and if $m_j \notin \mathrm{dom}(q)$ then $q_j \vdash \mathsf{f}_q\, \overline{m}_j \Rightarrow q_j(k) \coloneqq 1 : N_2$. Thus $q_j \vdash \mathsf{IsZero}\,(\mathsf{f}_q\, t.1) \Rightarrow^* N_0$ and by Lemma 4.1, $q_j \Vdash \mathsf{IsZero}\,(\mathsf{f}\, t.1) = N_0$. From $\Vdash t.2 : \mathsf{IsZero}\,(\mathsf{f}\, t.1)$ by monotonicity and Lemma 3.24 we have $q_j \Vdash t.2 : N_0$ which is impossible. If on the other hand $m_j \notin \mathrm{dom}(q_j)$ then since $q_j \lhd q_j(m_j, 0), q_j(m_j, 1)$ we can apply the above argument with $q_j(m_j, 1)$ instead of q_j. ◀

▶ **Lemma 5.6.** $\Vdash \mathsf{mw} : \neg\mathrm{MP}$

Proof. Assume $p \Vdash t : \mathrm{MP}$ for some p and t. Thus whenever $q \leqslant p$ and $q \Vdash u : N \to N_2$ then $q \Vdash t\,u : \neg\neg(\Sigma(x : N)\,\mathsf{IsZero}\,(u\,x)) \to (\Sigma(x : N)\,\mathsf{IsZero}\,(u\,x))$. But we have $q \Vdash \mathsf{f}_q : N \to N_2$ by Lemma 5.3. Hence $q \Vdash t\,\mathsf{f}_q : \neg\neg(\Sigma(x : N)\mathsf{IsZero}\,(\mathsf{f}_q\, x)) \to (\Sigma(x : N)\mathsf{IsZero}\,(\mathsf{f}_q\, x))$. But $q \Vdash \mathsf{w}_q : \neg\neg(\Sigma(x : N)\mathsf{IsZero}\,(\mathsf{f}_q\, x))$ by Lemma 5.4. Thus $q \Vdash (t\,\mathsf{f}_q)\,\mathsf{w}_q : \Sigma(x : N)\,\mathsf{IsZero}\,(\mathsf{f}_q\, x)$ which is impossible by Lemma 5.5. ◀

We have then the following result.

▶ **Theorem 5.7.** *There is a consistent extension of* MLTT *where* ¬MP *is derivable.*

Acknowledgements. We thank Simon Huber, Thomas Streicher, Martin Escardo and Chuangjie Xu. We also thank the reviewers for providing valuable comments which helped improve the manuscript.

References

1 Andreas Abel and Gabriel Scherer. On irrelevance and algorithmic equality in predicative type theory. *Logical Methods in Computer Science*, 8(1):1–36, 2012. doi:10.2168/LMCS-8(1:29)2012.

2 Peter Aczel. On relating type theories and set theories. In Thorsten Altenkirch, Bernhard Reus, and Wolfgang Naraschewski, editors, *Types for Proofs and Programs: International Workshop, TYPES' 98 Kloster Irsee, Germany, March 27–31, 1998 Selected Papers*, pages 1–18, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

3 Errett Bishop. *Foundations of Constructive Analysis.* McGraw-Hill, 1967.

4 L. E. J. Brouwer. Essentially negative properties. In A. Heyting, editor, *Collected Works*, volume I, pages 478 – 479. North-Holland, 1975.

5 Paul J Cohen. The independence of the continuum hypothesis. *Proceedings of the National Academy of Sciences of the United States of America*, 50(6):1143–1148, 12 1963.

6 Robert L. Constable and Scott Fraser Smith. Partial objects in constructive type theory. In *Proceedings of Second IEEE Symposium on Logic in Computer Science*, pages 183–193, 1987.

7 Thierry Coquand and Guilhem Jaber. A note on forcing and type theory. *Fundamenta Informaticae*, 100(1-4):43–52, 2010.

8 Peter Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *The Journal of Symbolic Logic*, 65(2):525–549, 2000.

9 Martin Hofmann and Thomas Streicher. Lifting grothendieck universes. unpublished note, publication year unknown. URL: http://www.mathematik.tu-darmstadt.de/~streicher/NOTES/lift.pdf.

10 J.M.E. Hyland and C.-H.L. Ong. Modified realizability toposes and strong normalization proofs (extended abstract). In *Typed Lambda Calculi and Applications, LNCS 664*, pages 179–194. Springer-Verlag, 1993.

11 Alexei Kopylov and Aleksey Nogin. Markov's principle for propositional type theory. In Laurent Fribourg, editor, *Computer Science Logic: 15th International Workshop, CSL 2001 10th Annual Conference of the EACSL Paris, France, September 10–13, 2001, Proceedings*, pages 570–584, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

12 Georg Kreisel. Interpretation of analysis by means of constructive functionals of finite types. In A. Heyting, editor, *Constructivity in Mathematics*, pages 101–128. Amsterdam, North-Holland Pub. Co., 1959.

13 Kenneth Kunen. *Set Theory: An Introduction to Independence Proofs*, volume 102 of *Studies in Logic and the Foundations of Mathematics.* Elsevier, 1980.

14 Maurice Margenstern. L'école constructive de Markov. *Revue d'histoire des mathématiques*, 1(2):271–305, 1995.

15 Per Martin-Löf. An intuitionistic theory of types. reprinted in Twenty-five years of constructive type theory, Oxford University Press, 1998, 127–172, 1972.

16 Ulf Norell. *Towards a practical programming language based on dependent type theory.* PhD thesis, Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, September 2007.

17 Thomas Streicher. Universes in toposes. In Laura Crosilla and Peter Schuster, editors, *From Sets and Types to Topology and Analysis: Towards practicable foundations for constructive mathematics*, pages 78–90. Oxford University Press, 2005.

18 William W. Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967.

19 Dirk van Dalen. An interpretation of intuitionistic analysis. *Annals of Mathematical Logic*, 13(1):1 – 43, 1978.

20 Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. *Information and Computation*, 115(1):38 – 94, 1994.

21 Chuangjie Xu and Martín Escardó. Universes in sheaf models. University of Birmingham, 2016.

On Undefined and Meaningless in Lambda Definability*

Fer-Jan de Vries

Computer Science, University of Leicester, Leicester, UK
fdv1@le.ac.uk

Abstract

We distinguish between undefined terms as used in lambda definability of partial recursive functions and meaningless terms as used in infinite lambda calculus for the infinitary terms models that generalise the Böhm model. While there are uncountable many known sets of meaningless terms, there are four known sets of undefined terms. Two of these four are sets of meaningless terms.

In this paper we first present set of sufficient conditions for a set of lambda terms to serve as set of undefined terms in lambda definability of partial functions. The four known sets of undefined terms satisfy these conditions.

Next we locate the smallest set of meaningless terms satisfying these conditions. This set sits very low in the lattice of all sets of meaningless terms. Any larger set of meaningless terms than this smallest set is a set of undefined terms. Thus we find uncountably many new sets of undefined terms.

As an unexpected bonus of our careful analysis of lambda definability we obtain a natural modification, strict lambda-definability, which allows for a Barendregt style of proof in which the representation of composition is truly the composition of representations.

1998 ACM Subject Classification F.4.1 [Mathematical Logic] lambda calculus and related systems

Keywords and phrases lambda calculus, lambda definability, partial recursive function, undefined term, meaningless term

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.18

1 Introduction

The intuition that not all lambda terms are equally *significant* from a computational point of view is as old as lambda calculus itself. It is of particular interest that in lambda calculus, unlike e.g. the notion of zero in arithmetic, the notion of insignificant term is not uniquely determined. There are many different reasonable choices that one can make for a set of unsignificant terms. Making a concrete choice is akin to choosing a semantics for the lambda calculus.

The oldest, relatively understudied application of insignificant terms is made in the lambda definability of partial recursive functions. In this area insignificant terms are traditionally called *undefined terms*. The other more modern and better understood application is the construction of infinitary term models of the lambda calculus. This construction generalises the Böhm model. In the latter case the insignificant terms are called *meaningless terms*.

[*] Dedicated in friendship to Albert Visser on the occasion of his retirement.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 18; pp. 18:1–18:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

There are four well-known sets of undefined terms used in lambda definability. In contrast there are uncountably many sets of meaningless terms, that each give rise to their own model of lambda calculus. As it happens only two of the four known sets of undefined terms are also sets of meaningless terms. Hence it is a natural question to ask which of the uncountable many other sets of meaningless terms can also play the role of set of undefined terms, so that in the corresponding model the recursive functions can naturally be interpreted.

The proof technique of Statman's theorem which, as Barendregt has shown, works uniformly for each of the four known sets of undefined terms does not generalise to arbitrary sets of meaningless terms, because in general sets of meaningless terms are not co-Visser-sets. Instead we analyse the proof of lambda definability in Barendregt's PhD thesis. With our modern insight in sets of meaningless terms we can generalise and improve this old proof, but we can also improve it.

Church and Kleene were the first to give lambda-representation of recursive functions. In their representation the role of undefined terms is played by the terms without a finite normal form. Barendregt criticises their representation and expresses the clear ideal that the lambda-representation of recursive functions should preserve the way they are defined (this wish is also known as Kreisel's Superthesis). The Church-Kleene lambda-representation falls short of this ideal: the representation of the composition of two recursive functions is not the composition of their represenation. Barendregt then takes the rather revolutionary step to replace their old notion of undefined term by the new concept of unsolvable term.[1]

This new notion of undefined term indeed allows for an improvement of the old proof. But there is a surprise. Barendregts lambda-represenation of partial recursive functions falls arguably short of his own ideal. He gives first a lambda-representation of the total recursive functions and then a lambda representation of the partial recursive functions. In case of the total functions he use the natural notion of composition of their representations. In case of the partial recursive functions he defines composition in a slightly ad hoc way. He gets around this definition by using a very clever "jamming" trick.

We observe in this paper that by using the novel concept of strict lambda definability we can represent the partial recursive functions in such a way that their definition is completely preserved, in line with Barendregt's original "dream improvement" of the old proof by Church. This reformulated proof is also more general: it now applies to any set of meaningless terms satisfying one particular extra closure condition.

Ordered by inclusion the sets of meaningless terms form a lattice. The largest set of meaningless terms that can be used to make a model of lambda calculus is the set of unsolvables. The smallest such set is the set of rootactive terms[2]. The infinitary term models constructed with these two sets of meaningless terms are the Böhm model and the Berarducci model. In the Böhm model they are exactly the unsolvables that are equated with $\perp$. In the Berarducci model they are the rootactives that are equated with $\perp$.

The smallest set of undefined terms that we can identify sits very low in this lattice of sets of meaningless terms just above the the set of rootactives. This raises an open question: whether the partial recursive functions can be interpreted in the Berarducci model of the

[1] A closed lambda term M is *solvable* if $MN_1 \dots N_k = \mathbf{I}$ for some sequence $N_1, \dots, N_k$ with $k \geq 0$. An open lambda term is called *solvable* if its closure is solvable. A lambda terms is called *unsolvable* if it is not solvable.

[2] A lambda term M is rootactive if any reduct of M can further reduce to a redex. The classical rootactive term is $\mathbf{\Omega}$. The unsolvable $\Omega\mathbf{I}$ is not rootactive. Note that the definition of a rootactive terms allows for free variables. The term EyE with $E \equiv \mathbf{\Theta}\lambda xyz.zyx$ is a concrete example of a rootactive term with free variable y.

lambda calculus in such a way that if a partial recursive function f is not defined on a natural number n, then the interpretation of $f(n)$ equals bottom.

2 Brief recap of 80 years of lambda definability

Lambda definability goes some 80 years back to the exciting, early days of lambda calculus when, in the slipstream of Gödel's incompleteness theorems, Church and his students Kleene and Rosser were experimenting which functions could be represented in the lambda calculus, Gödel and Herbrand defined the recursive functions and Turing tried to capture the intuitive idea of "effective calculable" function with his machines. While Church was mainly using the $\lambda\mathbf{I}$-calculus[3] considers only in his papers, Turing realised that it is "naturally much simpler" [22] to use $\lambda\mathbf{K}$-calculus, what we now call the lambda calculus.

That recursive functions on natural numbers can be represented in lambda calculus is due to Kleene [15]. The converse, that lambda-definable functions on natural numbers are recursive, is due to Kleene and Church independently [15, 9]. These results are important as on one hand they led Church to his Church Thesis and on the other hand they demonstrate that lambda calculus is a paradigmatic programming language [3].

The first definition of lambda definability dealt with total functions, as partial recursive functions had not yet been defined.

▶ **Definition 1.** A total function $f : \mathbb{N} \to \mathbb{N}$ is λ-*definable* if for some lambda term F and each $n \in \mathbb{N}$ we have $F\ulcorner n\urcorner = \ulcorner f(n)\urcorner$.

▶ **Theorem 2.** *A total function $f : \mathbb{N} \to \mathbb{N}$ is λ-definable if and only if f is recursive.*

When Kleene [16] defined next the partial recursive functions, Theorem 2 was immediately extended to partial recursive functions [10, 11]. This required an extra clause to Definition 1 to explain what happens when the function that one wants to represent happens to be undefined on some input.

▶ **Definition 3.** Let $\mathcal{U}$ be a set of lambda terms. A partial function $\phi : \mathbb{N} \rightharpoonup \mathbb{N}$ is $\lambda_{\mathcal{U}}$-*definable* if for some lambda term F and each $n \in \mathbb{N}$:

$$\begin{array}{ll} F\ulcorner n\urcorner = \ulcorner \phi(n)\urcorner & \text{if } \phi(n)\downarrow \\ F\ulcorner n\urcorner \in \mathcal{U} & \text{else.} \end{array}$$

2.1 Kleene-Church: undefined is having no normal form

Church [10, p. 29] used the set $\overline{\mathcal{NF}}$ lambda terms without normal form for $\mathcal{U}$ to represent "undefined".

▶ **Theorem 4** (Kleene). *A partial function $\phi : \mathbb{N} \rightharpoonup \mathbb{N}$ is $\lambda_{\overline{\mathcal{NF}}}$-definable if and only if ϕ is partial recursive.*

In his thesis [1, 3] Barendregt points out that there is a practical problem with composition in the approach of Church and Kleene. If f, g are $\lambda_{\overline{\mathcal{NF}}}$-defined by F, G, then $f \circ g$ can not be $\lambda_{\overline{\mathcal{NF}}}$-defined by $\lambda x.F(Gx)$, which would be the natural way to $\lambda_{\overline{\mathcal{NF}}}$-define $F \circ G$. As example, Barendregt takes for f the constant zero function represented by $\lambda x.\ulcorner 0\urcorner$ and for g the function that is everywhere undefined represented by $\mathbf{\Omega}$. Then $f \circ g$ is totally undefined,

[3] The $\lambda\mathbf{I}$-calculus allows terms of the form $\lambda x.M$ only if x is a free variable of M.

but $F \circ G \equiv \lambda x.F(Gx) = \lambda x.\ulcorner 0 \urcorner$. The conclusion is then that "it is not immediate that the $\lambda_{\overline{\mathcal{NF}}}$-definable functions are closed under composition." Kleene and Church avoid this issue by using Kleene's normal form theorem. They represent only the normal form of a partial recursive function. Barendregt emphasises that their representation of the partial recursive functions is not intensional, as it does not preserve their definition trees.

2.2 Barendregt: undefined is being unsolvable

Barendregt's solution is to take for $\mathcal{U}$ the set $\overline{\mathcal{HNF}}$[4] of unsolvables.

▶ **Theorem 5** (Barendregt). *A partial function $\phi : \mathbb{N} \rightharpoonup \mathbb{N}$ is $\lambda_{\overline{\mathcal{HNF}}}$-definable if and only if ϕ is partial recursive.*

Barendregt uses Lercher's jamming factor trick and represents the composition $f \circ g$ by $\lambda x.Gx\mathbf{KII}F(Gx)$. This clever trick works because if $g(n)$ is undefined then $G\ulcorner n \urcorner$ is unsolvable and hence also $F\ulcorner n \urcorner \equiv G\ulcorner n \urcorner\mathbf{KII}F(G\ulcorner n \urcorner)$ is unsolvable, and if $g(n)$ is defined then it can be shown that $G\ulcorner n \urcorner\mathbf{KII} \twoheadrightarrow_\beta \mathbf{I}$ in which case $F\ulcorner n \urcorner = F(G\ulcorner n \urcorner)$. Cf. [3, Lemma 8.4.5].

Barendregt felt strongly about this change from $\overline{\mathcal{NF}}$ to $\overline{\mathcal{HNF}}$ in the definition of lambda definability. In his thesis he writes on page xvi: "This is not to be regarded as a mere technical improvement but simply central to the objects which are here intended." And in [4] he explains in detail:

> It has been stressed by Kreisel [18, p. 177-178] that in connection with the so-called "superthesis", Church's thesis expresses less than we know. When we say that all mechanically computable number theoretic functions are λ-definable or recursive, we merely speak of the results of computations, of their graphs. But we have in mind that λ-terms correspond to our procedures for defining these functions. As far as the μ-recursive functions and the λ-definable functions are concerned, strong definability proves the equivalence not only in the sense of Church but also of the super thesis: definitions are preserved. [4]

Given these strong arguments against the traditional Kleene-Church proof of $\lambda_{\overline{\mathcal{NF}}}$-definability, it is a bit unexpected that composition is not defined in the natural way as $\lambda x.F(Gx)$ in $\lambda_{\overline{\mathcal{HNF}}}$-definability. Barendregt's solution to deal with composition is arguably almost but not quite in the spirit of the superthesis.

In Section 3 we will show that Lercher's jamming factor trick is not needed for partial functions either, and that composition can indeed be represented compositionally.

2.3 Statman: undefined is belonging to a co-Visser set

Statman takes a general approach: any non-empty co-Visser set of closed lambda terms can be used as set $\mathcal{U}$ of undefined terms in $\lambda_{\mathcal{U}}$-definability of the partial recursive functions.

▶ **Definition 6.** A set $\mathcal{U} \subseteq \Lambda^0$ is a co-Visser set, if:

1. $\Lambda^0 \backslash \mathcal{U}$ is recursive enumerable,
2. $\Lambda^0 \backslash \mathcal{U}$ is closed under finite β-reduction.

▶ **Theorem 7** (Statman, 1990). *Let $\mathcal{A}$ be a non-empty co-Visser set. Then any partial recursive function is $\lambda_{\mathcal{A}}$-definable.*

[4] Wadsworth has shown that a term is unsolvable iff it has no head normal form. Hence our notation $\overline{\mathcal{HNF}}$ for the set of unsolvables.

Barendregt [5] has given a detailed proof of Statman's Theorem and Visser's Anti Diagonalisation Theorem [24, Thm. 4.4] on which Statman's Theorem is based. This way of proving lambda definability has two consequences. First, Statman's theorem is formulated for closed terms. This is because the proof makes use of a self-interpreter, i.e. a lambda term $\mathbf{E}$ such that for $M \in \Lambda^0$ one has

$$\mathbf{E}\ulcorner \# M \urcorner \twoheadrightarrow_\beta M,$$

where $\# : \Lambda \to \mathsf{Nat}$ is some effective bijection that assigns to a lambda term a unique natural number. This equation cannot hold when M contains free variables [3, Definition 8.5.1]. Secondly, the Statman proof does not give support for Kreisel's superthesis. Visser's theorem uses Ershov's precomplete numerations, so that the proof of Visser's theorem is '"coordinate free" i.e. the proof uses (nearly) no specific properties of lambda calculus' in the words of [24].

Barendregt lists four sets that satisfy the condition of Statman's theorem: (the subsets of closed terms of) $\overline{\mathcal{NF}}$, $\overline{\mathcal{HNF}}$, the set $\overline{\mathcal{WHNF}}$ of terms without a weak head normal form[5] and the set $\mathcal{E}$ of easy[6] terms. Two of these, $\overline{\mathcal{HNF}}, \overline{\mathcal{WHNF}}$ are sets of meaningless terms. We will see in Section 6 that there are many other sets of meaningless terms which don't satisfy the Statman condition and yet can be used as set of undefined terms.

3 Strict $\lambda_{\mathcal{U}}$-definability

In this section we search for general sufficient conditions for a set $\mathcal{U}$ of lambda terms so that we can generalise Barendregt's proof of lambda definability. We follow the notation of [3] for the standard Church coding:

$$\begin{array}{lcllcl}
\mathbf{F} & \equiv & \lambda xy.y & \mathbf{T} & \equiv & \lambda xy.x \\
\mathbf{and} & \equiv & \lambda xy.xyx & \textbf{if } B \textbf{ then } M \textbf{ else } N & \equiv & BMN \\
[M,N] & \equiv & \lambda z.zMN & \mathbf{Zero} & \equiv & \lambda x.x\mathbf{T} \\
\mathbf{S}^+ & \equiv & \lambda x.[\mathbf{T},x] & \mathbf{P}^- & \equiv & \lambda x.[\mathbf{F},x] \\
\mathbf{K} & \equiv & \lambda xy.x & \mathbf{I} & \equiv & \lambda x.x
\end{array}$$

▶ **Definition 8** (Turing's fixed point combinator [23]). We define:

$$\mathbf{\Theta} \equiv (\lambda xy(y(xxy))\lambda xy(y(xxy)).$$

▶ **Definition 9** (Barendregt numerals [2]). We define $\ulcorner\ \urcorner : \mathbb{N} \to \Lambda$ by induction:

$$\begin{array}{lcl}
\ulcorner 0 \urcorner & \equiv & \mathbf{I} \\
\ulcorner n+1 \urcorner & \equiv & [\mathbf{F}, \ulcorner n \urcorner]
\end{array}$$

Let us first follow [3] and define the class of partial recursive functions as the least class of partial numeric functions which contains the total recursive functions and is closed under composition and minimalisation.

Suppose our candidate set of undefined terms is $\mathcal{U}$. We will inspect Barendregt's proof to see what requirements we have to make on $\mathcal{U}$.

[5] A lambda term *has a weak head normal form* if it can reduce to either an abstraction or a term of the form $xM_1 \ldots M_n$.

[6] A lambda term is *easy* if it can consistently be equated to any other lambda term.

3.1 Composition

Let F, G be the representations of the partial numeric functions $f, g : \mathbb{N} \to \mathbb{N}$. The natural way to represent composition is

$$(F \circ G) \equiv (\lambda x.F(Gx)),$$

as in the lambda definability proof of the total recursive functions in [3]. This works all right in case g is well-defined on n and f is defined on $g(n)$, because then

$$\ulcorner f \circ g \urcorner \ulcorner n \urcorner \equiv (F \circ G)\ulcorner n \urcorner \twoheadrightarrow_\beta F(G\ulcorner n \urcorner) \equiv F\ulcorner g(n) \urcorner \equiv \ulcorner f(g(n)) \urcorner.$$

If $g(n)$ is undefined, then $G\ulcorner n \urcorner$ should reduce to some $U \in \mathcal{U}$. With the notion of $\lambda_{\mathcal{U}}$ definability we cannot infer from $F(G\ulcorner n \urcorner) \twoheadrightarrow_\beta FU$ that $FU \in \mathcal{U}$. Barendregt's jamming trick argument can be repeated, provided that the set $\mathcal{U}$ of undefined terms has the property: if $U \in \mathcal{U}$ then $UM \in \mathcal{U}$ for any $M \in \Lambda$. In particular the set of unsolvables has this property.

However, the jamming trick and the previous condition on $\mathcal{U}$ is not needed with the following "stricter" definition of lambda definability:

▶ **Definition 10.** Let $\mathcal{U}$ be a set of lambda terms. A partial function $\phi : \mathbb{N}^p \rightharpoonup \mathbb{N}$ is *strictly $\lambda_{\mathcal{U}}$-definable* if for some lambda term F and each $\vec{n} \in \mathbb{N}^p$

1. $F\ulcorner \vec{n} \urcorner \twoheadrightarrow \ulcorner m \urcorner$ if $\phi(\vec{n}) = m$,
2. $F\ulcorner \vec{n} \urcorner \in \mathcal{U}$ if $\phi(\vec{n}) \uparrow$,
3. $F\vec{N} \in \mathcal{U}$ for all $\vec{N} \in \Lambda^p$ with at least one $N_i \in \mathcal{U}$.

This new strictness clause does the trick for proving that the representation of composition is the composition of the representations:

▶ **Lemma 11.** *The strictly $\lambda_{\mathcal{U}}$-definable partial functions are closed under composition.*

Proof. To keep the notation simple we consider without loss of generality unary numeric functions. Let F, G be the strict representations of the partial numeric functions f, g. Then for $U \in \mathcal{U}$ we have $GU \in \mathcal{U}$ and hence also

$$(F \circ G)U \equiv (\lambda x.F(Gx))U \to F(GU) \in \mathcal{U}.$$

For $n \in \mathbb{N}$ we have either $g(n)\downarrow$ or $g(n)\uparrow$. If the former than

$$(F \circ G)\ulcorner n \urcorner \equiv (\lambda x.F(Gx))\ulcorner n \urcorner \to F(G\ulcorner n \urcorner) \twoheadrightarrow F\ulcorner g(n) \urcorner \twoheadrightarrow \ulcorner f(g(n)) \urcorner.$$

If the latter, then we have that $G\ulcorner n \urcorner \twoheadrightarrow_\beta U$ for some $U \in \mathcal{U}$ and therefore by the new strictness clause we get

$$(F \circ G)\ulcorner n \urcorner \equiv (\lambda x.F(Gx))\ulcorner n \urcorner \to F(G\ulcorner n \urcorner) \twoheadrightarrow FU \in \mathcal{U}.$$ ◀

3.2 Minimalisation

In [3] a lambda term P is called a predicate if $P\ulcorner n \urcorner$ reduces to either $\mathbf{T}$ or $\mathbf{F}$ for all $n \in \mathbb{N}$. We will use in this section *strict* predicates that satisfy the extra property that $PU \in \mathcal{U}$ whenever $U \in \mathcal{U}$ for some fixed set $\mathcal{U}$ of undefined terms.

In the proof of [3, Prop. 8.4.10] we find this definition of a lambda term:

$$H_P \equiv \mathbf{\Theta}(\lambda hx.\mathbf{if}\ Px\ \mathbf{then}\ x\ \mathbf{else}\ h(\mathbf{S}^+x))$$

where P is a predicate, together with the following reduction

$$H_P\ulcorner n\urcorner \twoheadrightarrow_\beta \textbf{if } P\ulcorner n\urcorner \textbf{ then } \ulcorner n\urcorner \textbf{ else } H_P\ulcorner n+1\urcorner.$$

Hence, this finite term H_P can reduce with an infinite reduction to the infinite expression

$$\textbf{if } P\ulcorner 0\urcorner \textbf{ then } \ulcorner 0\urcorner \textbf{ else if } P\ulcorner 1\urcorner \textbf{ then } \ulcorner 1\urcorner \textbf{ else if } P\ulcorner 2\urcorner \textbf{ then } \ulcorner 2\urcorner \textbf{ else } \ldots$$

Recall that minimalisation is the construction of a new partial function

$$\mu m[\chi(\vec{n}, m) = 0] : \mathbb{N}^p \to \mathbb{N}$$

from a given partial function $\chi : \mathbb{N}^{p+1} \to \mathbb{N}$. The new function calculates for given $\vec{n}$ the least m such that $\chi(\vec{n}, m) = 0$, if there is such an m and is undefined otherwise.

Suppose χ is $\lambda_{\mathcal{U}}$-defined by G. We will now represent $\mu m[\chi(\vec{n}, m) = 0]$ by the lambda term $\lambda\vec{n}.((\lambda p.H_p\ulcorner 0\urcorner)\ \lambda m.\textbf{Zero}(G\vec{n}m))$. we can safely say that this representation preserves the definition of minimalisation. After all, instead of the notation $\mu m[\chi(\vec{n}, m) = 0]$ one could just as well have opted for $\mu[\lambda m.\chi(\vec{n}, m) = 0]$ instead.

In [3] we find the following proposition:

▶ **Proposition 12** ([3, Prop. 8.4.10]). *Let P be such that for all $n \in \mathbb{N}$ one has $P\ulcorner n\urcorner \twoheadrightarrow \mathbf{F}$. Then:*

1. *μP has no normal form,*
2. *μP is unsolvable.*

Since we want to generalise from the set of unsolvables to other sets of undefined terms, we can not use the previous proposition. However, we can reuse its proof, which actually shows that μP is rootactive.[7]

This leads us to a more general proposition:

▶ **Proposition 13.** *Let P be such that for all $n \in \mathbb{N}$ we have $P\ulcorner n\urcorner \twoheadrightarrow \mathbf{F}$. Then$\mu P$ is rootactive.*

Proof. Consider the following reduction from [3] that we reproduce here with slightly more detail:

$$\begin{array}{lll}
\mu P & \equiv & H_P\ulcorner 0\urcorner \\
& \twoheadrightarrow & \textbf{if } P\ulcorner 0\urcorner \textbf{ then } \ulcorner 0\urcorner \textbf{ else } H_P\ulcorner 1\urcorner \\
& \twoheadrightarrow & \textbf{if } \mathbf{F} \textbf{ then } \ulcorner 0\urcorner \textbf{ else } H_P\ulcorner 1\urcorner \\
& \equiv & \mathbf{F}\ulcorner 0\urcorner(H_P\ulcorner 1\urcorner) \\
& \equiv & (\lambda xy.y)\ulcorner 0\urcorner(H_P\ulcorner 1\urcorner) \\
& \to & (\lambda y.y)(H_P\ulcorner 1\urcorner) \\
& \to_0 & H_P\ulcorner 1\urcorner \\
& \twoheadrightarrow & \textbf{if } \mathbf{F} \textbf{ then } \ulcorner 1\urcorner \textbf{ else } H_P\ulcorner 2\urcorner \\
& \twoheadrightarrow & \lambda y.y(H_P\ulcorner 2\urcorner) \\
& \to_0 & H_P\ulcorner 2\urcorner \\
& \twoheadrightarrow & \ldots
\end{array}$$

In all segments $H_P\ulcorner n\urcorner \twoheadrightarrow H_P\ulcorner n+1\urcorner$ at least one reduction step takes place at the root (the step $\to_0$). Hence using the terminology of [13] the infinite reduction starting from μP

[7] It is well known that the set of rootactives is a proper subset of the set of unsolvables. E.g. $\lambda x.\Omega$, $\lambda x.\Omega x$ and $\mathbf{\Theta K}$ are unsolvables that are not rootactive.

is hypercollapsing.[8] Hence by [13, Theorem 12.8.3] we obtain that the initial term μP is rootactive. ◀

▶ **Lemma 14.** *Let $\mathcal{U}$ be a set of lambda terms such that $\mathcal{U}$ contains all rootactive terms and $U \in \mathcal{U}$ implies $U\mathbf{T}\ulcorner n\urcorner M \in \mathcal{U}$, for any $M \in \Lambda$. The strictly $\lambda_{\mathcal{U}}$-definable partial functions are closed under minimalisation.*

Proof. Let $\mathcal{U}$ be a set of terms such that $\mathcal{U}$ contains all rootactive terms and $U \in \mathcal{U}$ implies $U\mathbf{T}\ulcorner n\urcorner M \in \mathcal{U}$. Let $\phi(\vec{n}) \equiv \mu m[\chi(\vec{n}, m) = 0]$, where χ is total and $\lambda_{\mathcal{U}}$-definable by, say, G. Then we define:

$$F \equiv \lambda\vec{x}.\mu[\lambda y.\mathbf{Zero}(G\vec{x}y)].$$

If $\phi(\vec{n}) \downarrow$, then $\chi(\vec{n}, m) = 0$ for some $m \in \mathbb{N}$. Then by [3, Lemma 6.3.9(ii)] we get

$$F\ulcorner\vec{n}\urcorner = \ulcorner\phi(\vec{n})\urcorner.$$

And if $\phi(\vec{n}) \uparrow$, then $\chi(\vec{n}, m) \neq 0$ and so $\mathbf{Zero}(G\vec{n}m) \twoheadrightarrow_\beta \mathbf{F}$ for all $m \in \mathbb{N}$. Hence by Proposition 13 we see that

$$F\ulcorner\vec{n}\urcorner \twoheadrightarrow_\beta \mu[\lambda y.\mathbf{Zero}(G\vec{n}y)]$$

is rootactive. Finally, consider an $\vec{N} \in \Lambda$ with at least one $N_i \in \mathcal{U}$. Because G is strict, this implies the existence of a $U \in \mathcal{U}$ such that $G\vec{N}\ulcorner 0\urcorner \twoheadrightarrow_\beta U$. We can now make the following reduction:

$$\begin{array}{rcl}
F\vec{N} & \twoheadrightarrow_\beta & \mu[\lambda y.\mathbf{Zero}(G\vec{N}y)] \\
& \twoheadrightarrow_\beta & H_P\ulcorner 0\urcorner \\
& \twoheadrightarrow_\beta & \mathbf{if}\ P\ulcorner 0\urcorner\ \mathbf{then}\ \ulcorner 0\urcorner\ \mathbf{else}\ H_P\ulcorner 1\urcorner \\
& \twoheadrightarrow_\beta & \mathbf{if}\ \mathbf{Zero}(G\vec{N}\ulcorner 0\urcorner)\ \mathbf{then}\ \ulcorner 0\urcorner\ \mathbf{else}\ H_P\ulcorner 1\urcorner \\
& \twoheadrightarrow_\beta & \mathbf{if}\ \mathbf{Zero}\ U\ \mathbf{then}\ \ulcorner 0\urcorner\ \mathbf{else}\ H_P\ulcorner 1\urcorner \\
& \equiv & \mathbf{Zero}\ U\ \ulcorner 0\urcorner\ (H_P\ulcorner 1\urcorner) \\
& \equiv & (\lambda x.x\mathbf{T})U\ulcorner 0\urcorner(H_P\ulcorner 1\urcorner) \\
& \twoheadrightarrow_\beta & U\mathbf{T}\ulcorner 0\urcorner(H_P\ulcorner 1\urcorner)
\end{array}$$

where P stands for $\lambda y.\mathbf{Zero}(G\vec{N}y)$. The last term $U\mathbf{T}\ulcorner 0\urcorner(H_P\ulcorner 1\urcorner)$ is undefined because of our assumption on $\mathcal{U}$.

Concluding, we have shown that $\phi(\vec{n})$ is strictly $\lambda_{\mathcal{U}}$-definable by F. ◀

3.3 Total recursive functions

After composition and minimalisation we will now look at a strict encoding of the total recursive functions. This presents another obstacle: the usual representation of a total recursive function is not strict. Consider for instance the constant 0 function represented by $\mathbf{Z} \equiv \lambda x.\ulcorner 0\urcorner$ in [3]. We get:

$$(\lambda x.\ulcorner 0\urcorner)\mathbf{\Omega} \equiv (\lambda x.\mathbf{I})\mathbf{\Omega} \to \mathbf{I}.$$

[8] A hypercollapsing reduction is a "quasi root reduction," i.e. a reduction containing infinitely many root reduction steps of the form $(\lambda x.M)N \to_0 M[x := N]$.

Our previous analysis of minimalisation forced upon us the condition for $\mathcal{U}$, that if $U \in \mathcal{U}$ then also $U\mathbf{T}XY \in \mathcal{U}$ for any $X, Y \in \Lambda$. Now, if $\mathbf{I}$ would be an element of $\mathcal{U}$, then for any $M \in \Lambda$ we would get:

$$\mathbf{IT}MM \equiv \mathbf{IK}MM \to_\beta \mathbf{K}MM \to_\beta M \in \mathcal{U}.$$

This can not be the case, as the numerals are not supposed to be undefined.

There is, however, a strict way of representing the constant zero function. Consider the following infinite expression:

$$\lambda x.\mathbf{if}\ x = \ulcorner 0 \urcorner\ \mathbf{then}\ \ulcorner 0 \urcorner\ \mathbf{else}\ \mathbf{if}\ x = \ulcorner 1 \urcorner\ \mathbf{then}\ \ulcorner 0 \urcorner\ \mathbf{else}\ \mathbf{if}\ x = \ulcorner 2 \urcorner\ \mathbf{then}\ \ulcorner 0 \urcorner\ \mathbf{else}\ \dots$$

in which we use $x = \ulcorner m \urcorner$ as shorthand for $\mathbf{Zero}(\mathbf{P}^{-m}\ulcorner x \urcorner)$. Clearly, this expression will reduce to $\ulcorner 0 \urcorner$ whenever x is a numeral $\ulcorner n \urcorner$. The above infinite expression is of the simple form

$$\lambda x.\mathbf{if}\ x = \ulcorner 0 \urcorner\ \mathbf{then}\ X\ \mathbf{else}\ Y \text{ for some possibly infinite expressions } X, Y.$$

If we provide the previous term with input $U \in \mathcal{U}$, we get:

$$\begin{array}{rcl}
\mathbf{if}\ U = \ulcorner 0 \urcorner\ \mathbf{then}\ X\ \mathbf{else}\ Y' & \twoheadrightarrow_\beta & \lambda x.\mathbf{if}\ \mathbf{Zero}\ U\ \mathbf{then}\ X\ \mathbf{else}\ Y \\
& \twoheadrightarrow_\beta & \mathbf{Zero}\ UXY \\
& \equiv & (\lambda x.x\mathbf{T})UXY \\
& \twoheadrightarrow_\beta & U\mathbf{T}XY
\end{array}$$

We find that expression $\lambda x.\mathbf{if}\ x = \ulcorner 0 \urcorner\ \mathbf{then}\ X\ \mathbf{else}\ Y$ is strict for those sets $\mathcal{U}$ that satisfy the same property that we needed in Lemma 14, namely that $U \in \mathcal{U}$ must imply $U\mathbf{T}\ulcorner 0 \urcorner M \in \mathcal{U}$ for any $M \in \Lambda$.

In general, where Barendregt would use F to represent a total unary function f, we transform his F to a finite term which can reduce to an infinite representation for f:

$$\lambda n.\mathbf{if}\ n = \ulcorner 0 \urcorner\ \mathbf{then}\ F\ulcorner 0 \urcorner\ \mathbf{else}\ \mathbf{if}\ n = \ulcorner 1 \urcorner\ \mathbf{then}\ F\ulcorner 1 \urcorner\ \mathbf{else}\ \mathbf{if}\ n = \ulcorner 2 \urcorner\ \mathbf{then}\ F\ulcorner 2 \urcorner\ \mathbf{else}\ \dots.$$

This representation is strict, with a similar argument. And for all input of the form $\ulcorner n \urcorner$ with $n \in \mathbb{N}$ the expression reduces to $F\ulcorner n \urcorner$. Thanks to the fixed-point trickery of Turing [22] there is a finite term that reduces to this this infinite expression:

$$L \equiv \lambda f.\mathbf{\Theta}(\lambda wmn.(\mathbf{Zero}\ n)(fm)(w(\mathbf{P}^-\ n)(\mathbf{S}^+\ m)))$$

If F is the Church-Barendregt encoding of a total unary numeric function f, we get:

$$\begin{array}{rcl}
LF\ulcorner 0 \urcorner\ulcorner n \urcorner & \twoheadrightarrow & \mathbf{if}\ \mathbf{Zero}\ulcorner n \urcorner\ \mathbf{then}\ F\ulcorner 0 \urcorner\ \mathbf{else}\ KF\ulcorner 1 \urcorner(\mathbf{P}^-\ulcorner n \urcorner) \\
& \twoheadrightarrow_\beta & \mathbf{if}\ \mathbf{Zero}\ (\mathbf{P}^-\ulcorner n \urcorner)\ \mathbf{then}\ F\ulcorner 1 \urcorner\ \mathbf{else}\ LF\ulcorner 2 \urcorner(\mathbf{P}^{-2}\ulcorner n \urcorner) \\
& \twoheadrightarrow_\beta & \dots \\
& \twoheadrightarrow_\beta & \mathbf{if}\ \mathbf{Zero}\ (\mathbf{P}^{-n}\ulcorner n \urcorner)\ \mathbf{then}\ F\ulcorner n \urcorner\ \mathbf{else}\ LF\ulcorner n+1 \urcorner(\mathbf{P}^{-n+1}\ulcorner n)\urcorner \\
& \twoheadrightarrow_\beta & \mathbf{if}\ \mathbf{Zero}\ \ulcorner 0 \urcorner\ \mathbf{then}\ F\ulcorner n \urcorner\ \mathbf{else}\ LF\ulcorner n+1 \urcorner(\mathbf{P}^{-n+1}\ulcorner n)\urcorner \\
& \twoheadrightarrow_\beta & F\ulcorner n \urcorner
\end{array}$$

▶ **Lemma 15.** *Let $\mathcal{U}$ be a set of lambda terms such that $U \in \mathcal{U}$ implies $UXYZ \in \mathcal{U}$, for any $X, Y, Z \in \Lambda$. Then the total unary recursive functions are strictly $\lambda_\mathcal{U}$-definable.*

Proof. If F is the Barendregt representation of the total unary recursive function f, then we will represent f now by $\lambda x.LF\ulcorner 0 \urcorner x$. ◀

▶ **Corollary 16.** *The function $S^+(n) = n + 1$ is strictly $\lambda_{\mathcal{U}}$-definable by $\lambda x.L\mathbf{S}^+\ulcorner 0\urcorner x$ and the function $Z(n) = 0$ is strictly $\lambda_{\mathcal{U}}$-definable by $\lambda x.L\mathbf{Z}x$, for any set $\mathcal{U}$ of lambda terms satisfying the condition $U \in \mathcal{U}$ implies $U\mathbf{T}XY \in \mathcal{U}$, for any $X, Y \in \Lambda$.*

Next we want to show that p-ary projections functions U^p_i can be strictly $\lambda_{\mathcal{U}}$-defined. With this goal in mind, consider the case where f is a constant function $\lambda n.g$ and g is total p-ary recursive function that can be $\lambda_{\mathcal{U}}$-defined by some lambda term G. Then we represent f by

$$\lambda n.L(\lambda m.G)\ulcorner 0\urcorner n$$

which reduces to the infinite term

$$\lambda n.\mathbf{if}\ n = \ulcorner 0\urcorner\ \mathbf{then}\ G\ \mathbf{else\ if}\ n = \ulcorner 1\urcorner\ \mathbf{then}\ G\ \mathbf{else\ if}\ n = \ulcorner 2\urcorner\ \mathbf{then}\ G\ \mathbf{else}\ \ldots.$$

▶ **Lemma 17.** *The functions $U^p_i \equiv \lambda x_1 \ldots x_p.x_i$ with $0 \le i \le p$ are strictly $\lambda_{\mathcal{U}}$-definable, for any set $\mathcal{U}$ of lambda terms satisfying the condition $U \in \mathcal{U}$ implies $U\mathbf{T}XY \in \mathcal{U}$, for any $X, Y \in \Lambda$.*

Proof Sketch. Note that U^p_i can be rewritten $\lambda\vec{x}.U^p_p\vec{y}$, where $\vec{y}$ is obtained from the sequence of variables $\vec{x}$ by moving x_i to the leftmost position. Next, note that $U^p_p \equiv \lambda x_1.(\lambda x_2.\ldots(\lambda x_p.x_p))$.

The recursive identity function $\lambda x_p.x_p$ is strictly $\lambda_{\mathcal{U}}$-definable by F_p with

$$F_p \equiv \lambda x_p.L\mathbf{I}\ulcorner 0\urcorner x_p.$$

But then $\lambda x_{p-1}x_p.x_p$, that is the constant function $\lambda x_{p-1}.(\lambda x_p.x_p)$, can be represented by F_{p-1} with

$$F_{p-1} \equiv \lambda x_{p-1}.L(\lambda m.F_p)\ulcorner 0\urcorner x_{p-1}.$$

We continue this process until we find that U^p_p can be represented by

$$F_1 \equiv \lambda x_0.L(\lambda m.F_1)\ulcorner 0\urcorner x_0.$$ ◀

Together, the functions of this lemma and the previous corollary are called the initial functions.

Summarising: in this section we found that, modulo the condition that $U \in \mathcal{U}$ implies $U\mathbf{T}XY \in \mathcal{U}$ for any $X, Y \in \Lambda$, all unary total recursive functions and all initial functions are strictly $\lambda_{\mathcal{U}}$-definable. But we left open whether all p-ary total recursive functions are strictly $\lambda_{\mathcal{U}}$-definable. Hence we can not yet conclude that all partial recursive functions are strictly $\lambda_{\mathcal{U}}$-definable. To obtain this conclusion we must first show closure under primitive recursion.

3.4 Primitive recursion

▶ **Lemma 18.** *The strictly $\lambda_{\mathcal{U}}$-definable partial functions are closed under primitive recursion, for any set $\mathcal{U}$ of lambda terms satisfying the condition $U \in \mathcal{U}$ implies $U\mathbf{T}XY \in \mathcal{U}$, for any $X, Y \in \Lambda$.*

Proof. We can mimic the proof of [3, Lemma 6.3.7] replacing λ-definable by strictly $\lambda_{\mathcal{U}}$-definable. Since the representing term F given there is of the form $\mathbf{if\ Zero}\ x\ \mathbf{then}\ X\ \mathbf{else}\ Y$, we get strictness. ◀

3.5 Undefined is satisfying certain conditions

We will show that the partial functions are *strictly $\lambda_{\mathcal{U}}$-definable* for any $\mathcal{U}$ satisfying certain conditions. The conditions we have seen so far are not yet enough.

▶ **Lemma 19.** *If F $\lambda_{\mathcal{U}}$-defines a p-ary partial function ϕ, then for all $n \in \mathbb{N}^p, m \in \mathbb{N}$:*

1. *$\phi(\vec{n}) = m$ iff $F\ulcorner\vec{n}\urcorner = \ulcorner m\urcorner$ and*
2. *$\phi(\vec{n})\uparrow$ iff $F\ulcorner\vec{n}\urcorner \in \mathcal{U}$,*

provided $\mathcal{U}$ satisfies the conditions:

1. *$M \notin \mathcal{U}$, for any M such that $M \twoheadrightarrow \ulcorner n\urcorner$ for some $n \in \mathbb{N}$ and*
2. *$\mathcal{U}$ is closed under reduction.*

Proof. One direction is by definition for both items.

1. If $F\ulcorner\vec{n}\urcorner = \ulcorner m\urcorner$ then, because $\ulcorner m\urcorner \notin \mathcal{U}$, $\phi(\vec{n})\downarrow$ and $\phi(\vec{n}) = m'$. But then $\ulcorner m\urcorner = \ulcorner m'\urcorner$ and hence $m = m'$. This argument is exactly as in [3, Lemma 8.4.12], but we use that $\ulcorner m\urcorner \notin \mathcal{U}$, where Barendregt uses that $\ulcorner m\urcorner$ is solvable.
2. Next suppose $F\ulcorner\vec{n}\urcorner = U \in \mathcal{U}$, then $F\ulcorner\vec{n}\urcorner \neq \ulcorner m\urcorner$ for all $m \in \mathbb{N}$. Suppose $F\ulcorner\vec{n}\urcorner = \ulcorner m\urcorner$ for some $m \in \mathbb{N}$. Then U and $\ulcorner m\urcorner$ have no common reduct, because by the conditions on $\mathcal{U}$ any reduct of U belongs to $\mathcal{U}$, while the normal form $\ulcorner m\urcorner$ does not. Therefore $\phi(\vec{n}) \neq m$ for all $m \in \mathbb{N}$. Hence $\phi(\vec{n})\uparrow$. ◀

Let us now take the standard definition of partial recursive functions as the smallest class containing the initial functions, and closed under primitive recursion and minimalisation. This is equivalent to the definition used in [3] that we repeated in Section 3.

▶ **Theorem 20.** *Let $\mathcal{U}$ be a set of lambda terms such that*

1. *$M \notin \mathcal{U}$, for any M such that $M \twoheadrightarrow \ulcorner n\urcorner$ for some $n \in \mathbb{N}$.*
2. *$\mathcal{U}$ is closed under reduction.*
3. *$\mathcal{U}$ contains all rootactive terms.*
4. *$U \in \mathcal{U}$ implies $U\mathbf{T}M_1M_2 \in \mathcal{U}$, for any $M_1, M_2 \in \Lambda$.*

Then a partial function $\phi : \mathbb{N}^p \nrightarrow \mathbb{N}$ is strictly $\lambda_{\mathcal{U}}$-definable if and only if ϕ is partial recursive.

Proof. By Corollary 16 and Lemmas 17, 11, 18 and 14, and conditions 3 and 4 on $\mathcal{U}$ it follows that all partial recursive functions are $\lambda_{\mathcal{U}}$-definable.

For the converse, assume ϕ is strictly $\lambda_{\mathcal{U}}$-definable. Then by Lemma 19 and conditions 1 and 2 on $\mathcal{U}$ we get for all $n, m \in \mathbb{N}$: $\phi(n) = m$ iff $\lambda\beta \vdash F\ulcorner n\urcorner = \ulcorner m\urcorner$. As we can recursively enumerate all conversions of the form $F\ulcorner n\urcorner = \ulcorner m\urcorner$ that can be derived in the classical lambda calculus, it follows that the graph of ϕ is recursive enumerable as well. Hence ϕ is partial recursive. ◀

4 Proposal for a definition of a set of undefined terms

In Theorem 20 we needed four conditions on $\mathcal{U}$. Let us promote them to a definition.

▶ **Definition 21.** A set $\mathcal{U}$ of lambda terms is a *set of undefined terms* if it satisfies the following conditions:

1. $M \notin \mathcal{U}$, for any M such that $M \twoheadrightarrow \ulcorner n\urcorner$ for some $n \in \mathbb{N}$.
2. $\mathcal{U}$ is closed under reduction.
3. $\mathcal{U}$ contains all rootactive terms.
4. $U \in \mathcal{U}$ implies $U\mathbf{T}M_1, M_2 \in \mathcal{U}$, for any $M_1, M_2 \in \Lambda$.

Theorem 20 implies that this collection of condition is sufficient to prove that a partial function $\phi : \mathbb{N} \rightharpoonup \mathbb{N}$ is strictly $\lambda_{\mathcal{U}}$-definable if and only if ϕ is partial recursive.

We leave it open whether this set of conditions is necessary. But we consider the above set of conditions to be a reasonable first attempt at defining the concept of a set of undefined terms. Apart from condition 4, perhaps, these conditions feel quite natural.

Let us now go back to the four sets of terms $\overline{\mathcal{NF}}$, $\overline{\mathcal{HNF}}$, $\overline{\mathcal{WHNF}}$ and $\mathcal{E}$ that satisfied the condition of Statman's theorem. Note that condition 4 is implied by the condition (*) $U \in \mathcal{U}$ implies $UM \in \mathcal{U}$, for any $M \in \Lambda$. It is not difficult to see that $\overline{\mathcal{NF}}$, $\overline{\mathcal{HNF}}$ and $\overline{\mathcal{WHNF}}$ satisfy all four conditions of Definition 21. In case of the set $\mathcal{E}$ of easy terms condition 3 and (*) are well known. Hence condition 4 holds as well for $\mathcal{E}$. To show condition 1 we need a lemma, that likely belongs to folklore.

▶ **Lemma 22.** *The set $\mathcal{E}$ of easy terms is a set of undefined terms in the sense of Definition 21.*

Proof.

1. The numerals $\ulcorner n \urcorner$ are all $\beta\eta$-normal form. Hence by an application of Böhm's theorem [3, Corollary 10.4.3] they can not be easy.
2. This follows directly from the definition of easy term.
3. Condition 3 is shown in [6].
4. Condition (*) goes at least back to [12]. Hence condition 4 holds as well. ◀

5 Recap of definition of set of meaningless terms

Sets of meaningless terms were studied in the context of infinite lambda calculus. Adding infinite terms and infinite reductions that converge to a limit to the finite lambda calculus results is a calculus that is not confluent with respect to infinitary reduction. The construction of the Böhm model hints at the solution. First we add a fresh symbol $\perp$ to the syntax of finite lambda calculus and consider the set $\Lambda^{\infty}_{\perp}$ of finite and infinite λ-terms

$$M ::=_{coinduction} \perp \mid x \mid (\lambda x M) \mid (MM)$$

By Λ^{∞} we denote the subset of finite and infinite terms not containing $\perp$. Next, we choose a set $\mathcal{U} \subseteq \Lambda^{\infty}$ and add a new rule for some set $\mathcal{U} \subseteq \Lambda^{\infty}$:

$$\frac{M[\perp := \mathbf{\Omega}] \in \mathcal{U} \quad M \neq \perp}{M \to \perp}\,(\perp_{\mathcal{U}})$$

The resulting infinitary lambda calculus we denote by $\lambda^{\infty}_{\beta\perp_{\mathcal{U}}}$. We use the notation of [13]: $\twoheadrightarrow$ stands for finite reduction as in [3] and $\twoheadrightarrow\!\!\!\!\!\twoheadrightarrow$ stands for strongly converging (in-)finite reduction.

▶ **Definition 23** ([20])**.** $\mathcal{U} \subseteq \Lambda^{\infty}$ is called a *set of (finite or infinite) meaningless terms*, if it satisfies the axioms of meaninglessness:

1. *Axiom of Rootactiveness*: $\mathcal{R} \subseteq \mathcal{U}$.
2. *Axiom of Closure under β-reduction*: If $M \twoheadrightarrow\!\!\!\!\!\twoheadrightarrow_{\beta} N$ implies $N \in \mathcal{U}$ for all $M \in \mathcal{U}$.
3. *Axiom of Closure under Substitution*: If $M \in \mathcal{U}$ then any substitution instance of M is an element of $\mathcal{U}$.
4. *Axiom of (Weak) Overlap*: Either for each $\lambda x.P \in \mathcal{U}$, there is some $W \in \mathcal{U}$ such that $P \twoheadrightarrow\!\!\!\!\!\twoheadrightarrow_{\beta} Wx$, or alternatively $(\lambda x.P)Q \in \mathcal{U}$, for any $Q \in \Lambda^{\infty}_{\perp}$.
5. *Axiom of Indiscernibility*: Define $M \overset{\mathcal{U}}{\leftrightarrow} N$ if M can be transformed into N by replacing pairwise disjoint subterms of M in $\mathcal{U}$ by terms in $\mathcal{U}$. If $M \overset{\mathcal{U}}{\leftrightarrow} N$ then $M \in \mathcal{U} \Leftrightarrow N \in \mathcal{U}$.

The set Λ^∞ satisfies all these conditions. But the resulting lambda calculus is inconsistent, as all elements reduce to $\bot$. So the sets that we are interested in should be non-trivial.

▶ **Theorem 24** ([20]). *If $\mathcal{U}$ is a meaningless set, then $\lambda^\infty_{\beta\bot_\mathcal{U}}$ is confluent for infinitary β-reduction.*

For the converse we need one more condition: $\mathcal{U}$ is called closed under $\beta\bot$-expansion from $\bot$ if $N \twoheadrightarrow_{\beta\bot} \bot$ implies $N \in \mathcal{U}$ for all $N \in \Lambda^\infty$. Under this natural condition we have

▶ **Theorem 25** ([20]). *Let $\mathcal{U}$ satisfies Closure under $\beta\bot$-Expansion from $\bot$. If $\lambda^\infty_{\beta\bot_\mathcal{U}}$ is confluent, then $\mathcal{U}$ is a meaningless set.*

5.1 Sets of finite meaningless terms

There is now a mismatch: undefined terms are always finite and meaningless terms can be infinite. This can be reconciled. Instead of using the full set Λ^∞ we restrict to the closure Λ^{inf} of Λ under strongly convergent reduction. The previous two theorems hold for Λ^{inf} as well. We say that $\mathcal{U}$ is a set of *finite meaningless terms*, if its closure under strongly converging reduction is a set of meaningless terms. From now, whenever we write set of meaningless terms we mean a set of finite meaningless terms.

Let us go once more back to the four sets of terms $\overline{\mathcal{NF}}$, $\overline{\mathcal{HNF}}$, $\overline{\mathcal{WHNF}}$ and $\mathcal{E}$ that satisfied the condition of Statman's theorem. Of the four, $\overline{\mathcal{NF}}$ is the largest, and $\overline{\mathcal{HNF}}$ the second largest. Both $\overline{\mathcal{WHNF}}$ and $\mathcal{E}$[9] are subsets of $\overline{\mathcal{HNF}}$. It is well known that $\overline{\mathcal{HNF}}$ and $\overline{\mathcal{WHNF}}$ are sets of (finite) meaningless terms [14]. The other two are not:

▶ **Lemma 26.**

1. *[14] The set $\overline{\mathcal{NF}}$ does not satisfy Overlap.*
2. *The set $\mathcal{E}$ does not satisfy Indiscernibility.*

Proof.

1. $\lambda x.x\mathbf{I\Omega}$ has no finite normal form, but $(\lambda x.x\mathbf{I\Omega})\mathbf{K}$ reduces in two steps to the normal form $\mathbf{I}$. Note that the resulting extension infinitary term model is not consistent: $\mathbf{K} \leftarrow_\beta (\lambda x.x\mathbf{K\Omega})\mathbf{K} \rightarrow_\bot \bot \leftarrow_\bot (\lambda x.x\mathbf{K\Omega})\mathbf{I} \rightarrow_\beta \mathbf{I}$.
2. In [14] this was left open. But if we combine the fact that $\lambda z.\mathbf{\Omega}(\mathbf{\Theta}\lambda xyz.xzy)$ is an easy term [12][10] and the fact that $\lambda x.\Omega(xx)$ is not an easy term [7, Remark 6.2] with [21, Lemma 46(2)] (if a set of meaningless terms contains an abstraction, then it must contain all abstractions), we see that $\mathcal{E}$ can not be a set of meaningless terms. In [14] it has been shown that the first three properties hold, hence Indiscernibility does not hold. ◀

Since $\overline{\mathcal{HNF}}$ and $\overline{\mathcal{WHNF}}$ are sets of meaningless terms as well as sets of undefined terms, there is the natural question which other sets of meaningless terms can be taken as set of undefined terms. Statman's theorem is now of no help, as $\overline{\mathcal{HNF}}$ and $\overline{\mathcal{WHNF}}$ are the only sets of meaningless terms satisfying the Statman condition: the other sets of meaningless terms are not co-Visser sets. In the next section we will answer this question.

[9] It is straightforward to check that easy terms are unsolvable.

[10] The steps of the nice proof in [12] are: (1) $C \equiv \lambda xyz.xzy$ is right-invertible in $\lambda\beta\eta$. (2) $\mathbf{\Omega}$ is easy wrt $\lambda\beta\eta$. (3) C is easy wrt $\lambda\beta$. (4) If M is easy, then MN is easy for any N. (5) $C\mathbf{\Omega}(\mathbf{\Theta}(C\mathbf{\Omega}))$ is easy.

6 When is meaningless undefined?

▶ **Lemma 27** ([21]). *If $\mathcal{U}$ is a non-trivial set of finite or infinite meaningless terms (i.e. $\mathcal{U} \neq \Lambda^\infty$), then all its elements are unsolvable.*

▶ **Corollary 28.** *Let $\mathcal{U}$ be a non-trivial set of finite meaningless terms (i.e. $\mathcal{U} \neq \Lambda$) that satisfies Closure under $\beta\bot$-Expansion from $\bot$. Then $\mathcal{U}$ satisfies conditions 1, 2 and 3 of Definition 21.*

Proof. 2 and 3 are trivial. If $M \twoheadrightarrow \ulcorner n \urcorner$ for some $n \in \mathbb{N}$, then M reduces to a finite normal form. Hence M is solvable, because M has a head normal form. But then $M \notin \mathcal{U}$ by Lemma 27. ◀

There is a natural smallest set of meaningless terms satisfying condition 4 of Definition 21.

▶ **Definition 29.** Let us call lambda term M in Λ *almost rootactive*, if M can reduce to a term of the form $R\mathbf{T}M_1N_1 \ldots \mathbf{T}M_kN_k$ where R is rootactive and $M_i, N_i \in \Lambda$ for $1 \leq i \leq k$. Let $\mathcal{W}$ denote the set of almost rootactive terms.

Clearly the set $\mathcal{W}$ is the smallest set of *undefined* terms that satisfies the four conditions for a set of undefined terms. We also have that

▶ **Lemma 30.** *The set $\mathcal{W}$ is the smallest set of* meaningless *terms that is a set of undefined terms.*

Proof. With the techniques of [19, 21] one can show that $\mathcal{W}$ satisfies all conditions of a set of meaningless terms. ◀

7 Conclusion

We have presented a set of sufficient conditions on a set $\mathcal{U}$ of lambda terms such that a partial function $\phi : \mathbb{N}^p \nrightarrow \mathbb{N}$ is strictly $\lambda_{\mathcal{U}}$-definable if and only if ϕ is partial recursive. The smallest set $\mathcal{W}$ satisfying the these conditions is also a set of meaningless terms. By the Axiom of Indiscernibility it follows that any larger set of meaningless terms is also a set of undefined terms. Since $\mathcal{W}$ is larger than the set $\mathcal{R}$ of rootactibve terms, we conjecture that $\mathcal{R}$ can not be used to prove that a partial recursive function $\phi : \mathbb{N}^p \nrightarrow \mathbb{N}$ is strictly $\lambda_{\mathcal{R}}$-definable.

The notion of strict $\lambda_{\mathcal{U}}$-definability is forced when one searches for representations of partial recursive functions that preserve their definition. This has interesting consequences. E.g. the strict predicates suggest strongly that lambda calculus contains a many-valued logic related to McCarthy's calculus for three-valued sequential logic [8, 17].

Acknowledgements. I am grateful to Alessandro Berarducci, Alexander Kurz and the anonymous referees for helpful comments and suggestions.

References

1 H. P. Barendregt. *Some extensional term models for combinatory logics and λ-calculi.* PhD thesis, Univ. Utrecht, 1971.

2 H. P. Barendregt. A global representation of the recursive functions in the lambda-calculus. *Theor. Comput. Sci.*, 3(2):225–242, 1976. doi:10.1016/0304-3975(76)90025-6.

3 H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North-Holland, Amsterdam, Revised edition, 1984.

4 H.P. Barendregt. Solvability in lambda calculi. In M. Guillaume, editor, *Colloque international de logique: Clermont-Ferrand, 1975*, pages 209–219. Éditions du C.N.R.S., 1977.

5 H.P. Barendregt. Representing 'undefined' in lambda calculus. *J. Funct. Program.*, 2(3):367–374, 1992. doi:10.1017/S0956796800000447.

6 A. Berarducci. Infinite λ-calculus and non-sensible models. In *Logic and algebra (Pontignano, 1994)*, pages 339–377. Dekker, New York, 1996.

7 A. Berarducci and B. Intrigila. Church-Rosser lambda-theories, Infinite lambda-terms and consistency problems. In W. Hodges, M. Hyland, C. Steinhorn, and J. Truss, editors, *Logic: from Foundations to Applications*, pages 33–58. Oxford Science Publications, 1996. URL: http://www.dm.unipi.it/~berardu/Art/1996Church/CRtheories.pdf.

8 J. A. Bergstra and J. van de Pol. A calculus for four-valued sequential logic. *Theor. Comput. Sci.*, 412(28):3122–3128, 2011. doi:10.1016/j.tcs.2011.02.035.

9 A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936.

10 A. Church. *The Calculi of Lambda Conversion.* Princeton University Press, 1941.

11 H. B. Curry, J. R. Hindley, and J. P. Seldin. *Combinatory Logic II.* North-Holland, 1972.

12 G. Jacopini and M. Venturini-Zilli. Easy terms in the lambda-calculus. *Fundamenta Informaticae*, VIII(2):225–233, 1985.

13 J. R. Kennaway and F. J. de Vries. Infinitary rewriting. In Terese, editor, *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*, pages 668–711. Cambridge University Press, 2003.

14 J. R. Kennaway, V. van Oostrom, and F. J. de Vries. Meaningless terms in rewriting. *Journal of Functional and Logic Programming*, 1999(1), 1999. URL: http://danae.uni-muenster.de/lehre/kuchen/JFLP/articles/1999/A99-01/A99-01.html.

15 S. C. Kleene. λ-definability and recursiveness. *Duke Math. J.*, 2(2):340–353, 06 1936. doi:10.1215/S0012-7094-36-00227-2.

16 S. C. Kleene. On notation for ordinal numbers. *J. Symb. Log.*, 3(4):150–155, 1938. doi:10.2307/2267778.

17 Beata Konikowska, Andrzej Tarlecki, and Andrzej Blikle. A three-valued logic for software specification and validation. *Fundam. Inform.*, 14(4):411–453, 1991.

18 G. Kreisel. Some reasons for generalising recursion theory. In R.O. Gandy and C.M.E. Yates, editors, *Logic Colloquium 1969: proceedings of the summerschool and colloquium in mathematical logic, Manchester, August 1969*, pages 139–189. North-Holland, 1971.

19 P. Severi and F. J. de Vries. Order Structures for Böhm-like models. In *Computer Science Logic*, volume 3634 of *LNCS*, pages 103–116. Springer-Verlag, 2005.

20 P. Severi and F. J. de Vries. Weakening the axiom of overlap in infinitary lambda calculus. In M. Schmidt-Schauß, editor, *Proc. of the 22nd Int'l Conf. on Rewriting Techniques and Applications, May 30 – June 1, 2011*, volume 10 of *LIPIcs*, pages 313–328. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.RTA.2011.313.

21 P. G. Severi and F. J. de Vries. Decomposing the lattice of meaningless sets in the infinitary lambda calculus. In L. D. Beklemishev and R. de Queiroz, editors, *Proc. of the 18th Int'l Workshop on Logic, Language, Information and Computation (WoLLIC 2011)*, pages 210–227, 2011. doi:10.1007/978-3-642-20920-8_22.

22 A. M. Turing. Computability and λ-definability. *J. Symb. Log.*, 2(4):153–163, 1937. doi:10.2307/2268280.

23 A. M. Turing. The 𝔭-function in λ-K-conversion. *J. Symb. Log.*, 2(4):164, 1937. doi:10.2307/2268281.

24 A. Visser. Numerations, λ-calculus & arithmetic. In J.R. Hindley and J.P. Seldin, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, pages 259–284. Academic Press, 1980.

The Intersection Type Unification Problem

Andrej Dudenhefner[1], Moritz Martens[2], and Jakob Rehof[3]

1 **Department of Computer Science, Technical University of Dortmund, Dortmund, Germany**
andrej.dudenhefner@cs.tu-dortmund.de

2 **Department of Computer Science, Technical University of Dortmund, Dortmund, Germany**
moritz.martens@cs.tu-dortmund.de

3 **Department of Computer Science, Technical University of Dortmund, Dortmund, Germany**
jakob.rehof@cs.tu-dortmund.de

Abstract

The intersection type unification problem is an important component in proof search related to several natural decision problems in intersection type systems. It is unknown and remains open whether the unification problem is decidable. We give the first nontrivial lower bound for the problem by showing (our main result) that it is exponential time hard. Furthermore, we show that this holds even under rank 1 solutions (substitutions whose codomains are restricted to contain rank 1 types). In addition, we provide a fixed-parameter intractability result for intersection type matching (one-sided unification), which is known to be NP-complete.

We place the intersection type unification problem in the context of unification theory. The equational theory of intersection types can be presented as an algebraic theory with an ACI (associative, commutative, and idempotent) operator (intersection type) combined with distributivity properties with respect to a second operator (function type). Although the problem is algebraically natural and interesting, it appears to occupy a hitherto unstudied place in the theory of unification, and our investigation of the problem suggests that new methods are required to understand the problem. Thus, for the lower bound proof, we were not able to reduce from known results in ACI-unification theory and use game-theoretic methods for two-player tiling games.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Intersection Type, Equational Theory, Unification, Tiling, Complexity

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.19

1 Introduction

Intersection type systems occupy a prominent place within the theory of typed λ-calculus [5]. As is well known, variants of such systems characterize deep semantic properties of λ-terms, including normalization and solvability properties [5]. As a consequence of the enormous expressive power of intersection types, standard type-theoretic decision problems are undecidable for general intersection type systems, including the problem of type checking (given a term and a type, does the term have the type?) and inhabitation (given a type, does there exist a term having the type?). A combinatorial problem centrally placed in many classical type-theoretic decision problems is that of *type unification*: given two types σ and τ, does there exist a substitution S of types for type variables such that $S(\sigma) = S(\tau)$ in a suitable equational theory $(=)$ of types? In this paper we wish to initiate a study of the problem of *intersection type unification* which we believe to be of considerable systematic interest.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 19; pp. 19:1–19:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We consider the standard equational theory of intersection types induced by a canonical subtyping relation for intersection types [4]. Although decidability of intersection type unification appears to be surprisingly difficult and remains open, the present paper provides the first nontrivial lower bound indicating that the problem is of very high complexity: we prove that the problem is Exptime-hard. Our proof uses game-theoretic methods, in the form of two-player tiling games, which we believe to be of intrinsic interest and potentially helpful towards understanding the problem of decidability. Moreover, as we will show, the intersection type unification problem occupies a natural but hitherto (so far as we are aware) unstudied place in the theory of unification. Thus, we hope with this paper to stimulate further work on a fascinating open problem in type theory as well as in unification theory.

We briefly summarize some of the most important algebraic properties of the equational theory of intersection types needed to appreciate the systematic placement of the unification problem (full details are given later in the paper). Intersection type systems are characterized by the presence of an associative, commutative, idempotent operator, $\cap$ (intersection), which allows the formation of types of the form $\sigma \cap \tau$. In addition, we have function types, $\sigma \to \tau$. The standard equational theory, denoted $=$, of intersection types [4] is induced from a partial order $\leq$ on types, referred to as subtyping, by taking type equality to be the relation $\leq \cap \leq^{-1}$. Conversely, as will be discussed in the paper, it is also possible to give a purely equational presentation of subtyping. Because intersection is greatest lower bound with respect to subtyping, the intersection type unification problem is equivalent to the subtype satisfiability problem: given σ and τ, does there exist a type substitution S such that $S(\sigma) \leq S(\tau)$? The latter is equivalent to $S(\sigma) \cap S(\tau) = S(\tau)$, hence satisfiability is reducible to unification. The equational theory includes right-distributivity of $\to$ over $\cap$: $\sigma \to (\tau_1 \cap \tau_2) = (\sigma \to \tau_1) \cap (\sigma \to \tau_2)$ and left-contravariance of $\to$ with respect to subtyping: $\sigma_1 \to \tau_1 \leq \sigma_2 \to \tau_2$ whenever $\sigma_2 \leq \sigma_1$ and $\tau_1 \leq \tau_2$. As a consequence, one has "half left-distributivity" of $\to$ over $\cap$: $(\sigma_1 \to \tau) \cap (\sigma_2 \to \tau) \leq (\sigma_1 \cap \sigma_2) \to \tau$ (but the symmetric relation does not hold). Altogether, we could say for short that $\to$ is "$1\frac{1}{2}$-distributive" over $\cap$. Axioms specific to a special largest type, ω, are added in some variants of the theory (both variants, with or without ω, are important in type theory), including the recursion axiom $\omega = \omega \to \omega$, and we have the derived equation $\sigma \to \omega = \omega$. Thus, ω is unit (neutral element) with respect to $\cap$ and right-absorbing element with respect to $\to$.

In the remainder of this section we consider the most closely related work within unification theory and type theory.

1.1 Related work in unification theory

The single most directly related piece of work in the literature is the study from 2004 by Anantharaman, Narendran, and Rusinowitch on unification modulo ACUI (associativity, commutativity, unit, idempotence) plus distributivity axioms [2]. They consider equational theories over a binary ACUI symbol, denoted $+$, together with a binary operator, $*$, which distributes (left, right, or both) over $+$. Indeed, since (as summarized above) we have an ACUI theory of $\cap$ together with $\to$ enjoying distributivity properties over $\cap$, it would seem that we are temptingly close to the theories studied in [2], by thinking of their $+$ as $\cap$ and their $*$ as $\to$. In particular, algebraically closest among the theories covered in that paper, ACUI-unification with one-sided (say, left) distributivity (ACUID$_l$) is shown to be Exptime-complete, using techniques from unification modulo homomorphisms [3]. But it turns out that there are fundamental obstacles to transferring results or techniques from ACUID$_l$-unification to intersection type unification, as will be summarized next.

With regard to any upper bound, the main obstacle is that, whereas decidability of the ACUID-problems can be relatively straight-forwardly obtained by appeal to an occurs-check (nontrivial cyclic equations have no solutions), this is very far from being clear in the case of intersection type unification. Indeed, even in the absence of the recursive type ω, we can solve nontrivial cyclic constraints, due to contravariance. For example, the constraint $\alpha \dot{\leq} \alpha \to b$ (where $\dot{\leq}$ denotes a formal subtyping constraint, α is a type variable and b is a constant) can be solved, e.g., by setting $S(\alpha) = b \cap (b \to b)$. The theory of intersection types is *non-structural* in the sense that types with significantly different shapes (tree domains, when types are regarded as labeled trees) may be related, and this presents fundamental obstacles for bounding the depth of substitutions via any kind of standard occurs-check. Although "$1\frac{1}{2}$-distributivity" of → over ∩ may at first sight appear to be algebraically close to the ACUID-framework of [2], the contravariant "$\frac{1}{2}$-distributivity" makes the theory of intersection types significantly different. We cannot exclude that some kind of restricted occurs-check might be possible, but our investigations lead us to believe that, in case it exists, it is likely to be very complicated, and we have been unable to find such a bounding principle. Hence, decidability remains a challenging open problem.

With regard to the exponential time lower bound, the results of [2] (in fact, both the Exptime upper and lower bounds) rely essentially on reductions from unification modulo a set H of noncommuting homomorphisms (ACUIDH), which was shown to be Exptime-complete in [3]. The basic idea is to represent unification with distributivity to unification modulo homomorphisms by replacing $s * t$ by $h_s(t)$ where h_s is a homomorphism with respect to the AC(U)I-theory. However, again, such techniques fail in our case due to contravariance. The equational presentation of the theory of intersection types captures contravariant subtyping by the absorption axiom (written in the algebraic notation of [2]): $s*t = s*t+(s+s')*t$. One could attempt to represent this axiom by $h_s(t) = h_s(t) + h_{s+s'}(t)$. But here the expression $h_{s+s'}(t)$ does not fall within the homomorphic format, and it is therefore not clear how the homomorphic framework could be applied. Moreover, the bounding problem discussed above leads to the problem that it is not clear how the theory could be adequately represented using only a finite set of homomorphisms. We concluded that we need new methods in order to make progress on understanding lower bounds for intersection type unification, and the route we present in this paper for the Exptime-lower bound is entirely different, relying on game theoretical results on tiling problems.

1.2 Related work in type theory

It may be surprising that computational properties (decidability, complexity) of the intersection type unification problem have not previously been systematically pursued *per se*. The theory of intersection type subtyping and its equational counterpart have rather been studied from semantic (operational and denotational) perspectives. Indeed, as mentioned already, the intersection type system captures deep operational properties of λ-terms, and undecidability of type checking and typability follows immediately. The theories of intersection type subtyping and equality studied here arose naturally out of model-theoretic considerations. For example, a fundamental result [14, 4] shows that intersection type subtyping and equality are sound and complete for set-theoretic containment in a class of λ-models: $\sigma \leq \tau$ holds, if and only if $[\![\sigma]\!]_v^{\mathcal{M}} \subseteq [\![\tau]\!]_v^{\mathcal{M}}$ for all models $\mathcal{M}$ in the class and valuations v. The intersection type unification problem can therefore also be endowed with semantic interpretations.

Several extensions and variations of the standard algebraic operations of unification studied here have been considered in connection with intersection type systems, foremostly motivated by questions related to notions of principality (principal types, principal typings, principal pairs) in such systems. Ronchi della Rocca, working from such motivations, defines a notion of unification in [21] and gives a semi-decision procedure for the corresponding unification problem. But that problem involves operations (chains of substitutions together with special expansion operations) which are not present in the algebraic notion of unification we consider here. Similarly, so-called expansion variables with associated operations have been used by Kfoury and Wells to characterize principality properties [17] and so-called β-unification involving expansion variables has been shown to characterize strong normalization in the λ-calculus [16], see also [10, 6]. The algebraic unification problem considered here is a centrally placed component in most forms of proof search related to intersection type systems. For example, it is not difficult to see that type checking parametric functions (or, combinatory expressions [15]) with intersection type schemes contains intersection type unification (we will give some concrete examples below in the paper). The problem is therefore likely to be involved as soon as one attempts to combine intersection types with usual notions of type instantiation. A recent example is the so-called type tallying problem of [7], which is not known to be decidable and is closely related to the intersection type satisfiability problem.

Summarizing the situation with regard to intersection type unification within type theory, it appears to hold an interesting and rather unexplored intermediate position: it is contained in many decision problems associated with intersection type systems, it is known to be expressive enough to capture certain restrictions of the type system, but it is not known whether it is decidable. It is therefore also a problem of importance for advancing our understanding of restrictions of the intersection type system and computational properties of associated decision problems.

Organization of the paper. The remainder of this paper is organized as follows. Intersection types are introduced in Sec. 2 together with the standard theory of subtyping [4]. In Sec. 3 we briefly study the matching problem (one-sided unification) as a natural preparation for considering the unification problem. The unification problem is studied in Sec. 4, which contains our main result. We first introduce the unification problem and the equational theory of intersection types (Sec. 4.1) and then turn to the proof of the EXPTIME-lower bound. We introduce tiling games (Sec. 4.2) and prove EXPTIME-completeness of a special form of such ("spiral tiling games"), which is then used (Sec. 4.3) in our reduction to unification and satisfiability. We conclude the paper in Sec. 5.

2 Intersection types

▶ **Definition 1** ($\mathbb{T}$)**.** The set $\mathbb{T}$ of intersection types, ranged over by σ, τ, ρ, is given by

$$\mathbb{T} \ni \sigma, \tau, \rho \ ::= \ a \mid \alpha \mid \omega \mid \sigma \to \tau \mid \sigma \cap \tau$$

where $a, b, c, \ldots$ range over type constants $\mathbb{C}$, ω is a special (universal) constant, and α, β, γ range over type variables $\mathbb{V}$.

As a matter of notational convention, function types associate to the right, and $\cap$ binds stronger than $\to$. A type $\tau \cap \sigma$ is said to have τ and σ as *components*.

▶ **Definition 2** (Subtyping $\leq$)**.** Subtyping $\leq$ is the least preorder (reflexive and transitive relation) over $\mathbb{T}$ (cf. [4]) such that

$$\sigma \leq \omega, \quad \omega \leq \omega \to \omega, \quad \sigma \cap \tau \leq \sigma, \quad \sigma \cap \tau \leq \tau, \quad (\sigma \to \tau_1) \cap (\sigma \to \tau_2) \leq \sigma \to \tau_1 \cap \tau_2,$$
$$\text{if } \sigma \leq \tau_1 \text{ and } \sigma \leq \tau_2 \text{ then } \sigma \leq \tau_1 \cap \tau_2, \quad \text{if } \sigma_2 \leq \sigma_1 \text{ and } \tau_1 \leq \tau_2 \text{ then } \sigma_1 \to \tau_1 \leq \sigma_2 \to \tau_2$$

Type equality, written $\sigma = \tau$, holds when $\sigma \leq \tau$ and $\tau \leq \sigma$, thereby making $\leq$ a partial order over $\mathbb{T}$. We use $\equiv$ for syntactic identity. By the axioms of subtyping, $\cap$ is associative, commutative, idempotent and has the following distributivity properties

$$(\sigma \to \tau_1) \cap (\sigma \to \tau_2) = \sigma \to (\tau_1 \cap \tau_2)\,,$$
$$(\sigma_1 \to \tau_1) \cap (\sigma_2 \to \tau_2) \leq (\sigma_1 \cap \sigma_2) \to (\tau_1 \cap \tau_2)\,.$$

We write $\bigcap_{i=1}^{n} \tau_i$ or $\bigcap_{i \in I} \tau_i$ or $\bigcap\{\tau_i \mid i \in I\}$ for an intersection of several components, where the empty intersection is identified with ω.

Using [4](Lemma 2.4.1) we syntactically define the set $\mathbb{T}^\omega$ of all types equal to ω.

▶ **Definition 3** ($\mathbb{T}^\omega$). The set $\mathbb{T}^\omega$ of types in $\mathbb{T}$ equal to ω is given by

$$\mathbb{T}^\omega \ni \sigma^\omega, \tau^\omega ::= \omega \mid \sigma \to \tau^\omega \mid \sigma^\omega \cap \tau^\omega\,.$$

▶ **Lemma 4.** *For $\tau \in \mathbb{T}$ we have $\tau \in \mathbb{T}^\omega$ iff $\tau = \omega$.*

▶ **Lemma 5** (Beta-Soundness [4, 5]). *Given $\sigma = \bigcap_{i \in I}(\sigma_i \to \tau_i) \cap \bigcap_{j \in J} a_j \cap \bigcap_{k \in K} \alpha_k$, we have:*

(i) *If $\sigma \leq a$ for some $a \in \mathbb{C}$, then $a \equiv a_j$ for some $j \in J$.*
(ii) *If $\sigma \leq \alpha$ for some $\alpha \in \mathbb{V}$, then $\alpha \equiv \alpha_k$ for some $k \in K$.*
(iii) *If $\sigma \leq \sigma' \to \tau' \neq \omega$ for some $\sigma', \tau' \in \mathbb{T}$, then $I' = \{i \in I \mid \sigma' \leq \sigma_i\} \neq \emptyset$ and $\bigcap_{i \in I'} \tau_i \leq \tau'$.*

▶ **Problem 6.** *(Subtyping) Given $\sigma, \tau \in \mathbb{T}$, does $\sigma \leq \tau$ hold?*

The subtyping relation is known to be decidable in polynomial time. The algorithm sketched in the proof of the following lemma gives an improved quadratic upper bound.

▶ **Lemma 7.** *Problem 6 (Subtyping) is decidable in time $\mathcal{O}(n^2)$ where n is the sum of the sizes of the input types σ and τ.*

Proof. For a polynomial time decision algorithm with a quartic upper bound see [19]. For a different approach with a quintic upper bound using rewriting see [22]. However, a quadratic upper bound to decide $\sigma \leq \tau$ is achievable using Lemmas 4 and 5. First, in linear time, subterms of σ and τ of the shape defined by $\mathbb{T}^\omega$ are replaced by ω. Second, in linear time, nested intersection are flattened using associativity of $\cap$ and components equal to ω are dropped. Third, in quadratic time, Lemma 5 is applied recursively using the additional property $\rho \leq \bigcap_{i \in I} \tau_i$ iff $\rho \leq \tau_i$ for $i \in I$. The invariant that $\cap$ is not nested and does not contain ω as component is ensured in recursive calls using linked lists with constant time concatenation to store components of intersections. ◀

We recapitulate the notion of paths and organized types introduced in [13].

▶ **Definition 8** (Paths $\mathbb{P}$). The set $\mathbb{P}$ of paths in $\mathbb{T}$, ranged over by π, is given by

$$\mathbb{P} \ni \pi ::= a \mid \alpha \mid \tau \to \pi\,.$$

▶ **Definition 9** (Organized type). A type τ is *organized*, if $\tau \equiv \omega$ or $\tau \equiv \bigcap_{i \in I} \pi_i$ for some paths π_i for $i \in I$.

A type can be organized (transformed to an equivalent organized type) in polynomial time. Note that an organized type is not necessarily normalized [14]. Normalization can lead to an exponential blow-up of type size.

▶ **Lemma 10.** *Given two organized types $\sigma \equiv \bigcap_{i\in I} \pi_i$ and $\tau \equiv \bigcap_{j\in J} \pi_j$, we have $\sigma \leq \tau$ iff for all $j \in J$ there exists an $i \in I$ with $\pi_i \leq \pi_j$.*

▶ **Corollary 11.** *Given a path $\pi \in \mathbb{P}$ and types σ, τ, we have $\sigma \cap \tau \leq \pi$ iff $\sigma \leq \pi$ or $\tau \leq \pi$.*

For the sake of completeness, we outline the corresponding type assignment system [4], also called **BCD** in literature. A basis (also called context) is a finite set $\Gamma = \{x_1 : \tau_1, \dots, x_n : \tau_n\}$, where the variables x_i are pairwise distinct; we set $\mathrm{dom}(\Gamma) = \{x_1, \dots, x_n\}$ and we write $\Gamma, x : \tau$ for $\Gamma \cup \{x : \tau\}$, where $x \notin \mathrm{dom}(\Gamma)$.

▶ **Definition 12** (Type Assignment). **BCD** type assignment is given by the following rules

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ (\mathrm{Ax}) \qquad \frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash \lambda x.e : \sigma \to \tau}\ (\to\mathrm{I}) \qquad \frac{\Gamma \vdash e : \sigma \to \tau \qquad \Gamma \vdash e' : \sigma}{\Gamma \vdash (e\ e') : \tau}\ (\to\mathrm{E})$$

$$\frac{}{\Gamma \vdash e : \omega}\ (\omega) \qquad \frac{\Gamma \vdash e : \sigma \qquad \Gamma \vdash e : \tau}{\Gamma \vdash e : \sigma \cap \tau}\ (\cap\mathrm{I}) \qquad \frac{\Gamma \vdash e : \sigma \qquad \sigma \leq \tau}{\Gamma \vdash e : \tau}\ (\leq)$$

3 Intersection type matching

In order to understand the unification problem it is useful first to investigate its restriction to matching (one-sided unification). Intersection type matching occurs naturally during proof search in intersection type systems and is known to be NP-complete [12]. We strengthen this result by showing that the problem remains so even when restricted to the fixed-parameter case where only a single type variable and only a single constant is used in the input.

For $\tau \in \mathbb{T}$ let $\mathrm{Var}(\tau) \subseteq \mathbb{V}$ denote the set of variables occurring in τ.

▶ **Problem 13** (Matching). *Given a set of constraints $C = \{\sigma_1 \dot{\leq} \tau_1, \dots, \sigma_n \dot{\leq} \tau_n\}$, where for each $i \in \{1, \dots, n\}$ we have $\mathrm{Var}(\sigma_i) = \emptyset$ or $\mathrm{Var}(\tau_i) = \emptyset$, is there a substitution $S \colon \mathbb{V} \to \mathbb{T}$ such that $S(\sigma_i) \leq S(\tau_i)$ for $1 \leq i \leq n$?*

We say that a substitution S satisfies $\{\sigma_1 \dot{\leq} \tau_1, \dots, \sigma_n \dot{\leq} \tau_n\}$ if $S(\sigma_i) \leq S(\tau_i)$ for $1 \leq i \leq n$.

▶ **Problem 14** (One-Sided Unification). *Given a set of constraints $C = \{\sigma_1 \doteq \tau_1, \dots, \sigma_n \doteq \tau_n\}$, where for each $i \in \{1, \dots, n\}$ we have $\mathrm{Var}(\sigma_i) = \emptyset$ or $\mathrm{Var}(\tau_i) = \emptyset$, is there a substitution $S \colon \mathbb{V} \to \mathbb{T}$ such that $S(\sigma_i) = S(\tau_i)$ for $1 \leq i \leq n$?*

Note that any matching constraint set $C = \{\sigma_1 \dot{\leq} \tau_1, \dots, \sigma_n \dot{\leq} \tau_n\}$ can be reduced to a single matching (resp. one-sided unification) constraint $\sigma \dot{\leq} \tau$ (resp. $\sigma \cap \tau \doteq \sigma$) with $\mathrm{Var}(\sigma) = \emptyset$ by fixing a type constant $\bullet \in \mathbb{C}$ to define

$$(\sigma_i', \tau_i') = \begin{cases} (\sigma_i \to \bullet, \tau_i \to \bullet) & \text{if } \mathrm{Var}(\sigma_i) = \emptyset \\ (\tau_i, \sigma_i) & \text{if } \mathrm{Var}(\tau_i) = \emptyset \end{cases} \quad \text{for } 1 \leq i \leq n$$

and $\sigma \equiv \sigma_1' \to \dots \to \sigma_n' \to \bullet$ and $\tau \equiv \tau_1' \to \dots \to \tau_n' \to \bullet$. By Lemma 5, for any substitution S we have $S(\sigma) \leq S(\tau)$ (resp. $S(\sigma \cap \tau) = S(\sigma)$) iff $S(\sigma_i) \leq S(\tau_i)$ for $1 \leq i \leq n$. Therefore, matching and one-sided unification remain NP-complete even restricted to single constraints.

In [12] the lower bound for matching is shown by reduction from 3-SAT and requires two type variables $\alpha_x, \alpha_{\neg x}$ for each propositional variable x. Since 3-SAT, parameterized by the number of propositional variables, is fixed parameter tractable, we naturally ask whether the same holds for matching (resp. one-sided unification) parameterized by the number of type variables.

▶ **Proposition 15.** *Problem 13 (Matching) is* NP*-hard even if only a single type variable and a single constant is used in the input.*

Proof. (Sketch) We fix a 3-SAT instance F containing clauses $(L_1 \vee L_2 \vee L_3) \in F$ over propositional variables V where L_i is either x or $\neg x$ for some $x \in V$. We reduce satisfiability of F to matching with one type variable α. First, we fix a set of type constants $B = V \cup \{\neg x \mid x \in V\}$ and the type constant $\bullet$. Let $\sigma_x \equiv \bigcap(B \setminus \{\neg x\})$ and $\sigma_{\neg x} \equiv \bigcap(B \setminus \{x\})$ for $x \in V$. We construct the set C containing following constraints

$$\begin{aligned}&\text{for } x \in V \text{ (consistency)}:\\ &\quad((\sigma_{\neg x} \to \bullet) \to (\neg x \to \bullet)) \cap ((\sigma_x \to \bullet) \to (x \to \bullet)) \stackrel{.}{\leq} (\alpha \to \bullet) \to (\alpha \to \bullet)\\ &\text{for } (L_1 \vee L_2 \vee L_3) \in F \text{ (validity)}:\\ &\quad(L_1 \to \bullet) \cap (L_2 \to \bullet) \cap (L_3 \to \bullet) \stackrel{.}{\leq} \alpha \to \bullet\end{aligned}$$

If F is satisfied by a valuation v, then the substitution $\alpha \mapsto \bigcap_{v(x)=1} x \cap \bigcap_{v(x)=0} \neg x$ satisfies C. If C is satisfied by a substitution S, then by Corollary 11 and the consistency constraints we have either $\sigma_{\neg x} \leq S(\alpha) \leq \neg x$ or $\sigma_x \leq S(\alpha) \leq x$ for $x \in V$. A valuation v constructed according to these cases satisfies each clause in F due to Corollary 11 and the validity constraints.

Instead of using constants $\{a_1, \dots, a_k, \bullet\}$ for an instance of the matching problem, encode $[a_i] = \underbrace{\bullet \to \dots \to \bullet \to}_{i \text{ times}} \bullet$ for $1 \leq i \leq k$ in the proof. Using this technique it is easy to see that only one type constant $\bullet$ is sufficient. ◀

Combining Proposition 15 with the reduction in [12] we conclude that neither restricting substitutions to the shape $S : \{\alpha\} \to \mathbb{T}$ nor restricting to the shape $S : \mathbb{V} \to \mathbb{C}$ (atomic substitutions, mapping variables to type constants) reduces the complexity of matching.

4 Intersection type unification

4.1 The unification problem

▶ **Problem 16** (Satisfiability). *Given a set of constraints $C = \{\sigma_1 \stackrel{.}{\leq} \tau_1, \dots, \sigma_n \stackrel{.}{\leq} \tau_n\}$, is there a substitution $S \colon \mathbb{V} \to \mathbb{T}$ such that $S(\sigma_i) \leq S(\tau_i)$ for $1 \leq i \leq n$?*

▶ **Problem 17** (Unification). *Given a set of constraints $C = \{\sigma_1 \doteq \tau_1, \dots, \sigma_n \doteq \tau_n\}$, is there a substitution $S \colon \mathbb{V} \to \mathbb{T}$ such that $S(\sigma_i) = S(\tau_i)$ for $1 \leq i \leq n$?*

Since for any $\sigma, \tau \in \mathbb{T}$ and any substitution S we have $S(\sigma) \leq S(\tau) \iff S(\sigma) \cap S(\tau) = S(\sigma)$, satisfiability and unification are equivalent. Similarly to matching (resp. one-sided unification) restricting satisfiability (resp. unification) to single constraints does not reduce its complexity.

We now provide a number of observations that give some insight into the type-theoretical and combinatorial expressive power of unification.

Consider a combinatory logic with intersection types [15, 11] with arbitrary basis $\mathfrak{B}$, that is, a finite set of combinator symbols $F, G, \dots$ with type schemes $\tau_F, \tau_G, \dots$. Such a system is given by the rules (applicative fragment) $(\to\text{E}), (\cap\text{I}), (\leq)$ of Definition 12 together with a rule assigning types $S(\tau_F)$ to the combinator symbol F for any substitution S. Write $\mathfrak{B} \vdash E : \tau$ for derivability of the type τ for the combinatory expression E in this system.

▶ **Example 18.** Let $\mathfrak{B} = \{F : (\sigma \to \tau) \to \bullet, G : \alpha \to \alpha\}$, where wlog. $\alpha \notin \mathrm{Var}(\sigma) \cup \mathrm{Var}(\tau)$. In this scenario, type-checking $\mathfrak{B} \vdash F\,G : \bullet$ is equivalent to solving the satisfiability problem $\alpha \to \alpha \mathrel{\dot{\leq}} \sigma \to \tau$, equivalently, the unification problem $\sigma \cap \tau \doteq \sigma$, because we need to find substitutions $S, S_1, \dots S_n$ for some $n \in \mathbb{N}$ such that

$$\begin{aligned}
&\bigcap_{i=1}^{n} S_i(\alpha \to \alpha) \leq S(\sigma \to \tau)\\
\overset{\text{Lem. 5}}{\iff} &S(\sigma) \leq \bigcap_{i \in I} S_i(\alpha) \text{ and } \bigcap_{i \in I} S_i(\alpha) \leq S(\tau) \text{ for some } I \subseteq \{1, \dots, n\}\\
\iff &S(\alpha \to \alpha) \leq S(\sigma \to \tau) \text{ setting } S(\alpha) = \bigcap_{i \in I} S_i(\alpha)
\end{aligned}$$

Write $\mathfrak{B} \vdash^* E : \tau$ if $\mathfrak{B} \vdash E : \tau$ is derivable without the intersection introduction rule ($\cap$I). This restriction occupies an interesting ‘intermediate’ position: generalized to arbitrary bases $\mathfrak{B}$, it is the combinatory logic that subsumes the **BCD**-calculus without intersection introduction [18, 20]. For example, $\vdash^*$ is sufficient to type $S\,I\,I$, i.e. the SKI-combinatory logic equivalent of the λ-term $\lambda x.x\,x$ not typable in simple types.

▶ **Example 19.** Typability with respect to $\vdash^*$ is equivalent to unification. Let $\mathfrak{B} = \{F_1 : \tau_1, \dots F_n : \tau_n\}$, where wlog. $\mathrm{Var}(\tau_i) \cap \mathrm{Var}(\tau_j) = \emptyset$ for $i \neq j$, and let E be a combinatory term over $\mathfrak{B}$. We want to know whether there is a type τ such that $\mathfrak{B} \vdash^* E : \tau$.

For any combinatory term E' and type τ' we define

$$f(E', \tau') = \begin{cases} \{\tau_i \mathrel{\dot{\leq}} \tau'\} & \text{if } E' = F_i \text{ for some } i \in \{1, \dots, n\} \\ f(E_1, \alpha \to \beta) \cup f(E_2, \alpha) & \text{if } E' = E_1\,E_2 \text{ and } \alpha, \beta \text{ are fresh} \end{cases}$$

The unification problem instance $f(E, \alpha)$, where α is fresh, has a solution iff E is typable in the basis $\mathfrak{B}$, i.e. there exists a type τ such that $\mathfrak{B} \vdash^* E : \tau$. Conversely, given a satisfiability problem $\sigma \mathrel{\dot{\leq}} \tau$, we consider typability of $F\,G$ in the basis $\mathfrak{B} = \{F : \tau \to a, G : \sigma\}$.

The following example shows that unification can force exponential growth of the size of solutions.

▶ **Example 20.** Consider prime numbers $2, 3$ and the following unification constraints

$$a \to a \to (\beta_2 \cap b) \doteq \beta_2 \cap \alpha, \qquad a \to a \to a \to (\beta_3 \cap b) \doteq \beta_3 \cap \alpha\,.$$

The smallest substitution satisfying the above constraints is

$$\begin{aligned}
S(\beta_2) &= (a \to a \to b) \cap (a \to a \to a \to a \to b)\\
S(\beta_3) &= a \to a \to a \to b\\
S(\alpha) &= a \to a \to a \to a \to a \to a \to b
\end{aligned}$$

In particular, the size of $S(\alpha)$ is greater than the product of our initial primes. By adding an additional constraint $a \to a \to a \to a \to a \to (\beta_5 \cap b) \doteq \beta_5 \cap \alpha$, the size of $S(\alpha)$ becomes at least $2 \cdot 3 \cdot 5$, growing exponentially with additional constraints.

An axiomatization of the equational theory of intersection type subtyping (without ω) is derived in [22]. We add two additional axioms **(U)** and **(RE)** in the following Definition 21 to incorporate ω.

▶ **Definition 21** (ACIUD$_l$ReAb). The equational theory ACIUD$_l$ReAb is given by

(A) $\sigma \cap (\tau \cap \rho) \sim (\sigma \cap \tau) \cap \rho$
(C) $\sigma \cap \tau \sim \tau \cap \sigma$
(I) $\sigma \cap \sigma \sim \sigma$
(U) $\sigma \cap \omega \sim \sigma$,
(D$_l$) $(\sigma \to \tau) \cap (\sigma \to \tau') \sim \sigma \to \tau \cap \tau'$
(RE) $\omega \sim \omega \to \omega$
(AB) $\sigma \to \tau \sim (\sigma \to \tau) \cap (\sigma \cap \sigma' \to \tau)$

The recursion axiom **(RE)** captures the recursive nature of ω and the absorption axiom **(AB)** captures contra-variance.

▶ **Lemma 22.** *Given $\sigma, \tau \in \mathbb{T}$ we have $\sigma = \tau$ iff $\sigma \sim \tau$.*

Proof.

(⇒) Induction on the depth of the derivation of $\sigma \leq \tau$ to show $\sigma \cap \tau \sim \sigma$. Therefore, $\sigma \leq \tau$ and $\tau \leq \sigma$ imply $\sigma \sim \sigma \cap \tau \sim \tau$.

(⇐) Each axiom of ACIUD$_l$ReAb is derivable using subtyping. ◀

The absorption axiom **(AB)** distinguishes the above theory ACIUD$_l$ReAb from theories studied in literature. As discussed in the introduction, the closest equational theory ACIUD$_l$ of [2], which assumes $\omega \to \sigma \sim \omega \sim \sigma \to \omega$ and has no equivalent of the absorption axiom **(AB)**, is Exptime-complete. Unfortunately, the absorption axiom prevents the approaches presented in [2, 1] as shown by the following examples.

▶ **Example 23.** Consider $\alpha \cap (\alpha \to a) \doteq \alpha$ (or equivalently $\alpha \mathrel{\dot{\leq}} \alpha \to a$). A DAG-based (or 'occurs-check'-based) approach cannot stratify such a constraint (even in the absence of ω) since any solution $S(\alpha)$ contains at least one subterm $S(\alpha) \to a$ and therefore a circular dependency. Interestingly, using absorption there is a solution $S(\alpha) = a \cap (a \to a)$.

▶ **Example 24.** Consider $\alpha \cap (((\alpha \to c) \cap b) \to a) \doteq \alpha$ (or equivalently $\alpha \mathrel{\dot{\leq}} ((\alpha \to c) \cap b) \to a)$. In contrast to the previous example, all occurrences of α are positive. Again, we have a circular dependency. Using absorption there is a solution $S(\alpha) = b \to a$.

4.2 Tiling games

In this section we introduce a special kind of domino tiling game, referred to as two-player corridor tiling games, for which Chlebus showed in 1986 that the problem of existence of winning strategies is Exptime-complete [9]. We then show that Exptime-completeness is preserved when tilings are restricted to a particular ("spiral") shape, which will be used to prove our Exptime-lower bound for intersection type unification in Sec. 4.3.

▶ **Definition 25** (Tiling System). A *tiling system* is a tuple $(D, H, V, \bar{b}, \bar{t}, n)$, where:

- D is a finite set of tiles (also called dominoes)
- $H, V \subseteq D \times D$ are horizontal and vertical constraints
- $\bar{b}, \bar{t}$ are n-tuples of tiles
- n is a unary encoded natural number

▶ **Definition 26** (Corridor Tiling). Given a tiling system $(D, H, V, \bar{b}, \bar{t}, n)$, a *corridor tiling* is a mapping $\lambda : \{1, \ldots, l\} \times \{1, \ldots, n\} \to D$ for some $l \in \mathbb{N}$ such that:

- $\bar{b} = (\lambda(1, 1), \ldots, \lambda(1, n))$ (correct bottom row)
- $\bar{t} = (\lambda(l, 1), \ldots, \lambda(l, n))$ (correct top row)

- for $i \in \{1, \dots, l\}$ and $j \in \{1, \dots, n-1\}$ we have $(\lambda(i,j), \lambda(i,j+1)) \in H$, i.e. the horizontal constraints are satisfied
- for $i \in \{1, \dots, l-1\}$ and $j \in \{1, \dots, n\}$ we have $(\lambda(i,j), \lambda(i+1,j)) \in V$, i.e. the vertical constraints are satisfied

Given a tiling system $(D, H, V, \bar{b}, \bar{t}, n)$, a *Two-Player Corridor Tiling* game consists of two players (*Constructor* and *Spoiler*). The game is played on an $\mathbb{N} \times \{1, \dots, n\}$ board and starts with the bottom row $\bar{b}$. Each player places tiles in turn starting with Constructor. While Constructor tries to construct a corridor tiling, Spoiler tries to prevent it. Constructor wins if Spoiler makes an illegal move (with respect to H or V), or when a correct corridor tiling is completed. We say Constructor has winning strategy, if he can win no matter what Spoiler does.

▶ **Lemma 27** (Chlebus [9]). *The decision problem whether Constructor has a winning strategy in a given two-player corridor tiling game is* Exptime*-complete.*

Instead of directly encoding a Two-Player Corridor Tiling into intersection type satisfiability, we introduce a slightly different game that is played out as sequences instead of corridors. The main goal is to get rid of several structural constraints of corridors for a more accessible construction of a spiral where each new tile has a neighboring previous tile.

▶ **Definition 28** (Spiral Tiling). Given a tiling system $(D, H, V, \bar{b}, \bar{t}, n)$, a *spiral tiling* is a sequence $d_1 \dots d_m \in D^m$ for some $m \in \mathbb{N}$ such that:

- $d_1 \dots d_n = \bar{b}$
- $d_{m-n+1} \dots d_m = \bar{t}$
- $(d_i, d_{i+1}) \in H$ for $1 \leq i \leq m-1$
- $(d_i, d_{i+n}) \in V$ for $1 \leq i \leq m-n$

Given a tiling system $(D, H, V, \bar{b}, \bar{t}, n)$ a *Two-Player Spiral Tiling* game, played by Constructor and Spoiler, starts with the sequence $\bar{b}$. Each player adds a tile to the end of the current sequence taking turns starting with Constructor. While Constructor tries to construct a spiral tiling, Spoiler tries to prevent it. Constructor wins if Spoiler makes an illegal move (with respect to H or V), or when a correct spiral tiling is completed. Again, we are interested in whether Constructor has a winning strategy.

The main differences between a corridor tiling and a spiral tiling is the lack of individual rows. While a tile at the beginning of the new row of a corridor is not constrained by the previously placed tile, in a spiral each new tile is constraint by the previously placed one. Additionally, a corridor tiling always contains $l \cdot n$ tiles for some l; a spiral tiling does not obey such a restriction.

▶ **Lemma 29.** *The decision problem whether Constructor has a winning strategy in a given two-player spiral tiling game is* Exptime*-complete.*

Proof.

Lower Bound: Given a tiling system $T = (D, H, V, (b_1, \dots, b_n), (t_1, \dots, t_n), n)$, let:

$$\begin{aligned}
D' &= D \,\dot\cup\, \{\#\} \\
H' &= H \,\dot\cup\, \{(d, \#) \mid d \in D'\} \cup \{(\#, d) \mid d \in D'\} \\
V' &= V \,\dot\cup\, \{(\#, \#)\} \\
T' &= (D', H', V', (b_1, \dots, b_n, \#, \#), (t_1, \dots, t_n, \#, \#), n+2)
\end{aligned}$$

We show that Constructor has a winning strategy for Two-Player Corridor Tiling in T iff he has a winning strategy for Two-Player Spiral Tiling in T'.

By construction, both players are allowed to and have to place the tile # at exactly the turns $i(n+2)-1$ and $i(n+2)$ for $i \geq 1$. Therefore, a winning strategy does not branch nor end at those turns. Additionally, a correct spiral tiling ends in two consecutive # tiles, therefore necessarily contains $i(n+2)$ tiles.

From any correct corridor tiling $\lambda : \{1, \ldots, l\} \times \{1, \ldots, n\}$ for T we construct a spiral tiling $d_1 \ldots d_{l(n+2)}$ for T' by

$$d_k = \begin{cases} \lambda(i,j) & \text{if } k = (i-1)(n+2)+j \text{ and } i \geq 1 \text{ and } 1 \leq j \leq n \\ \# & \text{if } k = (i-1)(n+2)+j \text{ and } i \geq 1 \text{ and either } j = 0 \text{ or } j = n+1 \end{cases}$$

From a correct spiral tiling $d_1 \ldots d_{l(n+2)}$ for T' we construct a corridor tiling $\lambda : \{1, \ldots, l\} \times \{1, \ldots, n\}$ for T by $\lambda(i,j) = d_{(i-1)(n+2)+j}$.

In particular, Constructor's winning strategy (skipping/adding the forced # turns) is exactly the same for both games.

Upper Bound: Computation in APSPACE = EXPTIME (similar to Two-Player Corridor Tiling). To continue the game only the n previously placed tiles have to be considered. ◀

4.3 Exptime lower bound

We now prove our main result, that the intersection type unification problem is EXPTIME-hard. The proof will be by reduction from spiral tiling games (Lemma 29) to the intersection type satisfiability problem.

Let $T = (D, H, V, \bar{b} = b_1 \ldots b_n, \bar{t} = t_1 \ldots t_n, n)$ be a tiling system. Wlog. $(b_i, b_{i+1}) \in H$ and $(t_i, t_{i+1}) \in H$ for $1 \leq 1 < n$. We fix the set of type constants $\mathbb{C} = D \dot{\cup} \{\bullet\}$ and variables $\mathbb{V} = \{\alpha\} \cup \{\beta_d \mid d \in D\}$ and construct the following set of constraints $\mathcal{C}_T$:

(i) $\sigma_\perp^H \cap \sigma_\perp^V \cap \sigma_t \cap \bigcap_{d \in D} \beta_d \mathrel{\dot{\leq}} \sigma_b \cap \bigcap_{d' \in D} \bigcap_{d \in D} (d' \to d \to \beta_d)$ (Game moves)

(ii) $\bigcap_{(d',d) \in H} (d \to d' \to \alpha) \mathrel{\dot{\leq}} \bigcap_{d \in D} (d \to \beta_d)$ (d respects H)

(iii) $\bigcap_{(d',d) \in V} (d \to \underbrace{\omega \to \ldots \to \omega \to}_{n-1 \text{ times}} d' \to \alpha) \mathrel{\dot{\leq}} \bigcap_{d \in D} (d \to \beta_d)$ (d respects V)

where

$$\sigma_b \equiv b_n \to \ldots \to b_1 \to \bullet \qquad \text{(Initial state)}$$

$$\sigma_t \equiv (t_n \to \ldots \to t_1 \to \alpha) \cap (\omega \to t_n \to \ldots \to t_1 \to \alpha) \qquad \text{(Final states)}$$

$$\sigma_\perp^H \equiv \bigcap_{(d,d') \in D \times D \setminus H} (d' \to d \to \alpha) \qquad \text{(d' violates H)}$$

$$\sigma_\perp^V \equiv \bigcap_{(d,d') \in D \times D \setminus V} (d' \to \underbrace{\omega \to \ldots \to \omega \to}_{n-1 \text{ times}} d \to \alpha) \qquad \text{(d' violates V)}$$

Intuitively, we want to use Lemma 10 to realize alternation. The rhs of (i) represents an intersection of all board positions which Constructor may face. Therefore, for all such position he needs to find a suitable move by picking a path on the lhs of (i). He can either state that the Spoilers last move violates H or V choosing $\sigma_\perp^H$ or $\sigma_\perp^V$ or state that the game is finished choosing σ_t or pick his next move $d \in D$ choosing β_d. Intuitively, β_d captures all board positions in which Constructors decides to place d next. Note that on the rhs of (i) in the type $d' \to d \to \beta_d$ the tile d' is not constrained (representing all possible moves of Spoiler) while the tile d is constrained to the index of β_d, i.e. Constructors previous choice.

Therefore, by picking a move d Constructor is faced with all board positions that arise from the previous position extended by d and each possible d'. Constraints (ii) and (iii) ensure that whenever Constructor picks his next move $d \in D$ choosing β_d he has to respect H and V.

We show that Constructor has a winning strategy for two-player spiral tiling in T iff the constraint system $\mathcal{C}_T$ is satisfiable. To represent game positions as types, we define the mapping $[\cdot] : D^* \to \mathbb{T}$ such that $[\epsilon] = \bullet$ and $[\bar{s}d] = d \to [\bar{s}]$ for $\bar{s} \in D^*$ and $d \in D$. To improve readability, we use the notation $\sigma \overset{\phi}{\leq} \tau$, where ϕ is a hint why the inequality holds.

▶ **Lemma 30.** *Let T be a tiling system. If Constructor has a winning strategy in a two-player spiral tiling game in T, then the constraint system $\mathcal{C}_T$ is satisfiable.*

Proof. Assume that Constructor has a winning strategy that is represented by a labeled tree $f : \mathrm{dom}(f) \to \{C, S\}$ where

- $\mathrm{dom}(f) \subseteq D^*$ is finite and prefix-closed, i.e. $\bar{u}\bar{v} \in \mathrm{dom}(f)$ implies $\bar{u} \in \mathrm{dom}(f)$.
- $\mathrm{depth}(f) = \max\{k \mid d_1 \dots d_k \in \mathrm{dom}(f)\}$.
- For $\bar{s} = d_1 \dots d_k \in \mathrm{dom}(f)$ we have $f(\bar{s}) = C$ if k is even and $f(\bar{s}) = S$ if k is odd, i.e. C places a tile after an even number of turns and S after an odd number of turns.
- For $\bar{s} \in \mathrm{dom}(f)$ such that $f(\bar{s}) = S$ we have $\bar{s}d' \in \mathrm{dom}(f)$ for all $d' \in D$, i.e. the strategy has to consider all (possibly illegal) Spoilers moves.
- For $\bar{s} \in \mathrm{dom}(f)$ such that $f(\bar{s}) = C$ we have either
  - There exists exactly one $d \in D$ such that $\bar{s}d \in \mathrm{dom}(f)$ and $\bar{b}\bar{s} = \bar{u}d_1 \dots d_n$ for some $\bar{u} \in D^*$ and $d_1, \dots, d_n \in D$ with $(d_n, d) \in H$ and $(d_1, d) \in V$, i.e. Constructors next move is d which respects H and V.
  - $\bar{s}d \notin \mathrm{dom}(f)$ for all $d \in D$ and either
    - $\bar{b}\bar{s} = \bar{u}\bar{t}$ for some $\bar{u} \in D^*$, i.e. Constructor states that the game is finished.
    - $\bar{b}\bar{s} = \bar{u}\bar{t}d'$ for some $\bar{u} \in D^*$ and $d' \in D$, i.e. Constructor states that Spoilers last move d' is illegal because the game already ended.
    - $\bar{b}\bar{s} = \bar{u}dd'$ for some $\bar{u} \in D^*$, $d, d' \in D$ such that $(d, d') \notin H$, i.e. Constructor states that Spoilers last move d' violates H.
    - $\bar{b}\bar{s} = \bar{u}d\bar{v}d'$ for some $\bar{u} \in D^*$, $\bar{v} \in D^{n-1}$, $d, d' \in D$ such that $(d, d') \notin V$, i.e. Constructor states that Spoilers last move d' violates V.

We construct the following substitution S

$$S(\alpha) = \bigcap_{\substack{d_1 \dots d_k = \bar{s} \in D^* \\ k \leq \mathrm{depth}(f)+n}} [\bar{s}] \quad \text{and} \quad S(\beta_d) = \bigcap_{\substack{\bar{s} \in f^{-1}(C) \\ \bar{s}d \in \mathrm{dom}(f)}} [\bar{b}\bar{s}] \quad \text{for } d \in D\,.$$

We verify that the individual inequalities hold.

- $S(\sigma_\perp^H \cap \sigma_\perp^V \cap \sigma_t \cap \bigcap_{d \in D} \beta_d) \leq \sigma_b$:
  if $\bar{b} = \bar{t}$, then $S(\sigma_t) \leq \sigma_b$. Otherwise, according to f, there exists a $d \in D$ such that $d \in \mathrm{dom}(f)$. Therefore, $S(\beta_d) \leq [\bar{b}] \equiv \sigma_b$.
- $S(\sigma_\perp^H \cap \sigma_\perp^V \cap \sigma_t \cap \bigcap_{d \in D} \beta_d) \leq S(d' \to d \to \beta_d)$ for all $d, d' \in D$:
  we show that for any $\pi = d_k \to \dots \to d_1 \to \sigma_b = [\bar{b}\bar{s}]$ such that $d_1 \dots d_k = \bar{s} \in f^{-1}(C)$ and $\bar{s}d \in \mathrm{dom}(f)$ we have $S(\sigma_\perp^H \cap \sigma_\perp^V \cap \sigma_t \cap \bigcap_{d \in D} \beta_d) \leq d' \to d \to \pi \equiv [\bar{b}\bar{s}dd']$. Since $\bar{s}d \in \mathrm{dom}(f)$ and $f(\bar{s}) = C$ we have $\bar{s}dd' \in \mathrm{dom}(f)$ and $f(\bar{s}dd') = C$. According to f we have either
  - $\bar{s}dd'd'' \in \mathrm{dom}(f)$ for some $d'' \in D$. Therefore, $S(\beta_{d''}) \leq d' \to d \to \pi \equiv [\bar{b}\bar{s}dd']$
  - or

  - $\bar{b}\bar{s}dd' = \bar{u}\bar{t}$ for some $\bar{u} \in D^*$, then $S(\sigma_t) \leq [\bar{u}\bar{t}] \equiv [\bar{b}\bar{s}dd']$.
  - $\bar{b}\bar{s}dd' = \bar{u}\bar{t}d'$ for some $\bar{u} \in D^*$, then $S(\sigma_t) \leq [\bar{u}\bar{t}d'] \equiv [\bar{b}\bar{s}dd']$.
  - $(d, d') \notin H$, then $S(\sigma_\perp^H) \leq [\bar{b}\bar{s}dd']$.
  - $\bar{b}\bar{s}dd' = \bar{u}d''\bar{v}d'$ for some $\bar{u} \in D^*$, $\bar{v} \in D^{n-1}$, $d'' \in D$ such that $(d'', d') \notin V$, then $S(\sigma_\perp^V) \leq [\bar{u}d''\bar{v}d'] \equiv [\bar{b}\bar{s}dd']$.

- $S(\bigcap_{(d',d)\in H} (d \to d' \to \alpha)) \leq S(d \to \beta_d)$ for all $d \in D$:
  fix any $\pi = d_k \to \ldots \to d_1 \to \sigma_b = [\bar{b}\bar{s}]$ such that $d_1 \ldots d_k = \bar{s} \in f^{-1}(C)$ and $\bar{s}d \in \mathrm{dom}(f)$. Since $f(\bar{s}) = C$ and $\bar{s}d \in \mathrm{dom}(f)$, we have $\bar{b}\bar{s}d = \bar{u}d'd$ for some $\bar{u} \in D^*$ and $d' \in D$ such that $(d', d) \in H$. We have:

$$S(\bigcap_{(d',d)\in H} (d \to d' \to \alpha)) \overset{S(\alpha)\leq[\bar{u}]}{\leq} [\bar{u}d'd] \equiv [\bar{b}\bar{s}d] \equiv d \to \pi\,.$$

- $S(\bigcap_{(d',d)\in V} (d \to \underbrace{\omega \to \ldots \to \omega \to}_{n-1 \text{ times}} d' \to \alpha)) \leq S(d \to \beta_d)$ for all $d \in D$:
  fix any $\pi = d_k \to \ldots \to d_1 \to \sigma_b = [\bar{b}\bar{s}]$ such that $d_1 \ldots d_k = \bar{s} \in f^{-1}(C)$ and $\bar{s}d \in \mathrm{dom}(f)$. Since $f(\bar{s}) = C$ and $\bar{s}d \in \mathrm{dom}(f)$, we have $\bar{b}\bar{s}d = \bar{u}d'\bar{v}d$ for some $\bar{u} \in D^*$, $\bar{v} \in D^{n-1}$ and $d' \in D$ such that $(d', d) \in V$. We have

$$S(\bigcap_{(d',d)\in V} (d \to \underbrace{\omega \to \ldots \to \omega \to}_{n-1 \text{ times}} d' \to \alpha)) \overset{S(\alpha)\leq[\bar{u}]}{\leq} [\bar{u}d'\bar{v}d] \equiv [\bar{b}\bar{s}d] \equiv d \to \pi\,. \qquad ◀$$

▶ **Lemma 31.** *Let T be a tiling system. If the constraint system $\mathcal{C}_T$ is satisfiable, then Constructor has a winning strategy in a two-player spiral tiling game in T.*

Proof. Assume that there exists a substitution S that satisfies the constraints $\mathcal{C}_T$ and wlog. uses only organized types. Constructor wins the game regardless of Spoilers moves as follows:

Let $\tau \equiv S(\sigma_b \cap \bigcap_{d'\in D} \bigcap_{d\in D} (d' \to d \to \beta_d))$, i.e. the rhs of (i). The initial game position is $\bar{b}$. Note that $\tau \leq \sigma_b \equiv [\bar{b}]$. We consider a single turn of Constructor from any game position $\bar{s} \in D^*$ that he may face.

Assume ($\star$) that the current game position $\bar{s}$ satisfies $\tau \leq [\bar{s}]$. Due to (i) and Corollary 11 we have the following cases

- If $S(\sigma_\perp^H) \leq [\bar{s}]$, then there exist $d, d' \in D$ such that $(d, d') \notin H$ and for some path π we have $d' \to d \to \pi \leq [\bar{s}]$. Therefore, $\bar{s} = \bar{u}dd'$ for some $\bar{u} \in D^*$ and Constructor wins because Spoilers last move d' violates H. Note that this is not possible for $\bar{s} = \bar{b}$ since $\bar{b}$ is horizontally consistent.
- If $S(\sigma_\perp^V) \leq [\bar{s}]$, then there exist $d, d' \in D$ such that $(d, d') \notin V$ and for some path π we have $d' \to \underbrace{\omega \to \ldots \to \omega \to}_{n-1 \text{ times}} d \to \pi \leq [\bar{s}]$. Therefore, $\bar{s} = \bar{u}d\bar{v}d'$ for some $\bar{u} \in D^*$, $\bar{v} \in D^{n-1}$ and Constructor wins because Spoilers last move d' violates V. Note that this is not possible for $\bar{s} = \bar{b}$ since $\bar{b}$ is too short.
- If $S(\sigma_t) \leq [\bar{s}]$, then Constructor wins because $t_n \to \ldots \to t_1 \to \pi \leq [\bar{s}]$ for some path π implies the winning condition $\bar{s} = \bar{u}\bar{t}$ for some $\bar{u} \in D^*$. Alternatively $\omega \to t_n \to \ldots \to t_1 \to \pi \leq [\bar{s}]$ for some path π implies $\bar{s} = \bar{u}\bar{t}d'$ for some $\bar{u} \in D^*$ and $d' \in D$, therefore Spoilers last move d' was illegal because the game already ended.
- If $S(\beta_d) \leq [\bar{s}]$ for some $d \in D$, then Constructor may safely place d as the next tile. We verify consistency wrt. H and V. First, due to (ii) there exists a path $d \to d' \to \pi$ for

some $d' \in D$ with $(d', d) \in H$ such that

$$d \to d' \to \pi \overset{(ii)}{\leq} S(d \to \beta_d) \overset{S(\beta_d) \leq [\bar{s}]}{\leq} [\bar{s}d]\,.$$

Therefore, $\bar{s}d = \bar{u}d'd$ for some $\bar{u} \in D^*$ and placing d does not violate H. Second, due to (iii) there exists a path $d \to \underbrace{\omega \to \ldots \to \omega \to}_{n-1 \text{ times}} d' \to \pi'$ for some $d' \in D$ with $(d', d) \in V$ such that

$$d \to \underbrace{\omega \to \ldots \to \omega \to}_{n-1 \text{ times}} d' \to \pi' \overset{(iii)}{\leq} S(d \to \beta_d) \overset{S(\beta_d) \leq [\bar{s}]}{\leq} [\bar{s}d]\,.$$

Therefore, $\bar{s}d = \bar{u}d'\bar{v}d$ for some $\bar{u} \in D^*$, $\bar{v} \in D^{n-1}$ and placing d does not violate V. Note that in neither case Constructor loses. If Constructor placed the tile $d \in D$, in which case we have $S(\beta_d) \leq [\bar{s}]$, Spoiler may place any tile $d' \in D$. The new game position is $\bar{s}dd'$. Note that our initial assumption $(\star)$ is inductively satisfied

$$\tau \leq S(d' \to d \to \beta_d) \overset{S(\beta_d) \leq [\bar{s}]}{\leq} [\bar{s}dd']\,.$$

Therefore, we may apply our argument inductively. Additionally, the game necessarily ends after a finite number of turns: if $\tau = \bigcap_{i \in I} (\sigma_1^i \to \ldots \sigma_{l_i}^i \to c_i)$ (for some index set I, integers $l_i \geq 0$ for $i \in I$, types σ_j^i for $i \in I$ and $1 \leq j \leq l_i$ and type constants c_i for $i \in I$), then $(\star)$, i.e. $\tau \leq [\bar{s}]$, cannot be satisfied by any $\bar{s} \in D^k$ with $k > \max\{l_i \mid i \in I\}$. ◀

▶ **Theorem 32.** *The intersection type satisfiability problem and the intersection type unification problem are* Exptime*-hard.*

Proof. Immediate from Lemma 29, Lemma 30 and Lemma 31, since all reduction steps are evidently computable in polynomial time. Moreover, satisfiability is polynomial time equivalent to unification. ◀

▶ **Corollary 33.** *Satisfiability and unification are* Exptime*-hard even in the presence of only one constant.*

Proof. Instead of using constants $\{d_1, \ldots, d_k, \bullet\}$, encode $[d_i] = \underbrace{\bullet \to \ldots \to \bullet \to}_{i \text{ times}} \bullet$ for $1 \leq i \leq k$ in the original proofs. Therefore, only one type constant $\bullet$ is sufficient. ◀

Note that without any constants satisfiability and unification are trivial by mapping all type variables to ω.

▶ **Corollary 34.** *Satisfiability and unification are* Exptime*-hard even if the codomain of substitutions is restricted to types of rank 1, i.e. intersections of simple types.*

Proof. In the proof of Lemma 30 each variable is substituted by an intersection of simple types, i.e. a rank 1 intersection type. ◀

Interestingly, the axiom **(RE)** $\omega \sim \omega \to \omega$ (resp. $\omega \leq \omega \to \omega$) is not necessary for the Exptime lower bound proof, while the axioms **(U)** and **(AB)** (resp. $\sigma \leq \omega$ and $\omega \to \tau \leq \sigma \to \tau$ derived from contravariance) play a crucial role in the construction of $\sigma_\perp^V$ to capture an exponential number of cases.

5 Conclusion and future work

We have positioned the intersection type unification problem as a natural object of study within unification theory and type theory, and we have provided the first nontrivial lower bound showing that the problem is of high complexity. Our EXPTIME-lower bound uses game-theoretic methods which may be useful for making further progress on the main open question for future work, that of decidability. Next steps include exploring variants and restrictions. Variants of intersection type subtyping theories (see [5]) give rise to a *family* of intersection type unification problems yet to be studied, e.g., the ω-free theory. We conjecture an NEXPTIME-upper bound for rank 1 restricted unification, in which variables are substituted by intersections of simple types. Since organized rank 1 subtyping corresponds to set inclusion, one can reduce a rank 1 unification problem to satisfiability of set constraints with projections [8] in finite sets. Unfortunately, standard set constraint interpretations may contain infinite sets, which is why this approach needs further investigation.

Acknowledgements. We would like to thank Boris Düdder (Dortmund), Paweł Urzyczyn, Aleksy Schubert, and Marcin Benke (Warsaw) and Mariangiola Dezani and the Torino λ-calculus group for helpful discussions. Thanks are also due to our reviewers for useful comments.

References

1 S. Anantharaman, P. Narendran, and M. Rusinowitch. Acid-unification is NEXPTIME-decidable. In *Mathematical Foundations of Computer Science 2003*, pages 169–178. Springer, 2003.

2 S. Anantharaman, P. Narendran, and M. Rusinowitch. Unification Modulo ACUI Plus Distributivity Axioms. *Journal of Automated Reasoning*, 33(1):1–28, 2004.

3 F. Baader and P. Narendran. Unification of Concept Terms in Description Logics. *Journal of Symbolic Computation*, 31:277–305, 2001.

4 H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A Filter Lambda Model and the Completeness of Type Assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983. doi:10.2307/2273659.

5 H. Barendregt, W. Dekkers, and R. Statman. *Lambda Calculus with Types*. Perspectives in Logic, Cambridge University Press, 2013.

6 G. Boudol and P. Zimmer. On type inference in the intersection type discipline. *Electr. Notes Theor. Comput. Sci.*, 136:23–42, 2005. doi:10.1016/j.entcs.2005.06.016.

7 G. Castagna, K. Nguyen, Z. Xu, and P. Abate. Polymorphic functions with set-theoretic types: Part 2: Local type inference and type reconstruction. In *Principles of Programming Languages POPL 2015*, pages 289–302. ACM, 2015.

8 W. Charatonik and L. Pacholski. Set constraints with projections are in NEXPTIME. In *35th Annual Symposium on Foundations of Computer Science*, pages 642–653. IEEE, 1994. doi:10.1109/SFCS.1994.365727.

9 B. S. Chlebus. Domino-tiling games. *Journal of Computer and System Sciences*, 32(3):374–392, 1986.

10 M. Coppo and P. Giannini. Principal types and unification for a simple intersection type system. *Inf. Comput.*, 122(1):70–96, 1995. doi:10.1006/inco.1995.1141.

11 M. Dezani-Ciancaglini and J. R. Hindley. Intersection Types for Combinatory Logic. *Theoretical Computer Science*, 100(2):303–324, 1992.

12 B. Düdder, M. Martens, and J. Rehof. Intersection Type Matching with Subtyping. In *Proceedings of TLCA'13*, Springer LNCS, 2013.

13 B. Düdder, M. Martens, J. Rehof, and P. Urzyczyn. Bounded Combinatory Logic. In *Proceedings of CSL'12*, volume 16 of *LIPIcs*, pages 243–258, 2012.

14 J. R. Hindley. The Simple Semantics for Coppo-Dezani-Sallé Types. In *International Symposium on Programming*, volume 137 of *LNCS*, pages 212–226. Springer, 1982.

15 J. R. Hindley and J. P. Seldin. *Lambda-calculus and Combinators, an Introduction.* Cambridge University Press, 2008.

16 A. J. Kfoury. Beta-reduction as unification. *Banach Center Publications*, 46(1):137–158, 1999.

17 A. J. Kfoury and J. B. Wells. Principality and Type Inference for Intersection Types Using Expansion Variables. *Theor. Comput. Sci.*, 311(1-3):1–70, 2004.

18 T. Kurata and M. Takahashi. Decidable properties of intersection type systems. In *TLCA*, volume 902 of *LNCS*, pages 297–311. Springer, 1995.

19 J. Rehof and P. Urzyczyn. Finite Combinatory Logic with Intersection Types. In *Proceedings of TLCA'11*, volume 6690 of *LNCS*, pages 169–183. Springer, 2011.

20 J. Rehof and P. Urzyczyn. The Complexity of Inhabitation with Explicit Intersection. In *Kozen Festschrift*, volume 7230 of *LNCS*, pages 256–270. Springer, 2012.

21 S. Ronchi Della Rocca. Principal Type Scheme and Unification for Intersection Type Discipline. *Theor. Comput. Sci.*, 59:181–209, 1988.

22 R. Statman. A finite model property for intersection types. In *Proceedings Seventh Workshop on Intersection Types and Related Systems, ITRS 2014, Vienna, Austria, 18 July 2014.*, pages 1–9, 2015. doi:10.4204/EPTCS.177.1.

Computing Connected Proof(-Structure)s From Their Taylor Expansion

Giulio Guerrieri[1], Luc Pellissier[2], and Lorenzo Tortora de Falco[3]

1 **Dipartimento di Matematica e Fisica, Università Roma Tre, Rome, Italy; and Aix Marseille Université, CNRS, Centrale Marseille, Marseille, France**
`gguerrieri@uniroma3.it`

2 **LIPN, UMR 7030, Université Paris 13, Sorbonne Paris Cité, Villetaneuse, France**
`luc.pellissier@lipn.univ-paris13.fr`

3 **Dipartimento di Matematica e Fisica, Università Roma Tre, Rome, Italy**
`tortora@uniroma3.it`

Abstract

We show that every connected Multiplicative Exponential Linear Logic (MELL) proof-structure (with or without cuts) is uniquely determined by a well-chosen element of its Taylor expansion: the one obtained by taking two copies of the content of each box. As a consequence, the relational model is injective with respect to connected MELL proof-structures.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases proof-nets, (differential) linear logic, relational model, Taylor expansion

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.20

1 Introduction

Given a syntax $\mathcal{S}$ endowed with some rewrite rules, and given a denotational model $\mathcal{D}$ for $\mathcal{S}$ (*i.e.* a semantics which associates with every term t of $\mathcal{S}$ an interpretation $[\![t]\!]_{\mathcal{D}}$ that is invariant under the rewrite rules), we say that $\mathcal{D}$ is *injective* with respect to $\mathcal{S}$ if, for any two normal terms t and t' of $\mathcal{S}$, $[\![t]\!]_{\mathcal{D}} = [\![t']\!]_{\mathcal{D}}$ implies $t = t'$. In categorical terms, injectivity corresponds to faithfulness of the interpretation functor from $\mathcal{S}$ to $\mathcal{D}$. Injectivity is a natural and well studied question for denotational models of λ-calculi and term rewriting systems (see [10, 18]). In the framework of Linear Logic (LL, [11]) this question, addressed in [19], turned out to be remarkably complex: contrary to what happens in the λ-calculus, there exist semantics of LL that are not injective, such as the coherent model which is injective only with respect to some fragments of LL (see [19]). After the first partial positive results obtained in [19], it took a long time to obtain some improvements: in [5], the injectivity of the relational model is proven for MELL (the multiplicative-exponential fragment of LL, sufficiently expressive to encode the λ-calculus) proof-structures that are connected, and eventually in [3] the first complete positive result is achieved, since the author proves that the relational model is injective for all MELL proof-structures.

Ehrhard [6] introduced finiteness spaces, a denotational model of LL (and λ-calculus) which interprets formulas by topological vector spaces and proofs by analytical functions: in this model the operations of differentiation and Taylor expansion make sense. Ehrhard and Regnier [7, 8, 9] internalized these operations in the syntax and thus introduced differential linear logic DiLL_0 (which encodes the resource λ-calculus, see [8]), where the promotion rule (the only one in LL which is responsible for introducing the !-modality and hence for creating

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 20; pp. 20:1–20:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

resources available at will, marked by boxes in LL proof-structures) is replaced by three new "finitary" rules introducing the !-modality which are perfectly symmetric to the rules for the ?-modality: this allows a more subtle analysis of the resources consumption during the cut-elimination process. At the syntactic level, the Taylor expansion decomposes a LL proof-structure/λ-term in a (generally infinite) formal sum of DiLL_0 proof-structures/resource λ-terms, each of which contains resources usable only a fixed number of times. Roughly speaking, each element of the Taylor expansion $\mathcal{T}_R$ of a LL proof-structure/λ-term R is a DiLL_0 proof-structure/resource λ-term obtained from R by replacing each box/argument B in R with n_B copies of its content (for some $n_B \in \mathbb{N}$), recursively.

In the light of the differential approach, it is clear (and well-known) that the resource λ-term of order 1 in the Taylor expansion of a λ-term (which is obtained by taking exactly one copy of the argument of each application) is enough to entirely determine the λ-term: if two λ-terms t_1 and t_2 have the same element of order 1 in their Taylor expansion, then $t_1 = t_2$. One can formulate the results of [5] and [3] by saying that, given two LL proof-structures R_1 and R_2, if there exists an appropriate DiLL_0 proof-structure, whose order *depends on* R_1 *and* R_2, which occurs in the Taylor expansions of both R_1 and R_2, then $R_1 = R_2$. We prove, in the present paper, for connected MELL, a result which is very much in the style of the one just mentioned for the λ-calculus: if two connected MELL proof-structures R_1 and R_2 (with or without cuts) have the same element of order 2 in their Taylor expansions (which is obtained by taking exactly two copies of the content of each box), then $R_1 = R_2$ (*i.e.* the element of order 2 of the Taylor expansion of a connected MELL proof-structure is enough to entirely determine the proof-structure). Since it is known (see [12] for details) that the elements of the Taylor expansion of a LL proof-structure/λ-term are essentially the elements of its interpretation in the relational model, we immediately obtain another proof of the injectivity of the relational model for connected MELL proof-structures.

It is widely acknowledged, in the LL community, that the subsystem of LL corresponding to the λ-calculus enjoys all the possible good properties, while many of them are lost in the general MELL fragment. Our result seems to suggest the following hierarchy:

1. full MELL, for which there does not seem to be a way to bound "a priori" the complexity of the element of the Taylor expansion allowing to distinguish two different proof-structures;
2. connected MELL (containing the λ-calculus) for which the element of order 2 of the Taylor expansion of a proof-structure is enough to entirely determine the proof-structure;
3. the λ-calculus, for which the element of order 1 of the Taylor expansion of a λ-term is enough to entirely determine the λ-term.

Outline. After laying out precise definitions of proof-structure (§2) and Taylor expansion (§3), in §4 we show how a connected MELL proof-structure can be univocally computed by the point of order 2 of its Taylor expansion. Finally, in §5 we infer from this the injectivity of the relational model for connected MELL.

▶ **Notation.** We set $\mathcal{L}_{\mathsf{MELL}} = \{1, \bot, \otimes, \parr, !, ?, \mathit{ax}, \mathit{cut}\}$. The set $\mathcal{F}_{\mathsf{MELL}}$ of MELL *formulas* is generated by the grammar: $A, B, C ::= X \mid X^\bot \mid 1 \mid \bot \mid A \otimes B \mid A \parr B \mid !A \mid ?A$, where X ranges over an infinite set of propositional variables. The linear negation is involutive, *i.e.* $A^{\bot\bot} = A$, and defined via De Morgan laws $1^\bot = \bot$, $(A \otimes B)^\bot = A^\bot \parr B^\bot$ and $(!A)^\bot = ?A^\bot$.

Let $\mathcal{A}$ be a set: $\mathscr{P}(\mathcal{A})$ is the power set of $\mathcal{A}$, $\bigcup\mathcal{A}$ is the union of $\mathcal{A}$, $\mathcal{A}^*$ is the set of finite sequences over $\mathcal{A}$. If $\mathcal{A}$ is ordered by $\leq$, for any $a \in \mathcal{A}$ we set $\downarrow_{\mathcal{A}} a = \{b \in \mathcal{A} \mid b \leq a\}$. The empty sequence is denoted by $(\,)$. Given a finite sequence $a = (a_1, \dots, a_n)$ with $n \in \mathbb{N}$, we set $|a| = n$ and, if $n > 0$, $a^- = (a_1, \dots, a_{n-1})$; if moreover $b = (b_1, \dots, b_m)$, we set

$a \cdot b = (a_1, \ldots, a_n, b_1, \ldots, b_m)$; if $n = 1$ (resp. $m = 1$), then $a_1 \cdot b$ (resp. $a \cdot b_1$) stands for $a \cdot b$. We write $a \sqsubseteq b$ if $a \cdot c = b$ for some finite sequence c. Let $\mathsf{f} \colon \mathcal{A} \to \mathcal{B}$ be a partial function (without "partial", a function is always total): $\mathsf{dom}(\mathsf{f})$ and $\mathsf{im}(\mathsf{f})$ are the domain and image of f; the partial function $\bar{\mathsf{f}} \colon \mathscr{P}(\mathcal{A}) \to \mathscr{P}(\mathcal{B})$ is defined by $\bar{\mathsf{f}}(\mathcal{A}') = \{\mathsf{f}(a) \mid a \in \mathcal{A}' \cap \mathsf{dom}(\mathsf{f})\}$ for any $\mathcal{A}' \subseteq \mathcal{A}$.

2 A non-inductive syntax for proof structures

It is well-known that for LL proof-nets there is no "canonical" representation: every paper about them introduces its own syntax for proof-nets, and more generally for proof-structures, depending on the purposes of the paper.[1] The first aim of the syntax for proof-structures that we present here is to give a rigorous and compact definition of the following notions: (1) equality between proof-structures; (2) Taylor expansion of a proof-structure. The first point naturally leads us to adopt a low-level syntax with generalized ?- and !-links, similarly to [5]. This choice can be made compatible with the second point by giving a completely non-inductive definition of proof-structures, which is in keeping with the intuition that a proof-structure is a directed graph, plus further information about the borders of boxes. We have also taken care of minimizing the information required to identify a proof-structure, especially the borders of its boxes.

We use terminology of interactions nets [13, 8], even if properly speaking our objects are not interaction nets. So, for instance, our cells correspond to links in [2, 14, 19]. Our syntax is inspired by [15, 16, 17, 20, 4, 5]. The main technical novelties with respect to them are that:

- there are no wires (the same port may be auxiliary for some cell and principal for another cell), so axioms and cuts are cells, and our ports corresponds to edges in [2, 14, 19];
- boxes do not have an explicit constructor or cell, hence boxes and depth of a proof-structure are recovered in a non-inductive way.

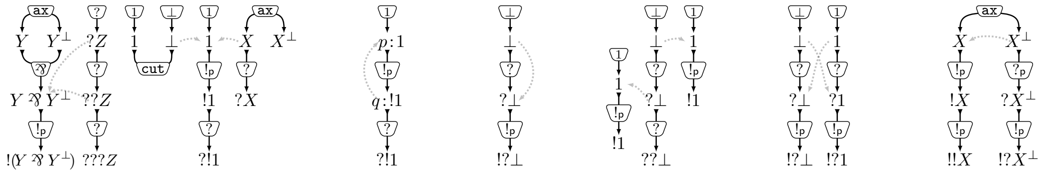
As in [15, 16, 17] and unlike [4, 5], our syntactic objects are typed by MELL formulas: we have opted for a typed version only to keep out immediately the possibility of "vicious cycles" (see Fact 3). All the results in this paper can be adapted also to the untyped case.

Pre-proof-structures and isomorphisms. We define here our basic syntactical object: *pre-proof-structure* (*pps* for short). All other syntactical objects, in particular proof-structures corresponding to the fragments or extensions of LL that we will consider (DiLL-, MELL- and DiLL$_0$-proof structures), are some special cases of pps. Essentially, a pps Φ is a directed labeled graph G_Φ called the *ground-structure* (*gs* for short) of Φ, plus a partial function box_Φ defined on certain edges (or nodes). The gs of Φ represents a "linearised" proof-structure, *i.e.* Φ without the border of its boxes; the partial function box_Φ marks the borders of the boxes of Φ. Examples of pps are in Fig. 1. Unlike [17, 5], our syntactical objects are not necessarily cut-free (nor with atomic axioms). Cut-elimination is not defined since it is not used here.

▶ **Definition 1** (Pre-proof-structure, ports, cells, ground-structure, fatness)**.** A *pre-proof-structure* (*pps* for short) is a 9-tuple $\Phi = (\mathcal{P}_\Phi, \mathcal{C}_\Phi, \mathsf{tc}_\Phi, \mathsf{P}^{\mathsf{pri}}_\Phi, \mathsf{P}^{\mathsf{aux}}_\Phi, \mathsf{P}^{\mathsf{left}}_\Phi, \mathsf{tp}_\Phi, \mathcal{C}^{\mathsf{box}}_\Phi, \mathsf{box}_\Phi)$ such that:

- $\mathcal{P}_\Phi$ and $\mathcal{C}_\Phi$ are finite sets, their elements are resp. the *ports* and the *cells* (or *links*) *of* Φ;

[1] Following [11], a proof-net is a proof-structure sequentializable (*i.e.* corresponding to a derivation) in LL sequent calculus: proof-nets can be (partly or completely, depending on the fragment of LL) characterized among proof-structures via "geometric" correctness criteria, see for instance [1, 19].



(a) A pps Φ.

(b) Two pps, Ψ_1 (on the left) and Ψ_2 (on the right).

(c) A pps X.

(d) Two pps, Ξ (on the left) and Ξ' (on the right).

Figure 1 Some examples of pps that are not DiLL-ps. See Def. 1 and 8.

- tc_Φ is a function from $\mathcal{C}_\Phi$ to $\mathcal{L}_{\mathsf{MELL}}$; for every $l \in \mathcal{C}_\Phi$, $\mathsf{tc}_\Phi(l)$ is the *label*, or *type*, *of* l; for every $t, t' \in \mathcal{L}_{\mathsf{MELL}}$, we set $\mathcal{C}_\Phi^t = \{l \in \mathcal{C}_\Phi \mid \mathsf{tc}_\Phi(l) = t\}$ (whose elements are the t-*cells*, or t-*links*, *of* Φ) and $\mathcal{C}_\Phi^{t,t'} = \mathcal{C}_\Phi^t \cup \mathcal{C}_\Phi^{t'}$;
- $\mathsf{P}_\Phi^{\mathsf{pri}} \colon \mathcal{C}_\Phi \to \mathscr{P}(\mathcal{P}_\Phi)$ is a function such that $\bigcup \mathsf{im}(\mathsf{P}_\Phi^{\mathsf{pri}}) = \mathcal{P}_\Phi$ and moreover, for all $l, l' \in \mathcal{C}_\Phi$,
  - if $l \neq l'$ then $\mathsf{P}_\Phi^{\mathsf{pri}}(l) \cap \mathsf{P}_\Phi^{\mathsf{pri}}(l') = \emptyset$,
  - if $\mathsf{tc}_\Phi(l) \in \{1, \perp, \otimes, ⅋, !, ?\}$ then $\mathsf{card}(\mathsf{P}_\Phi^{\mathsf{pri}}(l)) = 1$,
  - if $\mathsf{tc}_\Phi(l) = ax$ (resp. $\mathsf{tc}_\Phi(l) = cut$) then $\mathsf{card}(\mathsf{P}_\Phi^{\mathsf{pri}}(l)) = 2$, (resp. $\mathsf{card}(\mathsf{P}_\Phi^{\mathsf{pri}}(l)) = 0$);

  for any $l \in \mathcal{C}_\Phi$, the elements of $\mathsf{P}_\Phi^{\mathsf{pri}}(l)$ are the *principal ports*, or *conclusions*, *of* l *in* Φ;
- $\mathsf{P}_\Phi^{\mathsf{aux}} \colon \mathcal{C}_\Phi \to \mathscr{P}(\mathcal{P}_\Phi)$ is a function such that, for all $l, l' \in \mathcal{C}_\Phi$,
  - if $l \neq l'$ then $\mathsf{P}_\Phi^{\mathsf{aux}}(l) \cap \mathsf{P}_\Phi^{\mathsf{aux}}(l') = \emptyset$,
  - if $\mathsf{tc}_\Phi(l) \in \{1, \perp, ax\}$ then $\mathsf{card}(\mathsf{P}_\Phi^{\mathsf{aux}}(l)) = 0$; if $\mathsf{tc}_\Phi(l) \in \{\otimes, ⅋, cut\}$ then $\mathsf{card}(\mathsf{P}_\Phi^{\mathsf{aux}}(l)) = 2$;

  for any $l \in \mathcal{C}_\Phi$, the elements of $\mathsf{P}_\Phi^{\mathsf{aux}}(l)$ are the *auxiliary ports*, or *premises*, *of* l *in* Φ;
- $\mathsf{P}_\Phi^{\mathsf{left}} \colon \mathcal{C}_\Phi^{\otimes,⅋} \to \mathcal{P}_\Phi$ is a function such that $\mathsf{P}_\Phi^{\mathsf{left}}(l) \in \mathsf{P}_\Phi^{\mathsf{aux}}(l)$ for any $l \in \mathcal{C}_\Phi^{\otimes,⅋}$;
- $\mathsf{tp}_\Phi \colon \mathcal{P}_\Phi \to \mathcal{F}_{\mathsf{MELL}}$ is a function (we write $p : A$ and we say that A is the *type of* p, when $\mathsf{tp}_\Phi(p) = A$) such that, for any $l \in \mathcal{C}_\Phi$, one has
  - if $\mathsf{tc}_\Phi(l) = ax$ (resp. $\mathsf{tc}_\Phi(l) = cut$) and $\mathsf{P}_\Phi^{\mathsf{pri}}(l) = \{p_1, p_2\}$ (resp. $\mathsf{P}_\Phi^{\mathsf{aux}}(l) = \{p_1, p_2\}$), then $\mathsf{tp}_\Phi(p_1) = A$ and $\mathsf{tp}_\Phi(p_2) = A^\perp$, for some $A \in \mathcal{F}_{\mathsf{MELL}}$,
  - if $\mathsf{tc}_\Phi(l) = A \in \{1, \perp\}$ and $\mathsf{P}_\Phi^{\mathsf{pri}}(l) = \{p\}$, then $\mathsf{tp}_\Phi(p) = A$,
  - if $\mathsf{tc}_\Phi(l) = \odot \in \{\otimes, ⅋\}$, $\mathsf{P}_\Phi^{\mathsf{pri}}(l) = \{p\}$, $\mathsf{P}_\Phi^{\mathsf{aux}}(l) = \{p_1, p_2\}$ and $\mathsf{P}_\Phi^{\mathsf{left}}(l) = p_1$, then $\mathsf{tp}_\Phi(p) = \mathsf{tp}_\Phi(p_1) \odot \mathsf{tp}_\Phi(p_2)$,
  - if $\mathsf{tc}_\Phi(l) = \lozenge \in \{!, ?\}$, $\mathsf{P}_\Phi^{\mathsf{pri}}(l) = \{p\}$ and $\mathsf{P}_\Phi^{\mathsf{aux}}(l) = \{p_1, \dots, p_n\}$ (with $n \in \mathbb{N}$), then $\mathsf{tp}_\Phi(p) = \lozenge A$ and $\mathsf{tp}_\Phi(p_i) = A$ for all $1 \leq i \leq n$, for some $A \in \mathcal{F}_{\mathsf{MELL}}$;
- $\mathcal{C}_\Phi^{\mathsf{box}} \subseteq \{l \in \mathcal{C}_\Phi^! \mid \mathsf{card}(\mathsf{P}_\Phi^{\mathsf{aux}}(l)) = 1\}$, the elements of $\mathcal{C}_\Phi^{\mathsf{box}}$ are the *box-cells of* Φ; for any $l \in \mathcal{C}_\Phi^{\mathsf{box}}$, its (unique) premise is denoted by $\mathsf{prid}_\Phi(l)$ and called the *principal door* or *pri-door of the box of* l (*in* R); we set $\mathcal{D}oors_\Phi^! = \bigcup \mathsf{P}_\Phi^{\mathsf{aux}}(\mathcal{C}_\Phi^{\mathsf{box}})$;[2]
- $\mathsf{box}_\Phi \colon \left(\bigcup \mathsf{P}_\Phi^{\mathsf{aux}}(\mathcal{C}_\Phi^{?,cut}) \cup \mathcal{D}oors_\Phi^!\right) \to \mathcal{C}_\Phi^{\mathsf{box}}$ is a partial function such that $\mathcal{D}oors_\Phi^! \subseteq \mathsf{dom}(\mathsf{box}_\Phi)$ and $\mathsf{box}_\Phi(\mathsf{prid}_\Phi(l)) = l$ for all $l \in \mathcal{C}_\Phi^{\mathsf{box}}$.[3]

We set: $\mathcal{P}_\Phi^{\mathsf{aux}} = \bigcup \mathsf{im}(\mathsf{P}_\Phi^{\mathsf{aux}})$, whose elements are the *auxiliary ports* of Φ; $\mathcal{P}_\Phi^{\mathsf{free}} = \mathcal{P}_\Phi \smallsetminus \mathcal{P}_\Phi^{\mathsf{aux}}$, whose elements are the *free ports*, or *conclusions*, *of* Φ; and $\mathcal{C}_\Phi^{\mathsf{free}} = \{l \in \mathcal{C}_\Phi \mid \mathsf{P}_\Phi^{\mathsf{pri}}(l) \subseteq \mathcal{P}_\Phi^{\mathsf{free}}\}$, whose elements are the *free*, or *terminal*, *cells* of Φ.[4]

[2] Hence, $\mathcal{D}oors_\Phi^! = \{\mathsf{prid}_\Phi(l) \mid l \in \mathcal{C}_\Phi^{\mathsf{box}}\}$, the set of premises of all *box*-cells of Φ.

[3] So, box_Φ is defined on $\mathcal{D}oors_\Phi^!$ and maps the (unique) premise of a *box*-cell l into l itself.

[4] Thus, a cell l of a pps Φ is in $\mathcal{C}_\Phi^{\mathsf{free}}$ iff either l is a *ax*-cell and both its conclusions are in $\mathcal{P}_\Phi^{\mathsf{free}}$, or l is a *cut*-cell, or l is neither an *ax*- nor a *cut*-cell and its unique conclusion is in $\mathcal{P}_\Phi^{\mathsf{free}}$.

For any pps $\varPhi$, the *ground-structure* (*gs* for short) *of* $\varPhi$ is the 7-tuple $G_\varPhi = (\mathcal{P}_\varPhi, \mathcal{C}_\varPhi, \mathsf{tc}_\varPhi, \mathsf{P}^{\mathsf{pri}}_\varPhi, \mathsf{P}^{\mathsf{aux}}_\varPhi, \mathsf{P}^{\mathsf{left}}_\varPhi, \mathsf{tp}_\varPhi)$.

A pps $\varPhi$ is *fat* (resp. *strongly fat*) if $\mathsf{card}(\mathsf{P}^{\mathsf{aux}}_\varPhi(l)) \geq 1$ (resp. $\mathsf{card}(\mathsf{P}^{\mathsf{aux}}_\varPhi(l)) \geq 2$) for all $l \in \mathcal{C}^{!}_\varPhi$.

Let us make some comments on Def. 1. Let $\varPhi$ be a pps.

- The function $\mathsf{P}^{\mathsf{left}}_\varPhi$ fixes an order on the two premises of any $\otimes$- and $⅋$-cell of $\varPhi$; the premises of the other types of cells are unordered, as well as the conclusions of the *ax*-cells.
- The conditions $\bigcup \mathsf{im}(\mathsf{P}^{\mathsf{pri}}_\varPhi) = \mathcal{P}_\varPhi$ and "for all $l, l' \in \mathcal{C}_\varPhi$, if $l \neq l'$ then $\mathsf{P}^{\mathsf{pri}}_\varPhi(l) \cap \mathsf{P}^{\mathsf{pri}}_\varPhi(l') = \emptyset = \mathsf{P}^{\mathsf{aux}}_\varPhi(l) \cap \mathsf{P}^{\mathsf{aux}}_\varPhi(l')$" mean that every port is conclusion of exactly one cell and premise of at most one cell; the elements of $\mathcal{P}^{\mathsf{free}}_\varPhi$ are the ports of $\varPhi$ that are not premises of any cell.
- No condition is required for $\mathsf{card}(\mathsf{P}^{\mathsf{aux}}_\varPhi(l))$ when $l \in \mathcal{C}^{!,?}_\varPhi$: l can have $n \in \mathbb{N}$ premises since we use generalized ?- and !-cells for (co-)contraction, (co-)weakening and (co-)dereliction.
- The gs $G_\varPhi$ of $\varPhi$ is obtained from $\varPhi$ by forgetting $\mathsf{box}_\varPhi$ and $\mathcal{C}^{\mathsf{box}}_\varPhi$. In a way, $G_\varPhi$ encodes the "geometric structure" of $\varPhi$ (see below).

For any pps $\varPhi$, the fact that $\mathsf{box}_\varPhi$ is defined on $\mathcal{D}oors^{!}_\varPhi$ is not needed but it simplifies the definition of the function $\mathsf{box}^{\mathsf{ext}}_{\mathcal{P}_\varPhi}$ (Def. 6), an extension of $\mathsf{box}_\varPhi$ that will be useful in the sequel. Provided that some suitable conditions are fulfilled (Def. 8), any *box*-cell l of $\varPhi$ is the starting point to compute the box associated with l partial function $\mathsf{box}_\varPhi$ allows to recover the border of this box. In general, not all !-cells of $\varPhi$ with exactly one premise are *box*-cells.

▶ **Notation.** For any pps $\varPhi$ we set $\mathcal{D}oors_\varPhi = \mathsf{dom}(\mathsf{box}_\varPhi)$ and $\mathcal{D}oors^{?}_\varPhi = \mathcal{D}oors_\varPhi \cap \bigcup \overline{\mathsf{P}^{\mathsf{aux}}_\varPhi}(\mathcal{C}^{?}_\varPhi)$, $\mathcal{D}oors^{cut}_\varPhi = \mathcal{D}oors_\varPhi \cap \bigcup \overline{\mathsf{P}^{\mathsf{aux}}_\varPhi}(\mathcal{C}^{cut}_\varPhi)$ and $\mathcal{C}^{\mathsf{bord}}_\varPhi = \mathcal{C}^{\mathsf{box}}_\varPhi \cup \{l \in \mathcal{C}^{?}_\varPhi \mid \exists\, p \in \mathcal{D}oors^{?}_\varPhi \cap \mathsf{P}^{\mathsf{aux}}_\varPhi(l)\}$. From now on, $\bullet \notin \mathcal{C}_\varPhi$ (in particular, $\bullet \notin \mathcal{C}^{\mathsf{box}}_\varPhi$) for any pps $\varPhi$.

With the gs $G_\varPhi$ of any pps $\varPhi$ is naturally associated a directed labeled graph $\mathfrak{G}(G_\varPhi)$: its nodes are the cells of $\varPhi$, labeled by their type; its oriented edges are the ports of $\varPhi$, labeled by their type; a premise (resp. conclusion) of a cell l is incoming in (resp. outgoing from) l.

In the graphical representation of a pps $\varPhi$, a dotted arrow is depicted from a premise q of a ?-cell or *cut*-cell to the premise of a *box*-cell l when $q \in \mathsf{box}^{-1}_\varPhi(l)$. In pictures, the label of a *box*-cell is marked as $!\mathsf{p}$, and the names or types of ports and cells are sometimes omitted.

▶ **Definition 2** ((Pre-)order on the ports of a pre-proof-structure)**.** Let $\varPhi$ be a gs. The binary relation $<^{1}_\varPhi$ on $\mathcal{P}_\varPhi$ is defined by: $p <^{1}_\varPhi q$ if there exists $l \in \mathcal{C}_\varPhi$ such that $p \in \mathsf{P}^{\mathsf{pri}}_\varPhi(l)$ and $q \in \mathsf{P}^{\mathsf{aux}}_\varPhi(l)$. The preorder relation $\leq_\varPhi$ on $\mathcal{P}_\varPhi$ is the reflexive-transitive closure of $<^{1}_\varPhi$. When $p \leq_\varPhi q$ we say that q is *above* p. We write $p <_\varPhi q$ if $p \leq_\varPhi q$ and $p \neq q$.

In a pps $\varPhi$, the binary relation $\leq_\varPhi$ has a geometric meaning (note that $\mathcal{C}^{\mathsf{box}}_\varPhi$ and $\mathsf{box}_\varPhi$, as well as $\mathsf{tc}_\varPhi$, $\mathsf{P}^{\mathsf{left}}_\varPhi$ and $\mathsf{tp}_\varPhi$, play no role in Def. 2): for any $p, q \in \mathcal{P}_\varPhi$, if $p \leq_\varPhi q$ then in the directed graph $\mathfrak{G}(G_\varPhi)$ there is a directed path from q to p that does not cross any *ax*- or *cut*-cell.

▶ Remark (Predecessor of a port). Let $\varPhi$ be a pps. For all $p \in \mathcal{P}^{\mathsf{aux}}_\varPhi \smallsetminus \overline{\mathsf{P}^{\mathsf{aux}}_\varPhi}(\mathcal{C}^{cut}_\varPhi)$, there is a unique $q \in \mathcal{P}_\varPhi$ (denoted by $\mathsf{pred}_\varPhi(p)$, the *predecessor of* p) such that $q <^{1}_\varPhi p$; moreover $\mathsf{pred}_\varPhi(p) \neq p$. Indeed, by hypothesis p is a premise of some cell of $\varPhi$, but the only cells with more than one conclusion are the *ax*-cells, which have no premises; so, p is a premise of a cell of $\varPhi$ having just one conclusion q; also, $\mathsf{tp}_\varPhi(p)$ is a proper subformula of $\mathsf{tp}_\varPhi(q)$, thus $p \neq q$.

▶ **Fact 3** (Tree-like order on ports)**.** *Let $\varPhi$ be a pps: $\leq_\varPhi$ is a tree-like order relation on $\mathcal{P}_\varPhi$.*

According to Fact 3, a pps $\varPhi$ cannot have "vicious cycles" like for example a cell l such that $\mathsf{P}^{\mathsf{pri}}_\varPhi(l) \cap \mathsf{P}^{\mathsf{aux}}_\varPhi(l) \neq \emptyset$ (*i.e.* a port cannot be both a premise and a conclusion of a cell l).

(a)

$$\begin{tikzcd}
\mathscr{P}(\mathcal{P}_\Phi) \arrow[d, "\overline{\varphi_\mathcal{P}}"'] & \mathcal{C}_\Phi \arrow[l, "\mathsf{P}^{\mathsf{aux}}_\Phi"] \arrow[r, "\mathsf{P}^{\mathsf{pri}}_\Phi"'] \arrow[d, "\varphi_\mathcal{C}"'] & \mathscr{P}(\mathcal{P}_\Phi) \arrow[d, "\overline{\varphi_\mathcal{P}}"] \\
\mathscr{P}(\mathcal{P}_\Psi) & \mathcal{C}_\Psi \arrow[l, "\mathsf{P}^{\mathsf{aux}}_\Psi"'] \arrow[r, "\mathsf{P}^{\mathsf{pri}}_\Psi"] & \mathscr{P}(\mathcal{P}_\Psi)
\end{tikzcd}
\quad
\begin{tikzcd}
\mathcal{C}_\Phi \arrow[r, "\mathsf{tc}_\Phi"'] \arrow[d, "\varphi_\mathcal{C}"'] & \mathcal{L}_{\mathsf{MELL}} \\
\mathcal{C}_\Psi \arrow[ur, "\mathsf{tc}_\Psi"'] &
\end{tikzcd}
\quad
\begin{tikzcd}
\mathcal{P}_\Phi \arrow[r, "\mathsf{tp}_\Phi"'] \arrow[d, "\varphi_\mathcal{P}"'] & \mathcal{F}_{\mathsf{MELL}} \\
\mathcal{P}_\Psi \arrow[ur, "\mathsf{tp}_\Psi"'] &
\end{tikzcd}
\quad
\begin{tikzcd}
\mathcal{C}^{\otimes,\parr}_\Phi \arrow[r, "\mathsf{P}^{\mathsf{left}}_\Phi"'] \arrow[d, "\varphi_\mathcal{C}"'] & \mathcal{P}_\Phi \arrow[d, "\varphi_\mathcal{P}"] \\
\mathcal{C}^{\otimes,\parr}_\Psi \arrow[r, "\mathsf{P}^{\mathsf{left}}_\Psi"] & \mathcal{P}_\Psi
\end{tikzcd}$$

(b)

$$\begin{tikzcd}
\mathcal{D}oors_\Phi \arrow[r, "\mathsf{box}_\Phi"'] \arrow[d, "\varphi_\mathcal{P}"] & \mathcal{C}^{\mathsf{box}}_\Phi \arrow[d, "\varphi_\mathcal{C}"'] \\
\mathcal{D}oors_\Psi \arrow[r, "\mathsf{box}_\Psi"] & \mathcal{C}^{\mathsf{box}}_\Psi
\end{tikzcd}$$

Figure 2 Commutative diagrams for isomorphism of gs (Fig. 2a) and of pps (Fig. 2b). See Def. 4.

The names of ports and cells of a pps (ports and cells being nothing but their names) will be important to define the labeled Taylor expansion (Def. 11), a more informative variant of the usual Taylor expansion (Def. 15). Nevertheless, a precise answer to the question "When two pps can be considered equal?" leads naturally to the notion of isomorphism between pps (Def. 4), inspired by the notion of isomorphism between graphs: intuitively, two pps are isomorphic if they are identical up to the names of their ports and cells.

▶ **Definition 4** (Isomorphism on ground-structures and pre-proof-structures)**.** Let Φ, Ψ be pps.

An *isomorphism from* G_Φ *to* G_Ψ is a pair $\varphi = (\varphi_\mathcal{P}, \varphi_\mathcal{C})$ of bijections $\varphi_\mathcal{P} \colon \mathcal{P}_\Phi \to \mathcal{P}_\Psi$ and $\varphi_\mathcal{C} \colon \mathcal{C}_\Phi \to \mathcal{C}_\Psi$ such that the diagrams in Fig. 2a commute. We write then $\varphi \colon G_\Phi \simeq G_\Psi$.

An *isomorphism from* Φ *to* Ψ is a pair $\varphi = (\varphi_\mathcal{P}, \varphi_\mathcal{C})$ of bijections $\varphi_\mathcal{P} \colon \mathcal{P}_\Phi \to \mathcal{P}_\Psi$ and $\varphi_\mathcal{C} \colon \mathcal{C}_\Phi \to \mathcal{C}_\Psi$ such that $\varphi \colon G_\Phi \simeq G_\Psi$, $\mathsf{im}(\varphi_\mathcal{C}\restriction_{\mathcal{C}^{\mathsf{box}}_\Phi}) = \mathcal{C}^{\mathsf{box}}_\Psi$, $\mathsf{im}(\varphi_\mathcal{P}\restriction_{\mathcal{D}oors_\Phi}) = \mathcal{D}oors_\Psi$ and the diagram in Fig. 2b commutes. We write then $\varphi \colon \Phi \simeq \Psi$.
If there is an isomorphism from Φ to Ψ, we say: Φ *and* Ψ *are isomorphic* and we write $\Phi \simeq \Psi$.

The relation $\simeq$ is an equivalence on the set of pps. Equivalence classes for $\simeq$ share the same graphical representation up to the order of the premises of their !- and ?-cells: any such representation can be seen as a canonical representative of an equivalence class.

▶ Remark. Let Φ and Ψ be some pps with $\varphi = (\varphi_\mathcal{P}, \varphi_\mathcal{C}) \colon G_\Phi \simeq G_\Psi$. We have:

1. $\mathsf{card}(\mathsf{P}^{\mathsf{aux}}_\Phi(l)) = \mathsf{card}(\mathsf{P}^{\mathsf{aux}}_\Psi(\varphi_\mathcal{C}(l)))$ for every $l \in \mathcal{C}_\Phi$, in particular Φ is fat (resp. strongly fat) iff Ψ is fat (resp. strongly fat); moreover, $\mathcal{P}^{\mathsf{free}}_\Psi = \overline{\varphi_\mathcal{P}}(\mathcal{P}^{\mathsf{free}}_\Phi)$ and $\mathcal{C}^{\mathsf{free}}_\Psi = \overline{\varphi_\mathcal{C}}(\mathcal{C}^{\mathsf{free}}_\Phi)$;
2. for every $p, q \in \mathcal{P}_\Phi$, $p \leq_\Phi q$ implies $\varphi_\mathcal{P}(p) \leq_\Psi \varphi_\mathcal{P}(q)$ ($\varphi_\mathcal{P}$ is non-decreasing).

DiLL-, DiLL$_0$- and MELL-proof-structures. A pps Φ is a very "light" structure and in order to associate with any $l \in \mathcal{C}^{\mathsf{box}}_\Phi$ the sub-pps of Φ usually called the box of l, some conditions need to be satisfied: for example, boxes have to be ordered by a tree-like order (nesting), *cut*- and *ax*-cells cannot cross the border of a box, *etc.* We introduce here some restrictions to pps in order to define proof-structures corresponding to some fragments or extension of LL: MELL, DiLL and DiLL$_0$. Full differential linear logic (DiLL) is an extension of MELL (with the same language as MELL) provided with both promotion rule (*i.e.* boxes) and co-structural rules (the duals of the structural rules handling the ?-modality) for the !-modality: DiLL$_0$ and MELL are particular subsystems of DiLL, respectively the promotion-free one (*i.e.* without boxes) and the one without co-structural rules. Our interest for DiLL is just to have an unitary syntax subsuming both MELL and DiLL$_0$ without considering cut-elimination: for this reason, unlike [16, 20], our DiLL-ps are not allowed to contain a set of DiLL-ps inside a box.

▶ **Definition 5** (DiLL$_0$-proof-structure)**.** A DiLL$_0$*-proof structure* (DiLL$_0$*-ps* or *diffnet* for short) is a pps Φ with $\mathcal{C}^{\mathsf{box}}_\Phi = \emptyset$. The set of DiLL$_0$-ps is denoted by $\mathbf{PS}_{\mathsf{DiLL}_0}$, and $\rho, \sigma, \ldots$ range over it.

So, a DiLL$_0$-ps ρ is a pps without *box*-cells: in this case, box_ρ is the empty function. Thus, any DiLL$_0$-ps ρ can be identified with its gs G_ρ.

To define the conditions that a pps has to fulfill to be a DiLL-ps, we first extend the partial function box_{Φ} to a function $\mathsf{box}^{\mathsf{ext}}_{\mathcal{P}_{\Phi}}$ that associates with every port p of Φ the "deepest" *box*-cell (if any) whose box contains p; it returns a dummy element $\bullet$ if p is not contained in any box.

▶ **Definition 6** (Extension of box_{Φ})**.** Let Φ be a pps. The *extension of* box_{Φ} is a function $\mathsf{box}^{\mathsf{ext}}_{\mathcal{P}_{\Phi}} \colon \mathcal{P}_{\Phi} \to \mathcal{C}^{\mathsf{box}}_{\Phi} \cup \{\bullet\}$ defined as follows: for any $p \in \mathcal{P}_{\Phi}$,

$$\mathsf{box}^{\mathsf{ext}}_{\mathcal{P}_{\Phi}}(p) = \begin{cases} \mathsf{box}_{\Phi}(\max_{\leq_{\Phi}}(\downarrow_{\mathcal{P}_{\Phi}} p \cap \mathcal{D}oors_{\Phi})) & \text{if } \downarrow_{\mathcal{P}_{\Phi}} p \cap \mathcal{D}oors_{\Phi} \neq \emptyset \\ \bullet & \text{otherwise.} \end{cases}$$

For every pps Φ, the function $\mathsf{box}^{\mathsf{ext}}_{\mathcal{P}_{\Phi}}$ is well-defined since , for all $p \in \mathcal{P}_{\Phi}$, the set $\downarrow_{\mathcal{P}_{\Phi}} p \cap \mathcal{D}oors_{\Phi}$ is finite and totally ordered by $\leq_{\Phi}$, according to Fact 3: therefore the greatest element of $\downarrow_{\mathcal{P}_{\Phi}} p \cap \mathcal{D}oors_{\Phi}$ exists as soon as $\downarrow_{\mathcal{P}_{\Phi}} p \cap \mathcal{D}oors_{\Phi} \neq \emptyset$.

In a pps Φ, computing $\mathsf{box}^{\mathsf{ext}}_{\mathcal{P}_{\Phi}}$ from box_{Φ} is simple. Given a port p of Φ, consider the maximal downwards path starting from p in the directed graph $\mathfrak{G}(G_{\Phi})$: the first time the path bumps into a port $q \in \mathcal{D}oors_{\Phi}$ (if any), we set $\mathsf{box}^{\mathsf{ext}}_{\mathcal{P}_{\Phi}}(p) = \mathsf{box}_{\Phi}(q)$; if the path does not bump into any $q \in \mathcal{D}oors_{\Phi}$, then $\mathsf{box}^{\mathsf{ext}}_{\mathcal{P}_{\Phi}}(p) = \bullet$.

▶ **Definition 7** (Preorder on *box*-cells of a pre-proof-structure)**.** Let Φ be a pps. The binary relation $\leq_{\mathcal{C}^{\mathsf{box}}_{\Phi}}$ on $\mathcal{C}^{\mathsf{box}}_{\Phi}$ is defined by: $l \leq_{\mathcal{C}^{\mathsf{box}}_{\Phi}} l'$ (say l' *is above* l) iff there are $p, p' \in \mathcal{D}oors_{\Phi}$ such that $p \leq_{\Phi} p'$, $\mathsf{box}_{\Phi}(p) = l$ and $\mathsf{box}_{\Phi}(p') = l'$. We write $l <_{\mathcal{C}^{\mathsf{box}}_{\Phi}} l'$ if $l \leq_{\mathcal{C}^{\mathsf{box}}_{\Phi}} l'$ and $l \neq l'$.

The binary relation $\leq_{\mathcal{C}^{\mathsf{box}}_{\Phi} \cup \{\bullet\}}$ on $\mathcal{C}^{\mathsf{box}}_{\Phi} \cup \{\bullet\}$ is defined by: $l \leq_{\mathcal{C}^{\mathsf{box}}_{\Phi} \cup \{\bullet\}} l'$ if either $l \leq_{\mathcal{C}^{\mathsf{box}}_{\Phi}} l'$ or $l = \bullet$. We write $l <_{\mathcal{C}^{\mathsf{box}}_{\Phi} \cup \{\bullet\}} l'$ when $l \leq_{\mathcal{C}^{\mathsf{box}}_{\Phi} \cup \{\bullet\}} l'$ and $l \neq l'$.

In any pps Φ, $\leq_{\mathcal{C}^{\mathsf{box}}_{\Phi}}$ is a preorder on $\mathcal{C}^{\mathsf{box}}_{\Phi}$, since $\leq_{\Phi}$ is a preorder on $\mathcal{P}_{\Phi}$. The preorder $\leq_{\mathcal{C}^{\mathsf{box}}_{\Phi} \cup \{\bullet\}}$ is the extension of $\leq_{\mathcal{C}^{\mathsf{box}}_{\Phi}}$ obtained by adding $\bullet$ as least element.

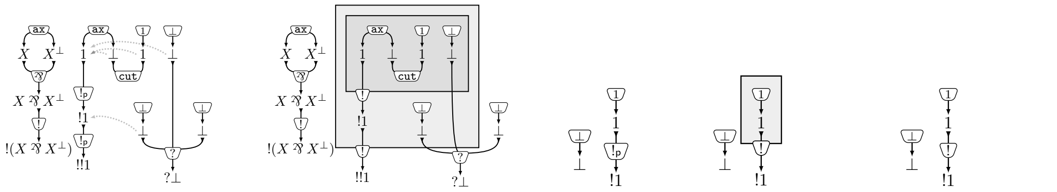
In Fig. 1d, Ξ is a pps such that $\leq_{\mathcal{C}^{\mathsf{box}}_{\Xi}}$ is not an order on $\mathcal{C}^{\mathsf{box}}_{\Xi}$; Ξ' is a pps such that $\leq_{\mathcal{C}^{\mathsf{box}}_{\Xi'}}$ is an order but not a tree-like order on $\mathcal{C}^{\mathsf{box}}_{\Xi'}$. A condition that a pps Φ must fulfill to be a DiLL-ps is just that $\leq_{\mathcal{C}^{\mathsf{box}}_{\Phi}}$ is a tree-like order (or equivalently, $\leq_{\mathcal{C}^{\mathsf{box}}_{\Phi} \cup \{\bullet\}}$ is a rooted tree-like order whose root is $\bullet$): this essentially amounts to the nesting of boxes (see [12] for details).

▶ **Definition 8** (DiLL-proof-structure and MELL-proof-structure)**.** A DiLL-*proof-structure* (DiLL-*ps* for short) is a pps Φ such that:

1. $\leq_{\mathcal{C}^{\mathsf{box}}_{\Phi}}$ is a tree-like order on $\mathcal{C}^{\mathsf{box}}_{\Phi}$;
2. $\mathsf{box}^{\mathsf{ext}}_{\mathcal{P}_{\Phi}}(p) = \mathsf{box}^{\mathsf{ext}}_{\mathcal{P}_{\Phi}}(q)$ for all $l \in \mathcal{C}^{ax}_{\Phi}$ with $\mathsf{P}^{\mathsf{pri}}_{\Phi}(l) = \{p, q\}$ and all $l \in \mathcal{C}^{cut}_{\Phi}$ with $\mathsf{P}^{\mathsf{aux}}_{\Phi}(l) = \{p, q\}$;
3. for all $p \in \mathcal{D}oors^{!}_{\Phi} \cup \mathcal{D}oors^{?}_{\Phi}$, one has $\mathsf{box}_{\Phi}(p) \neq \mathsf{box}^{\mathsf{ext}}_{\mathcal{P}_{\Phi}}(\mathsf{pred}_{\Phi}(p))$;
4. for all $l \in \mathcal{C}^{\mathsf{box}}_{\Phi} \cup \{\bullet\}$ and $p \in \mathcal{D}oors^{!}_{\Phi}$, if $l <_{\mathcal{C}^{\mathsf{box}}_{\Phi} \cup \{\bullet\}} \mathsf{box}_{\Phi}(p)$ then $l \leq_{\mathcal{C}^{\mathsf{box}}_{\Phi} \cup \{\bullet\}} \mathsf{box}^{\mathsf{ext}}_{\mathcal{P}_{\Phi}}(\mathsf{pred}_{\Phi}(p))$.

A MELL-*proof-structure* (MELL-*ps* for short) is a DiLL-ps Φ such that $\mathcal{C}^{\mathsf{box}}_{\Phi} = \mathcal{C}^{!}_{\Phi}$. The set of DiLL-ps (resp. MELL-ps) is denoted by $\mathbf{PS}_{\mathsf{DiLL}}$ (resp. $\mathbf{PS}_{\mathsf{MELL}}$) and $R, S, \ldots$ range over it.

In Def. 8, condition 2 means that a *cut*-cell (resp. *ax*-cell) cannot cross the border of a box, *i.e.* its premises (resp. conclusions) belong to the same boxes; the pps Φ in Fig. 1a does not fulfill condition 2. Condition 3 in Def. 8 entails that two ports on the border of the same box cannot be above each other (in the sense of $\leq_{\Phi}$); the pps Ψ_1 and Ψ_2 in Fig. 1b do not fulfill condition 3. Condition 4 in Def. 8 implies that the border of a box cannot have more than one !-cell: when the premise of a !-cell l' belongs to the box associated with a *box*-cell $l \neq l'$, then l' is itself contained in the box of l. The pps X in Fig. 1c does not fulfill condition 4, even if it satisfies conditions 1-3. See [12] for more details.



(a) $S \in \mathbf{PS}_{\mathsf{DiLL}} \smallsetminus (\mathbf{PS}_{\mathsf{MELL}} \cup \mathbf{PS}_{\mathsf{DiLL}_0})$. **(b)** $R_1 \in \mathbf{PS}_{\mathsf{MELL}} \smallsetminus \mathbf{PS}_{\mathsf{DiLL}_0}$. **(c)** $R_2 \in \mathbf{PS}_{\mathsf{DiLL}_0} \smallsetminus \mathbf{PS}_{\mathsf{MELL}}$.

Figure 3 Some examples of DiLL-ps. In R_1 (Fig. 3b) $\mathcal{C}_{R_1}^{\mathsf{box}} = \{l\}$ and box_{R_1} is the empty function. In R_2 (Fig. 3c) $\mathcal{C}_{R_2}^{\mathsf{box}} = \emptyset$, so box_{R_2} is the empty function. Both S (Fig. 3a) and R_1 (Fig. 3b) are in two different presentations: the "arrow-like" one (on the left) and the "inductive-like" one (on the right).

In [12] we show that the information encoded in a DiLL-ps R is enough to associate a box R_l with any *box*-cell l of R. So, as usual for LL, R_l can be graphically depicted (instead of using dotted arrows to pick out $\mathsf{box}_R^{-1}(l)$) by a rectangular frame containing all ports in $\mathsf{inbox}_R(l)$ (see Def. 9). Some examples of DiLL-ps are in Fig. 3.

▶ **Definition 9** (Content of the box, depth)**.** Let R be a DiLL-ps.

For any $l \in \mathcal{C}_R^{\mathsf{box}}$, the *content of the box of* l is $\mathsf{inbox}_R(l) = \{q \in \mathcal{P}_R \mid l \leq_{\mathcal{C}_R^{\mathsf{box}}} \mathsf{box}_{\mathcal{P}_R}^{\mathsf{ext}}(q)\}$.

The function $\mathsf{box}_{\mathcal{C}_R}^{\mathsf{ext}} \colon \mathcal{C}_R \to \mathcal{C}_R^{\mathsf{box}}$ is defined by: for every $l \in \mathcal{C}_R \smallsetminus \mathcal{C}_R^{cut}$ (resp. $l \in \mathcal{C}_R^{cut}$), we set $\mathsf{box}_{\mathcal{C}_R}^{\mathsf{ext}}(l) = \mathsf{box}_{\mathcal{P}_R}^{\mathsf{ext}}(p)$ where $p \in \mathsf{P}_R^{\mathsf{pri}}(l)$ (resp. $p \in \mathsf{P}_R^{\mathsf{aux}}(l)$).[5]

For every $p \in \mathcal{P}_R$ and $l \in \mathcal{C}_R$, the *depths of* p *and* l *in* R are defined as follows: $\mathsf{depth}_{\mathcal{P}_R}(p) = \mathsf{card}(\downarrow_{\mathcal{C}_R^{\mathsf{box}}}(\mathsf{box}_{\mathcal{P}_R}^{\mathsf{ext}}(p)))$ and $\mathsf{depth}_{\mathcal{C}_R}(l) = \mathsf{card}(\downarrow_{\mathcal{C}_R^{\mathsf{box}}}(\mathsf{box}_{\mathcal{C}_R}^{\mathsf{ext}}(l)))$. The *depth of* R is the natural number $\mathsf{depth}(R) = \sup\{\mathsf{depth}_{\mathcal{P}_R}(p) \mid p \in \mathcal{P}_R\}$.

Given a DiLL-ps R, for any *box*-cell l in R, $\mathsf{inbox}_R(l)$ represents the set of ports contained in the box of l. According to Definition 9, the meaning of $\mathsf{box}_{\mathcal{P}_R}^{\mathsf{ext}}$ is clear: for any port p of R, $\downarrow_{\mathcal{C}_R^{\mathsf{box}}}(\mathsf{box}_{\mathcal{P}_R}^{\mathsf{ext}}(p)) = \{l \in \mathcal{C}_R^{\mathsf{box}} \mid p \in \mathsf{inbox}_R(l)\}$ is the set of boxes in R containing p, and if $\mathsf{box}_{\mathcal{P}_R}^{\mathsf{ext}}(p) = \bullet$ then p has depth 0 (no box in R contains p), otherwise $\mathsf{box}_{\mathcal{P}_R}^{\mathsf{ext}}(p)$ is the deepest *box*-cell in R whose box contains p; the depth of p in R is the number of nested boxes in R containing p. According to Def. 9, for any *box*-cell l, $\mathsf{depth}_{\mathcal{P}_R}(\mathsf{prid}_R(l)) = \mathsf{depth}_{\mathcal{C}_R}(l) + 1$.

In a DiLL-ps R the ports in $\mathcal{D}oors_R^! \cup \mathcal{D}oors_R^?$ are the ones in the border of some box: more precisely, for any $p \in \mathcal{D}oors_R^! \cup \mathcal{D}oors_R^?$, p is in the border of the box of every *box*-cell l of R such that $\mathsf{box}_{\mathcal{P}_R}^{\mathsf{ext}}(\mathsf{pred}_R(p)) <_{\mathcal{C}_R^{\mathsf{box}} \cup \{\bullet\}} l \leq_{\mathcal{C}_R^{\mathsf{box}}} \mathsf{box}_{\mathcal{P}_R}^{\mathsf{ext}}(p)$. By conditions 1 and 3-4 in Def. 8, (the premise of) a *box*-cell is in the border of exactly one box: for any $l \in \mathcal{C}_R^{\mathsf{box}}$ with $\mathsf{P}_R^{\mathsf{pri}}(l) = \{p\}$, one has $\mathsf{box}_{\mathcal{P}_R}^{\mathsf{ext}}(p) <_{\mathcal{C}_R^{\mathsf{box}} \cup \{\bullet\}} l$ and there is no $l' \in \mathcal{C}_R^{\mathsf{box}} \cup \{\bullet\}$ such that $\mathsf{box}_{\mathcal{P}_R}^{\mathsf{ext}}(p) <_{\mathcal{C}_R^{\mathsf{box}} \cup \{\bullet\}} l' <_{\mathcal{C}_R^{\mathsf{box}} \cup \{\bullet\}} l$. This does not hold in general for ?-cells in $\mathcal{C}_R^{\mathsf{bord}}$, since we use generalized ?-links: a premise of a ?-cell can cross the border of several boxes, see for instance one of the premises of the ?-cell whose conclusion is of type ?⊥ in Fig. 3a.

3 Computing the Taylor expansion of a DiLL-proof-structure

The Taylor expansion of a MELL-ps, or more generally a DiLL-ps, R is a (usually infinite) set of DiLL_0-ps: roughly speaking, each element of the Taylor expansion of R is obtained from R by replacing each box B in R with n_B copies of its content (for some $n_B \in \mathbb{N}$), recursively on the depth of R. Note that n_B depends not only on B but also on which "copy" of the contents

[5] For every $l \in \mathcal{C}_R$, $\mathsf{box}_{\mathcal{C}_R}^{\mathsf{ext}}(l)$ is well-defined by condition 2 in Def. 8. Note that, for any $l \in \mathcal{C}_R^{\mathsf{box}}$, $\mathsf{box}_{\mathcal{C}_R}^{\mathsf{ext}}(l)$ is the immediate predecessor of l in the tree-like order $\leq_{\mathcal{C}_R^{\mathsf{box}} \cup \{\bullet\}}$.

of all boxes containing B we are considering. Usually, the Taylor expansion of MELL-ps [15, 17] is defined globally and inductively: with every MELL-ps R is directly associated its Taylor expansion (the whole set!) by induction on the depth of R. We adopt an alternative approach, which is pointwise and non-inductive: visually, it is exemplified by Fig. 4.

We introduce here *Taylor-functions*: a Taylor-function of a DiLL-ps R ascribes recursively a number of copies for each box of R. Any element of the Taylor expansion of R can be built from (at least) one element of the *proto-Taylor expansion* $\mathcal{T}_R^{\mathsf{proto}}$ of R, $\mathcal{T}_R^{\mathsf{proto}}$ being the set of Taylor-functions of R. We build in this way a more informative version of the Taylor expansion of R, the *labeled Taylor-expansion* $\mathcal{T}_R$ of R: one of the advantages of our pointwise and non-inductive approach is that it is easy to define the correspondence between ports and cells of any element ρ of the Taylor expansion of R and ports and cells of R (an operation intuitively clear but very awkward to define with the global and inductive approach), and to differentiate the various copies in ρ of the content of a same box in R. For this purpose, any port (or cell) of any DiLL_0-ps in the labeled Taylor expansion of R is of the shape (p, a), where p is the corresponding port (or cell) of R and the finite sequence a has to be intended as a list of indexes saying in which copy of the content of each box (p, a) is. These indexes are a syntactic counterpart of the ones used in the definition of k-experiment of *PLPS* in [5, Def. 35]. The information encoded in any element of the labeled Taylor expansion will be useful to prove some fundamental lemmas in §4. The usual Taylor expansion of a DiLL-ps R (whose elements do not contain this information, Def. 15) is then the quotient of $\mathcal{T}_R$ modulo isomorphism, *i.e.* modulo renaming of ports and cells of any DiLL_0-ps in $\mathcal{T}_R$.

▶ **Definition 10** (Taylor-function of a DiLL-proof-structure)**.** Let R be a DiLL-ps. A *Taylor-function of* R is a function $\mathsf{f}\colon \mathcal{C}_R^{\mathsf{box}} \cup \{\bullet\} \to \mathscr{P}_{\text{fin}}(\mathbb{N}^*)$ such that:

1. *(depth compatibility)* $\mathsf{f}(\bullet) = \{(\,)\}$ and $|a| = \mathsf{depth}_{\mathcal{P}_R}(\mathsf{prid}_R(l))$, for any $l \in \mathcal{C}_R^{\mathsf{box}}$ and $a \in \mathsf{f}(l)$;
2. *(vertical downclosure)* for all $l, l' \in \mathcal{C}_R^{\mathsf{box}}$ such that $l \leq_{\mathcal{C}_R^{\mathsf{box}}} l'$, with $k = \mathsf{depth}_{\mathcal{P}_R}(\mathsf{prid}_R(l))$ and $k' = \mathsf{depth}_{\mathcal{P}_R}(\mathsf{prid}_R(l'))$ (so $k \leq k'$), if $(n_1, \dots, n_k, \dots, n_{k'}) \in \mathsf{f}(l')$ then $(n_1, \dots, n_k) \in \mathsf{f}(l)$.

The *proto-Taylor expansion of* R is the set $\mathcal{T}_R^{\mathsf{proto}}$ of Taylor-functions of R.

Note that the notion of Taylor-function of a DiLL-ps R relies only on the tree-like order on $\mathcal{C}_R^{\mathsf{box}}$, hence we could define the Taylor-function of any tree. By the vertical downclosure condition, any Taylor-function of a DiLL-ps R can be naturally presented as a tree-like order which is an "level-by-level expansion" of the tree-like order on $\mathcal{C}_R^{\mathsf{box}}$: see Fig. 4a–4c.

Our approach in defining the elements of the Taylor expansion of a DiLL-ps R separates the analysis of the number of copies to take for each (copy of) box of R (every Taylor-function of R contains this information, which is the most important one) from the operation of copying the content of each box (given by the function τ_R defined below). Indeed, with any Taylor-function of R one can associate a unique element of the (labeled) Taylor expansion of R (Def. 11).

▶ **Definition 11** (Labeled Taylor expansion)**.** Let R be a DiLL-ps. The function $\tau_R\colon \mathcal{T}_R^{\mathsf{proto}} \to \mathbf{PS}_{\mathsf{DiLL}_0}$ associates with any $\mathsf{f} \in \mathcal{T}_R^{\mathsf{proto}}$ a DiLL_0-ps $\tau_R(\mathsf{f})$ defined by: $\mathcal{C}_{\tau_R(\mathsf{f})}^{\mathsf{box}} = \emptyset$, $\mathsf{box}_{\tau_R(\mathsf{f})}$ is the empty function, and

$$\begin{aligned}
\mathcal{P}_{\tau_R(\mathsf{f})} &= \{(p, a) \mid p \in \mathcal{P}_R \text{ and } a \in \mathsf{f}(\mathsf{box}_{\mathcal{P}_R}^{\mathsf{ext}}(p))\}\\
\mathcal{C}_{\tau_R(\mathsf{f})} &= \{(l, a) \mid l \in \mathcal{C}_R \text{ and } a \in \mathsf{f}(\mathsf{box}_{\mathcal{C}_R}^{\mathsf{ext}}(l))\}\\
\mathsf{tc}_{\tau_R(\mathsf{f})}((l, a)) &= \mathsf{tc}_R(l) \qquad \text{for every } (l, a) \in \mathcal{C}_{\tau_R(\mathsf{f})}\\
\mathsf{P}_{\tau_R(\mathsf{f})}^{\mathsf{pri}}((l, a)) &= \{(p, a) \mid p \in \mathsf{P}_R^{\mathsf{pri}}(l)\} \qquad \text{for every } (l, a) \in \mathcal{C}_{\tau_R(\mathsf{f})}\\
\mathsf{P}_{\tau_R(\mathsf{f})}^{\mathsf{aux}}((l, a)) &= \{(p, b) \mid p \in \mathsf{P}_R^{\mathsf{aux}}(l),\ a \sqsubseteq b \in \mathsf{f}(\mathsf{box}_{\mathcal{P}_R}^{\mathsf{ext}}(p))\} \quad \text{for any } (l, a) \in \mathcal{C}_{\tau_R(\mathsf{f})}
\end{aligned}$$

$$\mathsf{P}^{\mathsf{left}}_{\tau_R(\mathsf{f})}((l,a)) = (\mathsf{P}^{\mathsf{left}}_{R}(l), a) \qquad \text{for every } (l,a) \in \mathcal{C}^{\otimes,⅋}_{\tau_R(\mathsf{f})}$$
$$\mathsf{tp}_{\tau_R(\mathsf{f})}((p,a)) = \mathsf{tp}_R(p) \qquad \text{for every } (p,a) \in \mathcal{P}_{\tau_R(\mathsf{f})}$$

The *labeled Taylor expansion of* R is the set of DiLL_0-ps $\mathcal{T}_R = \mathsf{im}(\tau_R)$.

The proof that $\tau_R(\mathsf{f})$ is a DiLL_0-ps for any DiLL-ps R and any Taylor-function f of R, is left to the reader. The set $\mathcal{T}_R$ (as well as $\mathcal{T}^{\mathsf{proto}}_R$) is infinite iff $\mathsf{depth}(R) > 0$.

Note that when $l \in \mathcal{C}^{\mathsf{bord}}_R$, the condition $a \sqsubseteq b$ when defining $\mathsf{P}^{\mathsf{aux}}_{\tau_R(\mathsf{f})}((l,a))$ in Def. 11 plays a crucial role: for instance, given the MELL-ps R as in Fig. 4a and the Taylor-function f of R as in Fig. 4c, the premises of the !-cell $(l_1,(1))$ of $\tau_R(\mathsf{f})$ (whose conclusion is $(r_1,(1))$ in Fig. 4d) are $(p_1,(1,1))$, $(p_1,(1,2))$, $(p_1,(1,3))$, and not $(p_1,(2,1))$, since $(1) \not\sqsubseteq (2,1)$.

▶ Remark (Canonicity). Given a DiLL-ps R and $\mathsf{f} \in \mathcal{T}^{\mathsf{proto}}_R$, we say that f is *canonical* if

- *(horizontal downclosure)* for every $l \in \mathcal{C}^{\mathsf{box}}_R$, if $(n_1,\dots,n_m) \in \mathsf{f}(l)$ then $n_1,\dots,n_m \in \mathbb{N}^+$ and $(n_1,\dots,n_{m-1},k) \in \mathsf{f}(l)$ for any $1 \le k \le n_m$.

A $\rho \in \mathcal{T}_R$ is *canonical* if $\rho = \tau_R(\mathsf{f})$ for some canonical $\mathsf{f} \in \mathcal{T}^{\mathsf{proto}}_R$. In any canonical DiLL_0-ps of $\mathcal{T}_R$ the various copies of the content of a box are numbered sequentially starting from 1. It can easily be shown that for any $\rho \in \mathcal{T}_R$, there is a canonical $\sigma \in \mathcal{T}_R$ such that $\rho \simeq \sigma$.

The next example shows how to compute an element ρ of the labeled Taylor expansion of a DiLL-ps R starting from R and a Taylor-function of R. It shows also the information encoded in ρ with respect to R: the correspondence between ports (and cells) of ρ and ports (and cells) of R, and the differentiation of the various copies in ρ of the content of a same box in R.

▶ **Example 12.** Let R be the MELL-ps as in Fig. 4a (the tree-like order on $\mathcal{C}^{\mathsf{box}}_R$ is in Fig. 4b) and f be the Taylor-function of R as in Fig. 4c. The DiLL_0-ps $\tau_R(\mathsf{f}) \in \mathcal{T}_R$ obtained from f by applying Def. 11 is in Fig. 4d. Note that the ports $(p_2,(1,2))$ and $(p_2,(2,1))$ are two ports of $\tau_R(\mathsf{f})$ corresponding to the port p_2 of R: more precisely, $(p_2,(1,2))$ (resp. $(p_2,(2,1))$) is in the second (resp. first) copy of the content of the box of l_1 which is in the first (resp. second) copy of the content of o. Analogously for the other ports and cells of $\tau_R(\mathsf{f})$.

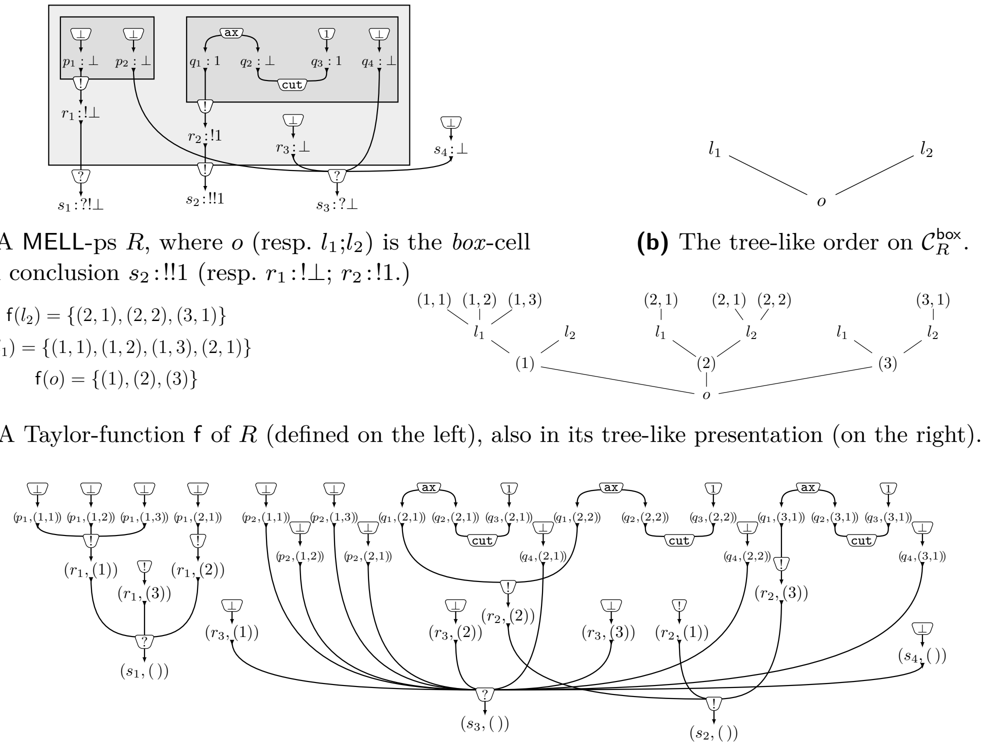
▶ **Definition 13** (Forgetful functions). Let $R \in \mathbf{PS}_{\mathsf{DiLL}}$ and $\rho \in \mathcal{T}_R$. The *forgetful functions* $\mathsf{forget}^{\rho,R}_{\mathcal{P}} \colon \mathcal{P}_\rho \to \mathcal{P}_R$ and $\mathsf{forget}^{\rho,R}_{\mathcal{C}} \colon \mathcal{C}_\rho \to \mathcal{C}_R$ are defined by: $\mathsf{forget}^{\rho,R}_{\mathcal{P}}((p,a)) = p$ and $\mathsf{forget}^{\rho,R}_{\mathcal{C}}((l,b)) = l$ for all $(p,a) \in \mathcal{P}_\rho$ and $(l,b) \in \mathcal{C}_\rho$.

By forgetting the indexes associated with the ports and cells of $\rho \in \mathcal{T}_R$, the functions $\mathsf{forget}^{\rho,R}_{\mathcal{P}}$ and $\mathsf{forget}^{\rho,R}_{\mathcal{C}}$ make explicit the correspondence (neither injective nor surjective) between ports and cells of ρ and ports and cells of R, implicitly given in Def. 11.

Given $\mathsf{f} \in \mathcal{T}^{\mathsf{proto}}_R$ such that $\rho = \tau_R(\mathsf{f}) \in \mathcal{T}_R$, the functions $\mathsf{f} \circ \mathsf{box}^{\mathsf{ext}}_{\mathcal{P}_R}$ and $\mathsf{f} \circ \mathsf{box}^{\mathsf{ext}}_{\mathcal{C}_R}$ are some kind of "inverses" of $\mathsf{forget}^{\rho,R}_{\mathcal{P}}$ and $\mathsf{forget}^{\rho,R}_{\mathcal{C}}$, respectively: with every port and cell of R, they associate the set of indexes of their corresponding ports and cells of ρ. In other words, for every port p and cell l of R, $\mathsf{f}(\mathsf{box}^{\mathsf{ext}}_{\mathcal{P}_R}(p))$ and $\mathsf{f}(\mathsf{box}^{\mathsf{ext}}_{\mathcal{C}_R}(l))$ are the sets of the "tracking numbers" of the copies of (the content of the boxes containing) p and l in ρ.

▶ **Example 14.** Let R be the MELL-ps as in Figure 5a and let f and g be the Taylor-functions of R defined in Fig. 5b-5c. Obviously, $\mathsf{f} \neq \mathsf{g}$, $\tau_R(\mathsf{f}) \neq \tau_R(\mathsf{g})$ (indeed, $(p,(2,2)) \in \mathcal{P}_{\tau_R(\mathsf{f})} \smallsetminus \mathcal{P}_{\tau_R(\mathsf{g})}$, see Fig. 5d-5e) but $\tau_R(\mathsf{f}) \simeq \tau_R(\mathsf{g})$ (and $\tau_R(\mathsf{f}), \tau_R(\mathsf{g}) \in \mathcal{T}_R$).

For any DiLL-ps R, it can be shown that the function τ_R is injective. However, Example 14 shows that there may exist two different Taylor-functions of R whose images via τ_R are different but isomorphic: the labeled Taylor expansion of a DiLL-ps may contain several elements which are isomorphic and differ from each other only by the name of their ports and



(a) A MELL-ps R, where o (resp. l_1;l_2) is the *box*-cell with conclusion $s_2 : !!1$ (resp. $r_1 : !\perp$; $r_2 : !1$.)

(b) The tree-like order on $\mathcal{C}_R^{\mathsf{box}}$.

$$\mathsf{f}(l_2) = \{(2,1),(2,2),(3,1)\}$$
$$\mathsf{f}(l_1) = \{(1,1),(1,2),(1,3),(2,1)\}$$
$$\mathsf{f}(o) = \{(1),(2),(3)\}$$

(c) A Taylor-function f of R (defined on the left), also in its tree-like presentation (on the right).

(d) The DiLL_0-ps $\tau_R(\mathsf{f}) \in \mathcal{T}_R$ (the types of the ports are omitted).

Figure 4 From a MELL-ps R (Fig. 4a) to an element of the labeled Taylor expansion of R (Fig. 4d), via a Taylor-function of R (Fig. 4c). See also Example 12.

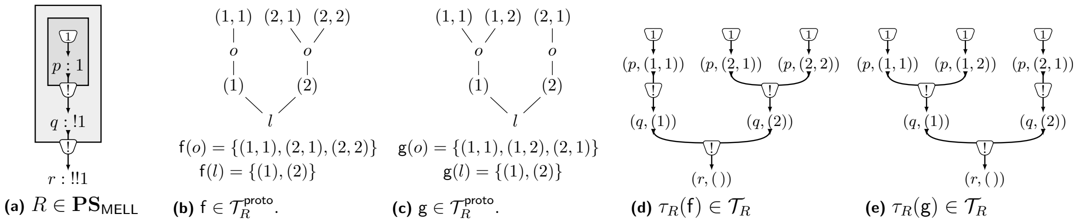
cells. Moreover, the Taylor expansion is not closed by isomorphism: from $\rho \in \mathcal{T}_R$ for some DiLL-ps R and $\sigma \simeq \rho$, it does not follow that $\sigma \in \mathcal{T}_R$ (and there might even exist a DiLL-ps $S \not\simeq R$ with $\sigma \in \mathcal{T}_S$). Indeed, although ρ and σ are isomorphic as DiLL_0-ps, all information about R available in ρ thanks to the names of its ports and cells might very well be lost in σ.

The definition of Taylor expansion of a MELL-ps coming from [9] and used in [15, Def. 9] and [17, Def. 5] forgets all the information encoded in the names of ports and cells of each element of our labeled Taylor expansion.

▶ **Definition 15** (Taylor expansion of a DiLL-proof-structure). Let R be a DiLL-ps. The *Taylor expansion of* R is $\mathcal{T}_R^{\simeq} = \big\{\{\tau \in \mathbf{PS}_{\mathsf{DiLL}_0} \mid \tau \simeq \rho\} \mid \rho \in \mathcal{T}_R\big\}$.

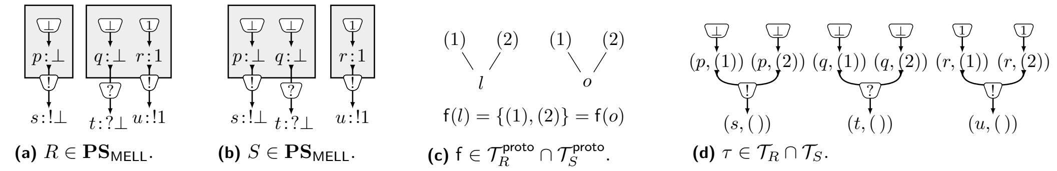
Let R be a DiLL-ps: the binary relation $\approx_R$ on $\mathbf{PS}_{\mathsf{DiLL}_0}$ defined by "$\tau \approx_R \tau'$ iff there is $\rho \in \mathcal{T}_R$ such that $\tau \simeq \rho \simeq \tau'$" is a partial equivalence relation, and, for any $\rho \in \mathcal{T}_R$, $\{\tau \in \mathbf{PS}_{\mathsf{DiLL}_0} \mid \tau \simeq \rho\}$ is a partial equivalence class on $\mathbf{PS}_{\mathsf{DiLL}_0}$ modulo $\approx_R$. Morally, $\mathcal{T}_R^{\simeq}$ is the quotient of $\mathcal{T}_R$ modulo isomorphism, *i.e.* modulo renaming of ports and cells of each element of $\mathcal{T}_R$: any element of $\mathcal{T}_R^{\simeq}$ can be seen as an element of $\mathcal{T}_R$ where all the information encoded in the names of its ports and cells is forgotten. Clearly, if $R \simeq S$ then $\mathcal{T}_R^{\simeq} = \mathcal{T}_S^{\simeq}$.

Let us stress the differences between $\mathcal{T}_R$ and $\mathcal{T}_R^{\simeq}$ of a DiLL-ps R. Given a (co-)contraction cell l of $\rho \in \mathcal{T}_R$ (*i.e.* $l \in \mathcal{C}_\rho^{!,?}$ and $\mathsf{card}(\mathsf{P}_\rho^{\mathsf{aux}}(l)) \geq 2$), it is possible to distinguish if l is a "real" (co-)contraction (*i.e.* the corresponding !- or ?-cell l' of R has at least 2 premises) or not (and then l' is in the border of some box and has only one premise which is in $\mathcal{D}oors_R^! \cup \mathcal{D}oors_R^?$): only in the first case there are two premises (p,a) and (q,b) of l with $p \neq q$. We can make



(a) $R \in \mathbf{PS}_{\mathsf{MELL}}$ **(b)** $\mathsf{f} \in \mathcal{T}_R^{\mathsf{proto}}$. **(c)** $\mathsf{g} \in \mathcal{T}_R^{\mathsf{proto}}$. **(d)** $\tau_R(\mathsf{f}) \in \mathcal{T}_R$ **(e)** $\tau_R(\mathsf{g}) \in \mathcal{T}_R$

Figure 5 A MELL-ps R (Fig. 5a) where o (resp. l) is the *box*-cell with conclusion $q\,{:}\,!1$ (resp. $r\,{:}\,!!1$), and two different but isomorphic elements $\tau_R(\mathsf{f})$ (Fig. 5d) and $\tau_R(\mathsf{g})$ (Fig. 5e) of $\mathcal{T}_R$. See Example 14.



(a) $R \in \mathbf{PS}_{\mathsf{MELL}}$. **(b)** $S \in \mathbf{PS}_{\mathsf{MELL}}$. **(c)** $\mathsf{f} \in \mathcal{T}_R^{\mathsf{proto}} \cap \mathcal{T}_S^{\mathsf{proto}}$. **(d)** $\tau \in \mathcal{T}_R \cap \mathcal{T}_S$.

Figure 6 Two non-isomorphic MELL-ps R (Fig. 6a) and S (Fig. 6b), where l (resp. o) is the *box*-cell of R and S with conclusion $s\,{:}\,!\bot$ (resp. $u\,{:}\,!1$). The DiLL$_0$-ps $\tau \in \mathcal{T}_R \cap \mathcal{T}_S$ (Fig. 6d) is (the 2-diffnet of R and S) generated by the Taylor-function f of R and S (Fig. 6c), *i.e.* $\tau_R(\mathsf{f}) = \tau = \tau_S(\mathsf{f})$.

this distinction via the information encoded in the names of ports and cells of $\rho \in \mathcal{T}_R$, but in general we are not able to do that in (any representative of) an element of $\mathcal{T}_R^{\simeq}$.

Nevertheless, the information encoded in the labeled Taylor expansion of a DiLL-ps has some limitations: in general, a DiLL-ps R is not completely characterized by any $\rho \in \mathcal{T}_R$ (even if ρ is R-fat or strongly R-fat, see Def. 16 below), *i.e.* the fact that $\rho \in \mathcal{T}_R \cap \mathcal{T}_S$ for some DiLL-ps R and S does not imply $R \simeq S$. For instance, the DiLL$_0$-ps τ in Fig. 6d is an element of both $\mathcal{T}_R$ and $\mathcal{T}_S$, where R and S are as in Fig. 6a and 6b, respectively.

Elements of special interest of the (labeled) Taylor expansion of a DiLL proof-structure

▶ **Definition 16** (R-fatness, k-diffnet of a DiLL-ps)**.** Let $R \in \mathbf{PS}_{\mathsf{DiLL}}$, $\rho \in \mathcal{T}_R$ and $k \in \mathbb{N}$.

- ρ is *R-fat* (resp. *strongly R-fat*) if, for every $(l, b) \in \mathcal{C}_\rho^!$ such that $l \in \mathcal{C}_R^{\mathsf{box}}$, one has $\mathsf{card}(\mathsf{P}_\rho^{\mathsf{aux}}((l, b))) \geq 1$ (resp. $\mathsf{card}(\mathsf{P}_\rho^{\mathsf{aux}}((l, b))) \geq 2$).
- ρ is a *k-diffnet of R* if $\mathsf{card}(\mathsf{P}_\rho^{\mathsf{aux}}((l, b))) = k$ for any $(l, b) \in \mathcal{C}_\rho^!$ such that $l \in \mathcal{C}_R^{\mathsf{box}}$.
- The *element of order k of $\mathcal{T}_R^{\simeq}$* is the $\rho_0 \in \mathcal{T}_R^{\simeq}$ such that $\rho \in \rho_0$ for some k-diffnet ρ of R.

Given a DiLL-ps R and $\rho \in \mathcal{T}_R$: ρ is R-fat (resp. strongly R-fat) when ρ is obtained by taking at least one (resp. two) copies of the content of any box in R; ρ is a k-diffnet of R when ρ is obtained by taking exactly k copies of the content of every box in R . Any k-diffnet of R with $k \geq 1$ (resp. $k \geq 2$) is R-fat (resp. strongly R-fat). Given $k \in \mathbb{N}$, all k-diffnets of R are isomorphic and there is a unique canonical k-diffnet of R; moreover, there is a unique element of order k in $\mathcal{T}_R^{\simeq}$: the set of all DiLL$_0$-ps isomorphic to any k-diffnet of R. Following [5, Def. 16–17], it can be shown that the LPS of R is univocally determined by any R-fat $\rho \in \mathcal{T}_R$.

▶ **Fact 17** (Isomorphism of gs)**.** *Let $R, S \in \mathbf{PS}_{\mathsf{DiLL}}$ and ρ (resp. σ) be a 1-diffnet of R (resp. S).*

1. *The functions $\mathsf{forget}_{\mathcal{P}}^{\rho,R}$ and $\mathsf{forget}_{\mathcal{C}}^{\rho,R}$ are bijective, and $(\mathsf{forget}_{\mathcal{P}}^{\rho,R}, \mathsf{forget}_{\mathcal{C}}^{\rho,R})\colon G_\rho \simeq G_R$.*
2. *Suppose $\varphi_1\colon \rho \simeq \sigma$. Let $\varphi_{\mathcal{P}}\colon \mathcal{P}_R \to \mathcal{P}_S$ and $\varphi_{\mathcal{C}}\colon \mathcal{C}_R \to \mathcal{C}_S$ be functions defined by $\varphi_{\mathcal{P}} = \mathsf{forget}_{\mathcal{P}}^{\sigma,S} \circ \varphi_{1\mathcal{P}} \circ (\mathsf{forget}_{\mathcal{P}}^{\rho,R})^{-1}$ and $\varphi_{\mathcal{C}} = \mathsf{forget}_{\mathcal{C}}^{\sigma,S} \circ \varphi_{1\mathcal{C}} \circ (\mathsf{forget}_{\mathcal{C}}^{\rho,R})^{-1}$.
   Then, $\varphi_{\mathcal{P}}$ and $\varphi_{\mathcal{C}}$ are bijective and $(\varphi_{\mathcal{P}}, \varphi_{\mathcal{C}})\colon G_R \simeq G_S$.*

The fact that $\rho \in \mathcal{T}_R$ for some DiLL-ps R and $\sigma \simeq \rho$ do not imply that $\sigma \in \mathcal{T}_R$ (and there may exist a DiLL-ps $S \not\simeq R$ such that $\sigma \in \mathcal{T}_S$).

Indeed, all the information about R encoded in the names of ports and cells of ρ is lost in σ, since σ is "the same as ρ up to the names of ports and cells". In general looking at σ one is not able to recognize where the border of the boxes in R are. Fact 17.2 only says that if R, S are DiLL-ps and ρ (resp. σ) is the 1-diffnet of R (resp. S) with $\varphi_1 \colon \rho \simeq \sigma$, then φ_1 induces an isomorphism φ from the gs G_R of R to the gs G_S of S, but in general φ does not make the diagram in Fig. 2b (Def. 4) commute. This is not surprising, since a 1-diffnet of a DiLL-ps R is essentially the gs of R (Fact 17.1), *i.e.* R having forgotten the border of boxes in R.

4 Connected case: computing a MELL-ps from its Taylor expansion

We show here our main result (Thm. 23): a connected (in the sense of Def. 19) MELL-ps R is completely characterized by any $\gamma \in \mathcal{T}_R^{\simeq}$ strongly fat.[6] The idea is that, by means of the "geometry" of γ (the same in all elements of γ, since they are isomorphic), we can recover the information about R encoded in the names of ports and cells of some *suitable* $\rho \in \mathcal{T}_R \cap \gamma$: in particular, we can identify the "real" contraction cells from the "fake" ones. A key-tool for this approach is the notion of ?-accessibility (Def. 18): it allows to separate the different copies of the content of a box, so it plays at a syntactic level the same role played by bridges in [5, Def. 73]. Intuitively, in a pps Φ, q is a ? -accessible port from p if there is a path in $\mathfrak{G}(G_\Phi)$ seen as undirected graph (see page 5) starting upward from p and ending in q, paying attention that, when crossing downward a cell l with type ? (here "upward" and "downward" are in the sense of the order relation $\leq_\Phi$ of Def. 2), we require that all the premises of l are reachable by a path starting upward from p.

▶ **Definition 18** (?-path, ?-accessibility)**.** Let Φ be a pps. A *?-path on Φ (from p_0 to p_n)* is a finite sequence $(p_0, \dots, p_n)$ of ports of Φ defined by induction as follows:

(i) (p) is a ?-path for any $p \in \mathcal{P}_\Phi$;

(ii) if $\vec{p} = (p_0, \dots, p_n)$ is a ?-path where $p_n \in \mathsf{P}^{\mathsf{pri}}_\Phi(l)$ for some $l \in \mathcal{C}_\Phi$, then $\vec{p}\cdot q$ is a ?-path, for any $q \in (\mathsf{P}^{\mathsf{pri}}_\Phi(l) \cup \mathsf{P}^{\mathsf{aux}}_\Phi(l)) \smallsetminus \{p_n\}$;

(iii) if $\vec{p} = (p_0, \dots, p_n)$ is a ?-path with $p_n \in \mathsf{P}^{\mathsf{aux}}_\Phi(l) \smallsetminus \{p_0\}$ for some $l \in \mathcal{C}_\Phi$ such that $\mathsf{tc}_\Phi(l) \neq ?$, then $\vec{p}\cdot q$ is a ?-path, for any $q \in (\mathsf{P}^{\mathsf{pri}}_\Phi(l) \cup \mathsf{P}^{\mathsf{aux}}_\Phi(l)) \smallsetminus \{p_n\}$;

(iv) if $\vec{p} = (p_0, \dots, p_n)$ is a ?-path with $p_n \in \mathsf{P}^{\mathsf{aux}}_\Phi(l) \smallsetminus \{p_0\}$ for some $l \in \mathcal{C}^?_\Phi$, if for any $r \in \mathsf{P}^{\mathsf{aux}}_\Phi(l)$ there is a ?-path from p_0 to r, then $\vec{p}q$ is a ?-path, for any $q \in (\mathsf{P}^{\mathsf{pri}}_\Phi(l) \cup \mathsf{P}^{\mathsf{aux}}_\Phi(l)) \smallsetminus \{p_n\}$.

For every $p \in \mathcal{P}_\Phi$, the set of the *?-accessible ports from p in Φ* is defined as $\mathsf{acces}^?_\Phi(p) := \{q \in \mathcal{P}_\Phi \mid \text{there is a ?-path in } \Phi \text{ from } p \text{ to } q\}$.

We require $p_0 \neq p_n$ in rules 3-4 so that in any ?-path on Φ of the form $p\cdot\vec{r}\cdot p\cdot q$, either $p <^1_\Phi q$, or p and q are the two conclusions of a same ax-cell (?-paths "start upwards").

According to Def. 18, given a pps Φ and $p \in \mathcal{P}_\Phi$, the set of ?-accessible ports from p in Φ

- is upward-closed (rule 2): if $q \in \mathsf{acces}^?_\Phi(p)$ and $q \leq_R q'$ then $q' \in \mathsf{acces}^?_\Phi(p)$;
- is "often" downward-closed (rules 3-4): if $q \in \mathsf{acces}^?_\Phi(p)$ and $q' \notin \mathsf{acces}^?_\Phi(p)$ with $q \in \mathsf{P}^{\mathsf{aux}}_\Phi(l)$ and $q' \in \mathsf{P}^{\mathsf{pri}}_\Phi(l)$ for some $l \in \mathcal{C}_\Phi$, then $p \in \mathsf{P}^{\mathsf{aux}}_\Phi(l)$, or $l \in \mathcal{C}^?_\Phi$ and $\mathsf{P}^{\mathsf{aux}}_\Phi(l) \not\subseteq \mathsf{acces}^?_\Phi(l)$;
- crosses ax-cells and cut-cells (rules 2-3): if $l \in \mathcal{C}^{ax}_\Phi$ then either $\mathsf{P}^{\mathsf{pri}}_\Phi(l) \subseteq \mathsf{acces}^?_\Phi(p)$ or $\mathsf{P}^{\mathsf{pri}}_\Phi(l) \cap \mathsf{acces}^?_\Phi(p) = \emptyset$; if $l \in \mathcal{C}^{cut}_\Phi$ then either $\mathsf{P}^{\mathsf{aux}}_\Phi(l) \subseteq \mathsf{acces}^?_\Phi(p)$ or $\mathsf{P}^{\mathsf{aux}}_\Phi(l) \cap \mathsf{acces}^?_\Phi(p) = \emptyset$.

[6] According to Def. 1, (strong) fatness is not defined for a set of pps, but this notion can be extended to a set of isomorphic pps thanks to Remark 2.1.

The set $\mathcal{C}_{\Phi}^{\mathsf{box}}$ and the partial function box_{Φ} play no role in Def. 18: in other words, ?-paths and ?-accessibility can be equivalently defined in the gs G_{Φ} of Φ.

Note that ?-accessibility cannot be defined as a binary symmetric relation on the ports of a pps Φ: in general, $q \in \mathsf{acces}^{?}_{\Phi}(p)$ does not imply that $p \in \mathsf{acces}^{?}_{\Phi}(q)$, as exemplified by the MELL-ps S in Fig. 8c.

▶ Remark. Recalling Remark 2.2, one can easily see that, if Φ and Ψ are pps such that $\varphi\colon G_{\Phi} \simeq G_{\Psi}$, then for every $p \in \mathcal{P}_{\Phi}$ one has: $\overline{\varphi_{\mathcal{P}}}(\mathsf{acces}^{?}_{\Phi}(p)) = \mathsf{acces}^{?}_{\Psi}(\varphi_{\mathcal{P}}(p))$.

We now define the geometric key-notion of box-connectedness: a DiLL-ps is box-connected if, seen as an undirected graph, what is *inside* any box is recursively connected, that is (following [19, 5]), for any two ports p and q on the border of a same box, p and q are connected by a path crossing only ports with depth at least the depth of p (and q). Formally, our definition relies instead on ?-paths, which are a tool used in the proof of Lemma 20.

▶ **Definition 19** (?-path inside a box, box-connectedness). Given $R \in \mathbf{PS}_{\mathsf{DiLL}}$ and $l \in \mathcal{C}_{R}^{\mathsf{box}}$, a ?-path $\vec{p} = (p_0, \dots, p_n)$ in R is *inside the box of* l if $p_i \in \mathsf{inbox}_R(l)$ for all $0 \leq i \leq n$.

A DiLL-ps R is *box-connected* if, for any $l \in \mathcal{C}_{R}^{\mathsf{box}}$ and $p \in \mathsf{inbox}_R(l)$, there is a ?-path in R from $\mathsf{prid}_R(l)$ to p inside the box of l.

For example, the DiLL-ps R_1 and R_2 in Fig. 3b-3c, and R and S in Fig. 7a-7b are box-connected; the DiLL-ps R and S in Fig. 4a and 3a are not box-connected. Clearly, any DiLL_0-ps, or more generally, any DiLL-ps R such that $\mathcal{D}oors^{?}_{R} = \emptyset = \mathcal{D}oors^{cut}_{R}$, is box-connected.

We stress that the box-connectedness condition (a crucial hypothesis in our main result) is quite general and not *ad hoc*. Indeed, it can be proven that: any ACC[7] DiLL-ps having neither $\bot$-cells nor weakenings (*i.e.* ?-cells with no premises) inside boxes is box-connected. In particular, any derivation in MELL sequent calculus without mix-rules, nor $\bot$-rules nor weakening rules corresponds to a box-connected MELL-ps. Also, any MELL-ps which is the translation of an untyped λ-term (according to the call-by-name type identity $o = {!o} \multimap o$) is box-connected. Finally, box-connectedness is preserved under cut-elimination.

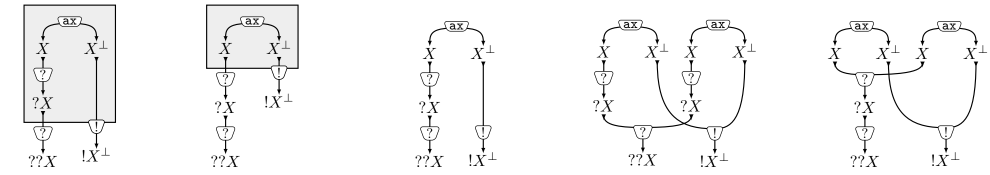
Box-connection and Taylor expansion. Given a box-connected DiLL-ps R and a strongly R-fat $\rho \in \mathcal{T}_R$, all information encoded in the indexes of ports and cells of ρ can be recovered in a " geometric" way via ?-accessibility, without looking at the names of ports and cells of ρ: by Lemma 20, in ρ the copy with index a of the content of the box associated with a *box*-cell l of R is exactly the set of ?-accessible ports from the premise $(\mathsf{prid}_R(l), a)$ of the !-cell (l, a^{-}) of ρ.

▶ **Lemma 20** (Geometric characterization of the copies of a box in an element of the labeled Taylor expansion). *Let R be a* DiLL*-ps, $\rho \in \mathcal{T}_R$ and $(p, a) \in \mathcal{P}_{\rho}$ with $p = \mathsf{prid}_R(l)$ for some $l \in \mathcal{C}_{R}^{\mathsf{box}}$.*[8] *Let $\mathcal{P}_{\rho}^{l,a} = \{(q, a\cdot b) \in \mathcal{P}_{\rho} \mid b \in \mathbb{N}^{*}$ and $q \in \mathsf{inbox}_R(l)\}$. If R is box-connected and ρ is strongly R-fat , then $\mathcal{P}_{\rho}^{l,a} = \mathsf{acces}^{?}_{\rho}((p, a))$ and thus $\mathsf{inbox}_R(l) = \overline{\mathsf{forget}^{\rho,R}_{\mathcal{P}}}(\mathsf{acces}^{?}_{\rho}((p, a)))$.*

In the proof of Lemma 20, the hypothesis of box-connectedness (resp. strong R-fatness) ensures that the ?-accessible ports from $(\mathsf{prid}_R(l), a)$ in ρ contain at least (resp. at most) all the content of the copy with index a of the content of the box associated with the *box*-cell l of R. In Fig. 6, τ is a 2-diffnet of both R and S (so τ is strongly R- and S-fat) but R and S are not box-connected, and indeed (setting $\mathcal{A}^{1}_{p} = \mathsf{acces}^{?}_{\tau}((p, (1)))$ and $\mathcal{A}^{1}_{r} = \mathsf{acces}^{?}_{\tau}((r, (1)))$):

[7] See [19, Def. A.6, Rmk. A.7] for the definition of ACC for MELL-ps, which can easily be adapted to DiLL-ps: ?-cells (resp. !-cells which are not *box*-cells) are considered as generalized ⅋-cells (resp. ⊗-cells).

[8] This implies that $(l, a^{-}) \in \mathcal{C}^{!}_{\rho}$ and $(p, a) \in \mathsf{P}^{\mathsf{aux}}_{\rho}((l, a^{-}))$, according to Def. 9–11.



(a) $R \in \mathbf{PS}_{\mathsf{MELL}}$ **(b)** $S \in \mathbf{PS}_{\mathsf{MELL}}$ **(c)** $\tau_1 \in \mathcal{T}_R^{\simeq} \cap \mathcal{T}_S^{\simeq}$ **(d)** $\rho_2 \in \mathcal{T}_R^{\simeq} \smallsetminus \mathcal{T}_S^{\simeq}$ **(e)** $\sigma_2 \in \mathcal{T}_S^{\simeq} \smallsetminus \mathcal{T}_R^{\simeq}$

Figure 7 Two non-isomorphic box-connected MELL-ps R (Fig. 7a) and S (Fig. 7b), having in their respective Taylor expansions $\mathcal{T}_R^{\simeq}$ and $\mathcal{T}_S^{\simeq}$ the same element τ_1 of order 1 (Fig. 7c), but two different elements ρ_2 (Fig. 7d) and σ_2 (Fig. 7e) of order 2, respectively.

- in R (Fig. 6a), one has $\mathsf{inbox}_R(l) = \overline{\mathsf{forget}_{\mathcal{P}}^{\tau,R}}(\mathcal{A}_p^1)$ but $\mathsf{inbox}_R(o) \not\subseteq \overline{\mathsf{forget}_{\mathcal{P}}^{\tau,R}}(\mathcal{A}_r^1)$;
- in S (Fig. 6b), one has $\mathsf{inbox}_S(o) = \overline{\mathsf{forget}_{\mathcal{P}}^{\tau,S}}(\mathcal{A}_r^1)$ but $\mathsf{inbox}_R(l) \not\subseteq \overline{\mathsf{forget}_{\mathcal{P}}^{\tau,S}}(\mathcal{A}_p^1)$.

In Fig. 7, (any representative of) τ_1 (Fig. 7c) is a 1-diffnet of S (hence τ_1 is not strongly S-fat) and the ?-accessible ports from the premise of the !-cell of τ_1 cover *more* than the content of the box of *box*-cell of S: only in σ_2 (Fig. 7e), taking two copies of the content of the box, the ?-accessible ports correspond exactly to the content of the box.

A consequence of Lemma 20 and Remark 4 is Cor. 21 given two box-connected MELL-ps R and S, and $\rho \in \mathcal{T}_R$ and $\sigma \in \mathcal{T}_S$ strongly fat, any isomorphism φ between ρ and σ "preserves" the copies of the content of a box (Cor. 21.1) and the depth of ports and cells (Cor. 21.2).

▶ **Corollary 21** (Boxes and copies preservation). *Let $R, S \in \mathbf{PS}_{\mathsf{MELL}}$, $\rho \in \mathcal{T}_R$ and $\sigma \in \mathcal{T}_S$ with $\varphi = (\varphi_{\mathcal{P}}, \varphi_{\mathcal{C}})\colon \rho \simeq \sigma$. If R and S are box-connected and ρ and σ are strongly fat, then for any $(p,a), (p',a') \in \mathcal{P}_\rho$ and $(q,b), (q',b') \in \mathcal{P}_\sigma$ with $\varphi_{\mathcal{P}}((p,a)) = (q,b)$ and $\varphi_{\mathcal{P}}((p',a')) = (q',b')$:*

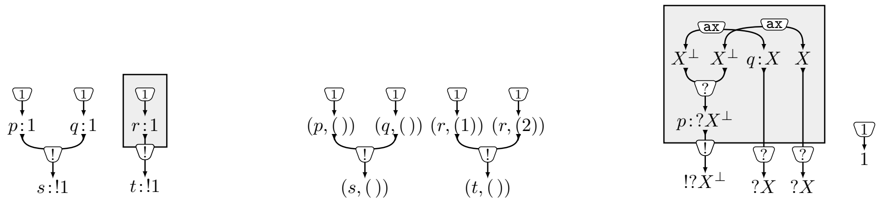
1. (copies preserv.) $\mathsf{box}_{\mathcal{P}_R}^{\mathsf{ext}}(p) = \mathsf{box}_{\mathcal{P}_R}^{\mathsf{ext}}(p')$ *and* $a = a'$ *iff* $\mathsf{box}_{\mathcal{P}_S}^{\mathsf{ext}}(q) = \mathsf{box}_{\mathcal{P}_S}^{\mathsf{ext}}(q')$ *and* $b = b'$*;*
2. (depth preserv.) $\mathsf{depth}_{\mathcal{P}_R}(p) = \mathsf{depth}_{\mathcal{P}_S}(q)$ *(and* $\mathsf{depth}_{\mathcal{P}_R}(p') = \mathsf{depth}_{\mathcal{P}_S}(q')$*).*

Cor. 21.2 says that if a port of ρ corresponds to a port of R contained in $n \in \mathbb{N}$ boxes, then its image in σ via φ corresponds to a port of S contained in n boxes, and conversely. Cor. 21.1 means that if two ports of ρ are in the same copy of the content of a box in R, then their images in σ via φ are in the same copy of a box in S , and conversely. The idea of the proof of Cor. 21.1 is that if two ports of ρ are in the same copy of a box in R, then (Lemma 20) they are ?-accessible from the same premise of a !-cell of ρ and thus, since ?-accessibility is preserved by isomorphism (Remark 4), their images via φ are ?-accessible from the same premise of a !-cell of σ, hence (Lemma 20 again) they are in the same copy of a box in S. The proof of Cor. 21.2 is similar. A fact analogous to Cor. 21 holds for cells.

▶ Remark (Box-cells preservation). Let $R, S \in \mathbf{PS}_{\mathsf{MELL}}$, $\rho \in \mathcal{T}_R$ and $\sigma \in \mathcal{T}_S$ with $\varphi = (\varphi_{\mathcal{P}}, \varphi_{\mathcal{C}})\colon \rho \simeq \sigma$. Let $a \in \mathbb{N}^*$ and $l \in \mathcal{C}_R^{\mathsf{box}}$: if $(\mathsf{prid}_R(l), a) \in \mathcal{P}_\rho$ then there are $o \in \mathcal{C}_S^{\mathsf{box}}$ and $b \in \mathbb{N}^*$ such that $\varphi_{\mathcal{P}}((\mathsf{prid}_R(l), a)) = (\mathsf{prid}_S(o), b)$ and $\varphi_{\mathcal{C}}((l, a^-)) = (o, b^-)$, as in a MELL-ps !-cells and *box*-cells coincide. Analogously for every $b \in \mathbb{N}^*$ and $o \in \mathcal{C}_S^{\mathsf{box}}$ with $(\mathsf{prid}_S(o), b) \in \mathcal{P}_\sigma$.

Remark 4 is false in general if R or S is a DiLL-ps: given $R \in \mathbf{PS}_{\mathsf{DiLL}} \smallsetminus \mathbf{PS}_{\mathsf{MELL}}$ and $\rho \in \mathcal{T}_R$ as in Fig. 8a–8b, it is easy to find $\varphi = (\varphi_{\mathcal{P}}, \varphi_{\mathcal{C}})\colon \rho \simeq \rho$ with $\varphi_{\mathcal{C}}\big((l,())\big) = (o,())$, *i.e.* φ maps the !-cell of ρ corresponding to the *box*-cell of R into the !-cell of ρ not corresponding to the *box*-cell of R. For this reason Cor. 21 holds only for MELL-ps and not for DiLL-ps, in general.

Cor. 21 (together with Fact 17) is crucial in the proof of the next lemma, which shows how to build an isomorphism ϕ between two box-connected MELL-ps R and S starting from an isomorphism φ between $\rho \in \mathcal{T}_R$ and $\sigma \in \mathcal{T}_S$ strongly fat: roughly speaking, ϕ is just the restriction of φ to only one copy (*e.g.* the first one) in ρ of the content of each box of R.



(a) $R \in \mathbf{PS}_{\mathsf{DiLL}} \smallsetminus \mathbf{PS}_{\mathsf{MELL}}$ **(b)** $\rho \in \mathcal{T}_R$ **(c)** $S \in \mathbf{PS}_{\mathsf{MELL}}$

Figure 8 A box-connected DiLL-ps R (Fig. 8a, with $\mathcal{C}_R^{\mathsf{box}} = \{l\}$ and $\mathcal{C}_R^{!} \smallsetminus \mathcal{C}_R^{\mathsf{box}} = \{o\}$) and a 2-diffnet ρ (Fig. 8b) of R (see also Remark 4). Moreover, a box-connected MELL-ps S (Fig. 8c).

▶ **Lemma 22** (Building isomorphism). *Let $R, S \in \mathbf{PS}_{\mathsf{MELL}}$, $\rho \in \mathcal{T}_R$ and $\sigma \in \mathcal{T}_S$. Suppose ρ and σ are strongly fat and canonical, and $\varphi = (\varphi_{\mathcal{P}}, \varphi_{\mathcal{C}}) \colon \rho \simeq \sigma$. Let $\phi_{\mathcal{P}} \colon \mathcal{P}_R \to \mathcal{P}_S$ and $\phi_{\mathcal{C}} \colon \mathcal{C}_R \to \mathcal{C}_S$ be functions defined in Eq. (1). If R and S are box-connected, then $\phi = (\phi_{\mathcal{P}}, \phi_{\mathcal{C}}) \colon R \simeq S$.*

$$\begin{aligned} \phi_{\mathcal{P}}(p) &= \mathsf{forget}_{\mathcal{P}}^{\sigma,S}(\varphi_{\mathcal{P}}((p,a))) \quad \text{for every } p \in \mathcal{P}_R \text{ where } (p,a) \in \mathcal{P}_\rho \text{ with } a \in \{1\}^*; \\ \phi_{\mathcal{C}}(l) &= \mathsf{forget}_{\mathcal{C}}^{\sigma,S}(\varphi_{\mathcal{C}}((l,a))) \quad \text{for every } l \in \mathcal{C}_R \text{ where } (l,a) \in \mathcal{C}_\rho \text{ with } a \in \{1\}^*. \end{aligned} \tag{1}$$

▶ **Theorem 23.** *Let R and S be some box-connected* MELL*-ps. Let $\rho_0 \in \mathcal{T}_R^{\simeq}$ and $\sigma_0 \in \mathcal{T}_S^{\simeq}$ be strongly fat. If $\rho_0 = \sigma_0$ then $R \simeq S$.*

Proof. According to Def. 15, $\rho_0 = \sigma_0$ implies that there are $\rho \in \mathcal{T}_R \cap \rho_0$, $\sigma \in \mathcal{T}_S \cap \sigma_0$ and $\varphi = (\varphi_{\mathcal{P}}, \varphi_{\mathcal{C}}) \colon \rho \simeq \sigma$. By Remark 3, we can suppose without loss of generality that ρ and σ are canonical. By hypothesis, ρ and σ are strongly fat. By Lemma 22, there is $\phi \colon R \simeq S$. ◀

We point out that Thm. 23 holds for any $\rho_0 \in \mathcal{T}_R^{\simeq}$ strongly fat, in particular when ρ_0 is the element of order 2 of the Taylor expansion of R, *i.e.* ρ_0 is obtained from R (up to isomorphism, see Def. 15-16) by taking exactly 2 copies of the content of each box in R. If R or S is not box-connected, or ρ_0 is not strongly fat, then in general $R \not\simeq S$, see Fig. 6-7.

5 Conclusion: injectivity of the relational model

Thm. 23 has a semantic counterpart: the injectivity of relational semantics for box-connected MELL-ps. The *relational model* is the simplest model of MELL; it can be seen as a degenerate case of Girard's coherent semantics [11], where formulas are interpreted as sets and proofs as relations between them. It is more or less well-known that, given a MELL-ps R, there is a correspondence between certain equivalence classes on its relational interpretation $[\![R]\!]$ and elements of its Taylor expansion $\mathcal{T}_R^{\simeq}$ (see [12] for a detailed proof): in particular, two cut-free MELL-ps with atomic axioms have the same relational semantics iff they have the same Taylor expansion. Thus, from Thm. 23 it follows that:

▶ **Corollary 24** (Injectivity for box-connected MELL). *Let R and S be cut-free* MELL*-ps with atomic axioms and conclusions of the same type.*

1. *If R and S are box-connected, and if $[\![R]\!] = [\![S]\!]$, then $R \simeq S$.*
2. *If R and S are sequentializable in* MELL *sequent calculus without* mix*-rules, $\bot$-rules and weakening-rules, and if $[\![R]\!] = [\![S]\!]$, then $R \simeq S$.*

Using different techniques, De Carvalho [3] proves the following, more general, theorem:

▶ **Theorem 25** (De Carvalho [3], injectivity for full MELL). *Let R and S be cut-free* MELL*-ps with atomic axioms and conclusions of the same type. If $[\![R]\!] = [\![S]\!]$, then $R \simeq S$.*

The injectivity proven in [5, Cor. 55] is the same as our Cor. 24.2,[9] even if the technique used in [5] allows to recover the LPS (see [5, Def. 16-17 and Cor. 52]) of any cut-free MELL-ps R with atomic axioms from its relational semantics $[\![R]\!]$: this eventually yields a slightly more general injectivity result than our Cor. 24.1.[10] As stressed in §**??**, our Thm. 23 (and our proof of Cor. 24) differs a lot from the proofs of Thm. 25 and [5, Cor. 52,54-55]: [3, 5] rely on the presence, in the interpretations of MELL-ps, of points with *arbitrarily large* complexity, depending on the two MELL-ps one wishes to discriminate. On the other hand, our result allows to discriminate any two different box-connected, cut-free MELL-ps with atomic axioms using a point of the relational semantics with *fixed* complexity (the order 2).

As a concluding remark, we believe that some kind of "converse" of Thm. 23 holds, which can be stated as follows: if R is a MELL-ps such that the element of order 2 of $\mathcal{T}_R^{\simeq}$ does not belong to $\mathcal{T}_S^{\simeq}$ for any MELL-ps $S \not\simeq R$, then R is "connected". Strictly, such a statement is wrong if we interpret "connected" as box-connected or connected graph in the sense of [5, Cor. 54]. However, we conjecture that a slight modification of these two notions yields a notion of connectedness for which Thm. 23 and its aforementioned converse (so as Cor. 24.1 and [5, Cor. 54]) hold. These results would strengthen the hierarchy outlined in §**??**.

Acknowledgements. Thanks to Daniel de Carvalho for pointing out that our approach to prove the injectivity of the relational semantics is not limited to ACC MELL-ps. Proof-nets are drawn using (a slight modification of) Marc Bagnol's package `pn.sty`. This work has been partially supported by the French ANR projects ANR-11-IS02-0002 (Locali) and ANR-12-JS02-0006 (CoQuaS), and the French A*MIDEX project ANR-11-IDEX-0001-02.

References

1 V. Danos and L. Regnier. The structure of multiplicatives. *Archive for Mathematical logic*, 28(3):181–203, 1989.

2 V. Danos and L. Regnier. Proof-nets and the Hilbert space. In *Advances in Linear Logic*, pages 307–328. Cambridge University Press, 1995.

3 D. de Carvalho. The relational model is injective for Multiplicative Exponential Linear Logic. Preprint available at `http://arxiv.org/abs/1502.02404`, April 2015.

4 D. de Carvalho, M. Pagani, and L. Tortora de Falco. A semantic measure of the execution time in Linear Logic. *Theoretical Computer Science*, 412(20):1884–1902, 2011.

5 D. de Carvalho and L. Tortora de Falco. The relational model is injective for multiplicative exponential linear logic (without weakening). *Ann. Pure Appl. Logic*, 163(9):1210–1236, 2012.

6 T. Ehrhard. Finiteness spaces. *Math. Struct. Comp. Science*, 15(4):615–646, 2005.

7 T. Ehrhard and L. Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1-3):1–41, 2003.

8 T. Ehrhard and L. Regnier. Differential interaction nets. *Theoretical Computer Science*, 364(2):166–195, 2006.

9 T. Ehrhard and L. Regnier. Uniformity and the Taylor expansion of ordinary lambda-terms. *Theoretical Computer Science*, 403(2-3):347–372, 2008.

10 H. Friedman. Equality between functionals. In Rohit Parikh, editor, *Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, pages 22–37. Springer Berlin, 1975.

[9] Our main results (Thm. 23, Cor. 24) can easily be generalized to untyped MELL-ps, as in [5].

[10] The injectivity proven in [5, Cor. 54] holds for a set of MELL-ps that is "similar" to the one for which our Cor. 24.1 holds: the notion of "connected graph" in [5, Cor. 54] is similar to our box-connectedness, even if, strictly speaking, neither of the two implies the other.

11 J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

12 G. Guerrieri, L. Pellissier, and L. Tortora de Falco. Syntax, Taylor expansion and relational semantics of MELL proof-structures: an unusual approach. Technical report, 2016. Available at `http://logica.uniroma3.it/~tortora/mell.pdf`.

13 Y. Lafont. From Proof-Nets to Interaction Nets. In *Advances in Linear Logic*, pages 225–247. Cambridge University Press, 1995.

14 O. Laurent. Polarized proof-nets and $\lambda\mu$-calculus. *Theor. Comp. Sci.*, 290(1):161–188, 2003.

15 D. Mazza and M. Pagani. The Separation Theorem for Differential Interaction Nets. In *Proceedings of LPAR 2007*, pages 393–407, 2007.

16 M. Pagani. The Cut-Elimination Thereom for Differential Nets with Boxes. In *Proceedings of TLCA 2009*, pages 219–233, 2009.

17 M. Pagani and C. Tasson. The Taylor Expansion Inverse Problem in Linear Logic. In *Proceedings of LICS 2009*, pages 222–231, 2009.

18 R. Statman. Completeness, invariance and λ-definability. *J. Symb. Logic*, 47(1):17–26, 1982.

19 L. Tortora de Falco. Obsessional Experiments For Linear Logic Proof-Nets. *Mathematical Structures in Computer Science*, 13(6):799–855, December 2003.

20 P. Tranquilli. Intuitionistic differential nets and lambda-calculus. *Theoretical Computer Science*, 412(20):1979–1997, 2011.

Strongly Normalising Cyclic Data Computation by Iteration Categories of Second-Order Algebraic Theories

Makoto Hamana

Department of Computer Science, Gunma University, Kiryu, Japan
hamana@cs.gunma-u.ac.jp

Abstract

Cyclic data structures, such as cyclic lists, in functional programming are tricky to handle because of their cyclicity. This paper presents an investigation of categorical, algebraic, and computational foundations of cyclic datatypes. Our framework of cyclic datatypes is based on second-order algebraic theories of Fiore et al., which give a uniform setting for syntax, types, and computation rules for describing and reasoning about cyclic datatypes. We extract the "fold" computation rules from the categorical semantics based on iteration categories of Bloom and Esik. Thereby, the rules are correct by construction. Finally, we prove strong normalisation using the General Schema criterion for second-order computation rules. Rather than the fixed point law, we particularly choose Bekič law for computation, which is a key to obtaining strong normalisation.

1998 ACM Subject Classification D.3.2 Language Classifications, E.1 Data Structures, F.3.2 Semantics of Programming Languages

Keywords and phrases cyclic data structures, traced cartesian category, fixed point, functional programming, fold

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.21

1 Introduction

Cyclic data structures in functional programming are tricky to handle because of their cyclicity. In Haskell, one can define cyclic data structures, such as cyclic lists by

```
clist = 2:1:clist
```

The feasibility of such a recursive definition of cyclic data depends on lazy evaluation. However, it does not ensure termination of computation. It might fall into a non-terminating situation. For example, what is the sum of all elements of `clist`? One may think that it is non-terminating, undefined, or impossible.

An answer using our framework in this paper is different. We do not rely on lazy evaluation. We provide a way to regard the sum of a cyclic list as a cyclic natural number, which is computed by the strongly normalising "fold" combinator. In this paper, we investigate a framework for syntax and semantics of *cyclic datatypes* that makes this understanding and computation correct.

Our framework of cyclic datatypes is founded on second-order algebraic theories of Fiore et al. [13, 14]. Second-order algebraic theories have been shown to be a useful framework that models various important notions of programming languages, such as logic programming [32], algebraic effects [15], quantum computation [33]. This paper gives another application of second-order algebraic theories, namely, to cyclic datatypes and its computation. We use second-order algebraic theories to give a uniform setting for typed syntax, equational

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 21; pp. 21:1–21:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

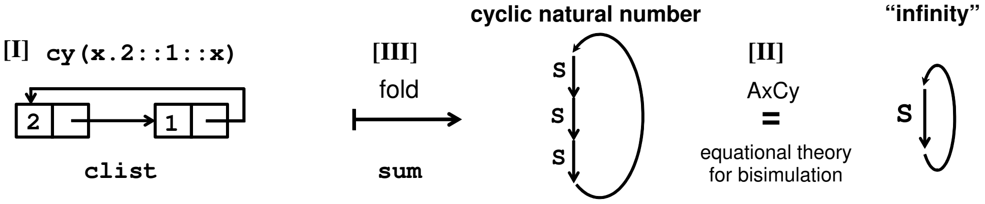


Figure 1 Framework: Second-order algebraic theories and iteration theories.

logic and computation rules for describing and reasoning about cyclic datatypes. We extract computation rules for the fold from the categorical semantics based on iteration categories [5]. Thereby the rules are correct by construction. Finally, we prove strong normalisation by using the General Schema criterion [3] for rewrite rules.

Overview. As an overview of cyclic datatypes and their operations we develop in this paper, we first demonstrate descriptions and an operation of cyclic datatypes by pseudo-program codes. The code fragments correspond one-to-one to theoretical data given in later sections. Therefore, they are theoretically meaningful and more intuitive than starting from detailed theory.

First we consider an example of cyclic lists. The code below with the keyword `ctype` is intended to declare cyclic datatype `CList` of cyclic lists having two ordinary constructors in Haskell or Agda style.

```
ctype CList where
  [] : CList
  :: : CNat,CList → CList
with axioms AxCy
```

We assume that any `ctype` declared datatype has a default constructor "`cy`" for making a cycle. For example, we express a cyclic list of 1 as a term `cy(x.1::x)`, where `cy` has a variable binding "`x.`", regarded as the "address" of the top of list. A variable occurrence `x` in the body means to refer to the top, hence it makes a cycle. The terms built from the constructors of `CList` and the default constructor `cy` is required to satisfy the axioms AxCy (given later in Fig. 3) as the keyword "`with axioms`" mentioned (we assume that any `ctype` datatype satisfies AxCy, so this is for ease of understanding). We next consider the above mentioned example of the sum of cyclic list. We define another cyclic datatype of natural numbers.

```
ctype CNat where
  0 : CNat
  S : CNat → CNat
with axioms AxCy
```

```
sum : CList → CNat
spec sum ([])    = 0
     sum (k::t) = plus(k, sum (t))
```

The code with keyword `spec` at the right column describes an equational *specification* of function. It requires that the sum function from cyclic lists to cyclic natural numbers must satisfy the ordinary definition. We intend that the `spec` code is merely a (loose) specification, and not a definition, because it lacks the case of `cy`-term.

```
fun sum t = fold (0, k,x.plus(k,x)) t
```

We here assume that the `plus` function on `CNat` has already been defined (as presented later in Example 4.2). The above code with the keyword `fun` defines the function `sum`. It is defined by the `fold` combinator on the cyclic datatypes, as in an ordinary fold on an algebraic datatypes. The first two arguments 0 and `k,x.plus(k,x)` correspond to the right-hand sides of the specification of `sum`. The `fold` is actually the fold on a cyclic datatype, which knows how to cope with `cy`-term. Actually, the sum of a cyclic list can be computed as follows:

$$\begin{aligned}&\texttt{sum(cy(x.S}^2\texttt{(0)::S(0)::x))} \to\\ &\qquad\qquad\qquad\texttt{cy(x.sum(x.S}^2\texttt{(0)::S(0)::x)@x)} \to^+ \texttt{cy(x.S(S(S(x))))}\end{aligned}$$

where we represent n by $\texttt{S}^n(0)$. The final term is a normal form that cannot be rewritten further. Therefore, we regard it as the computation result. Here `sum` is intended to denote "`fold`(0,..)". The steps presented above are actual rewrite steps by the second-order rewrite rules FOLDr given later in Fig. 8.

How to understand the meaning of the result `cy(x.S(S(S(x))))` is arguable. The overall situation we have demonstrated is illustrated in Fig. 1. In this paper, we also provide a formal basis to understand and to reason about cyclic data, as well as the computation result. We use second-order equational logic and the axioms AxCy to equate cyclic data formally (Fig. 1 [III]). It completely characterises the notion of bisimulation on cyclic data. The expression `cy(x.S(S(S(x))))` is equal to (i.e. bisimilar to) `cy(x.S(x))`, which is a minimal representation of the result, which may be regarded as ∞ (infinity). In this paper, we do not develop an explicit algorithm to extract such a minimal representation from the computation result, but it is noteworthy that this equational theory generated by AxCy is decidable. Consequently, it is computationally reasonable. More practical examples on cyclic datatypes and computation will be given in §6.

2 Second-Order Algebraic Theory of Cyclic Datatypes

We introduce the framework of second-order cartesian algebraic theory, which is a typed and cartesian extension of [13, 14] and [19]. Here "cartesian" means that the target sort of a function symbol is a sequence of base types. We use second-order algebraic theory as a formal framework to provide syntax and to describe axioms of algebraic datatypes enriched with cyclic constructs. The second-order feature is necessary for cycle operation and the fold function on them. We will often omit superscripts or subscripts of a mathematical object if they are clear from contexts. We use the vector notation $\overrightarrow{A}$ for a sequence $A_1, \cdots, A_n$, and $|\overrightarrow{A}|$ for its length.

2.1 Cartesian Second-Order Algebraic Theory

We assume that $\mathcal{B}$ is a set of *base types* (written as $a, b, c, \ldots$), and Σ, called a *signature*, is a set of function symbols of the form

$$f : (\overrightarrow{a_1} \to \overrightarrow{b_1}), \ldots, (\overrightarrow{a_m} \to \overrightarrow{b_m}) \to c_1, \ldots, c_n.$$

where all a_i, b_i, c_i are base types (thus any function symbol is of up to second-order type). A sequence of types may be empty in the above definition. The empty sequence is denoted by $()$, which may be omitted, e.g., $b_1, \ldots, b_m \to c$, or $() \to c$. The latter case is simply denoted by c. A signature Σ_c for type c denotes a subset of Σ, where every function symbol is of the form $f : \tau \to c$, which is regarded as a constructor of c. A *metavariable* is a variable of (at most) first-order type, declared as $\textsc{m} : \overrightarrow{a} \to b$ (written as small-caps letters $\textsc{z}, \textsc{t}, \textsc{s}, \textsc{m}, \ldots$). A *variable* of the order 0 type is merely called variable (written usually $x, y, \ldots$). The raw syntax is given as follows.

- *Terms* have the form: $t ::= x \mid x.t \mid f(t_1,\dots,t_n)$.
- *Meta-terms* extend terms to: $t ::= x \mid x.t \mid f(t_1,\dots,t_n) \mid \mathrm{M}[t_1,\dots,t_n]$.

Terms are used for representing concrete cyclic data, functional programs on them and equations we want to model. A second-order equational theory is a set of proved equations built from terms (NB. not meta-terms). Meta-terms are used for formulating equational axioms, which are expected to be instantiated to terms. We write $x_1,\dots,x_n.t$ for $x_1.\cdots.x_n.t$.

A metavariable context Θ is a sequence of metavariable:type-pairs, and a context Γ is a sequence of variable:type-pairs. A judgment is of the form $\Theta \triangleright \Gamma \vdash t : \overrightarrow{b}$. If Θ is empty, we may simply write $\Gamma \vdash t : \overrightarrow{b}$. A meta-term t is well-typed by the typing rules Fig. 4. We often omit the types for binders as $f(\overrightarrow{x_1}.t_1, \dots, \overrightarrow{x_n}.t_n)$. Given a meta-term t with free variables $x_1,\dots,x_n$, the notation $s\,[x_1 \mapsto s_1,\dots,x_n \mapsto s_n]$ denotes ordinary capture avoiding substitution that replaces the variables with meta-terms $s_1,\dots,s_n$. A substitution which replaces metavariables with meta-terms [14] is defined in Fig. 6. For meta-terms $\Theta \triangleright \Gamma \vdash s : \overrightarrow{b}$ and $\Theta \triangleright \Gamma \vdash t : \overrightarrow{b}$, an *equation* is of the form $\Theta \triangleright \Gamma \vdash s = t : \overrightarrow{b}$, or denoted by $\Gamma \vdash s = t : \overrightarrow{b}$ when Θ is empty. The cartesian second-order equational logic is a logic to deduce formally proved equations. The inference system of equational logic is given in Fig. 5.

Preliminaries for datatypes. The *default signature* Σ_{def} is given by the function symbols called *default constructors*:

$$\begin{array}{llll} \text{Empty sequence} & \langle\,\rangle \quad :() & \text{Tuple} & \langle -,\cdots,-\rangle : (\overrightarrow{c_1}),\dots,(\overrightarrow{c_n}) \to \overrightarrow{c_1},\dots,\overrightarrow{c_n} \\ \text{Cycle constr.} & \mathsf{cy}^{|\overrightarrow{c}|} : (\overrightarrow{c} \to \overrightarrow{c}) \to \overrightarrow{c} & \text{Composition} & \diamond_{(\overrightarrow{d},\overrightarrow{c})} : (\overrightarrow{d} \to \overrightarrow{c}), \overrightarrow{d} \to \overrightarrow{c}\,. \end{array}$$

defined for all base types $\overrightarrow{d}, \overrightarrow{c}, \overrightarrow{c_1},\dots,\overrightarrow{c_n} \in \mathcal{B}$. This means that any base type has default constructors. We assume that any signature includes Σ_{def} in this paper. We identify $\langle t, \langle\,\rangle\rangle$ and $\langle\langle\,\rangle, t\rangle$ with t, and $\langle\langle s\,,\,t\rangle\,,\,u\rangle$ with $\langle s\,,\,\langle t\,,\,u\rangle\rangle$; thus we will freely omit the angle brackets as $\langle t_1\,,\,\dots\,,\,t_n\rangle$.

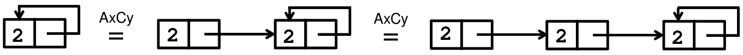
A *datatype declaration* for a type c is given by a triple $(c, \Sigma_c, \mathcal{E})$ consisting of a base type c, signature Σ and equational axioms $\mathcal{E}$, where every $f \in \Sigma_c$ is first-order, i.e. is of the form $f : b_1,\dots,b_n \to c$, and any equation in $\mathcal{E}$ is built from Σ_c-terms.

2.2 Instance (1): Cyclic Lists modulo Bisimulation

We will present an algebraic formulation of cyclic datatypes. By cyclic datatype, we mean algebraic datatype having the cycle construct cy satisfying the axioms that characterise cyclicity. The first example is the datatype of cyclic lists. It has already been defined as `CList` in Introduction as the pseudo code. We now give a formal definition using datatype declaration. Fix $a \in \mathcal{B}$. The datatype declaration for $\texttt{CList}_a$, the cyclic lists of type a, is given by $(\texttt{CList}_a, \Sigma_{\texttt{CList}_a}, \mathsf{AxCy})$ where $\Sigma_{\texttt{CList}_a}$ is

$$[\,] : \texttt{CList}_a, \qquad (- ::_a -) : a, \texttt{CList}_a \to \texttt{CList}_a$$

and the axioms AxCy are given in Fig. 3. Note that $\texttt{CList}_a$ has also the default constructors, thus one can form cycle (see the example below). The definition of `CList` in Introduction actually represents the datatype declaration $(\texttt{CList}_{\texttt{CNat}}, \Sigma_{\texttt{CList}_{\texttt{CNat}}}, \mathsf{AxCy})$. Hereafter, we will omit the type parameter subscript a of `CList`. The axioms AxCy mathematically characterise that cy is truly a cycle constructor in the sense of Conway fixed point operator [5]. The equational theory generated by AxCy captures the intended meaning of cyclic lists. For example, the following are identified as the same cyclic list:



$$\mathsf{cy}(x.2 :: x) = \quad 2 :: \mathsf{cy}(x.2 :: x) \quad = \quad 2 :: 2 :: \mathsf{cy}(x.2 :: x)\,.$$

These equalities come from the *fixed point property* of cy.

On axioms AxCy We explain the intuitive meaning of the axioms in AxCy. Parameterised fixed-point axioms axiomatise cy as a fixed point operator. They (minus (CI)) are equivalent to the axioms for Conway operators of [5, 23, 31]. Bekič law is well-known in denotational semantics (cf. [34, §10.1]) to calculate the fixed point of a pair of continuous functions. Conway operators are also arisen in work independently of Hyland and Hasegawa [23], who established a connection with the notion of traced cartesian categories [25]. There are equalities that holds in the cpo semantics but Conway operators do not satisfy. The axiom (CI) is the commutative identities of Bloom and Ésik [5, 31], which ensures that all equalities that hold in the cpo semantics do hold. See also [31, Section 2] for a useful overview about this. The equality generated by AxCy is actually bisimulation on cyclic lists. This is included in the equality of cyclic sharing trees given in next subsection.

2.3 Instance (2): Cyclic Sharing Trees modulo Bisimulation

Next we consider the datatype of binary branching trees, which can involve cycle and sharing. We call them *cyclic sharing trees*, or simply cyclic trees. We first give the declaration of datatype CTree of cyclic trees as the style of pseudo code below, where we assume that f's

```
ctype CTree where
  f : CTree → CTree
  [] : CTree
  + : CTree,CTree → CTree
with axioms AxCy,AxBr([],+)
```

part denotes various unary function symbols such as a,b,c,p,q,.... Formally, it is expressed as the datatype declaration (CTree, $\{f, [\,], +\}$, $\mathsf{AxCy} \cup \mathsf{AxBr}([\,], +)$). The binary operator + denotes a branch. For example, one can write b([])+c([]) (cf. Fig. 2 (A)). It can also express sharing by the constructor ⋄ of composition: x.a(b(x) + c(x)) ⋄ p([]) (Fig. 2 (F)). Note that the first argument of composition ⋄ has a binder (e.g. x.), which indicates placeholder filled by the shared part after ⋄ (e.g. p([])). A binder at the first argument of ⋄-term may be a sequence of variables (e.g. "x,y." in (E)), which will be filled by terms in a tuple (e.g. <p([]),q([])>). Cyclic trees are very expressive. They cover essentially XML trees with IDREF, the data model called *trees with pointers* [7], and arbitrary rooted directed graphs (cf. Fig. 2 (B)(E)).

We denote by $\sim$ the equivalence relation generated by the axioms $\mathsf{AxCy}, \mathsf{AxBr}([\,], +)$ in Fig. 3. Using the axioms $\mathsf{AxCy} \cup \mathsf{AxBr}([\,], +)$, we can reason this equality $\sim$ in the second-order equational logic. The equality $\sim$ gives reasonable meaning of cycles as in the case of cyclic lists and that the branch $+$ is associative, commutative and idempotent (cf. Fig. 2 (C)), thus nested $+$ can be seen as an n-ary branch (cf. Fig. 2 (D)). Moreover, a shared term and its unfolding are also identified by $\sim$ because of the axiom (sub) (cf. Fig. 2 (F)). The axiom (sub) is similar to the β-reduction in the λ-calculus.

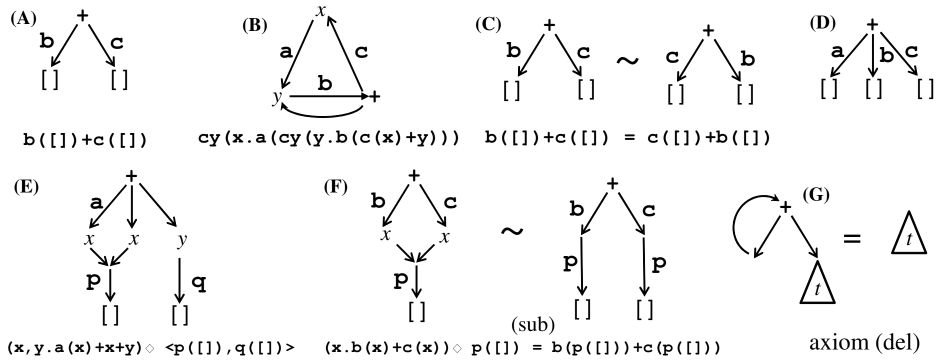


Figure 2 Examples of cyclic sharing trees.

Algebraic theory of bisimulation. Actually, $\sim$ is exactly *bisimulation* on cyclic trees. Since unary f expresses a labelled edge, and $+$ expresses a branch, cyclic sharing trees are essentially process graphs of regular behaviors, called *charts* by Milner in [29]. Infinite unfolding of them are synchronization trees [5]. Thus the standard notion of bisimulation between graphs can be defined. Intuitively, starting from the root, bisimulation is by comparing traces of labels of two graphs along edges (more detailed definition is given in [5, 29] or ([22] Appendix)). Now we see that (C),(F) and (G) are examples of bisimulation. Actually, the axioms in Fig. 3 are sound for bisimulation, i.e. for each axiom, the left and the right-hand sides are bisimilar. Moreover, it is complete.

▶ **Proposition 2.1** ([21],([22]§5.3)). *$\Gamma \vdash s = t \quad : \mathsf{CTree}$ is derivable from AxCy and $\mathsf{AxBr}([\,], +)$ iff if s and t are bisimilar.*

The main reason of this is that the axioms AxCy and $\mathsf{AxBr}([\,], +)$ are second-order representation of Bloom and Ésik's complete equational axioms of bisimulation [5]. A crucial fact is that bisimulation $s \sim t$ is decidable [5, 6]. There is also an efficient algorithm for checking bisimulation, e.g. [10]. Hence, cyclic datatypes with the axioms AxCy, or the axioms $\mathsf{AxCy} \cup \mathsf{AxBr}$ are computationally feasible, such as checking equality on cyclic structures we have seen in Fig. 1.

There are many other instances of cyclic datatypes, some of which will be given in §6.

3 Categorical Semantics of Cyclic Datatypes

In this section, we give a categorical semantics of cyclic datatypes. A reason to consider categorical semantics is to systematically obtain a "structure preserving map" on cyclic datatypes. We will formulate the "fold" function for a cyclic datatype as a functor on the category for cyclic datatypes (Thm. 3.8 and §4).

Since a cyclic datatype has cycles, the target categorical structure should have a notion of fixed point. It has been studied in iteration theories of Bloom and Ésik [5]. In particular iteration categories [11] are suitable for our purpose, which are traced cartesian categories [25, 23] satisfying the commutative identities axiom [5]. We write $\mathbf{1}$ for the terminal object, $\times$ for the cartesian product, $\langle -, - \rangle$ for pairing, and $\Delta = \langle \mathrm{id}, \mathrm{id} \rangle$ for diagonal in a cartesian category.

Axioms AxCy for cycles

$$\begin{array}{llll}
\text{(sub)} & \begin{array}{l}\textsc{t}: \overrightarrow{a} \to \overrightarrow{c},\\ \textsc{s}_1,\dots,\textsc{s}_n : \overrightarrow{a}\end{array} \rhd \vdash & (\overrightarrow{y}.\textsc{t}[\overrightarrow{y}]) \diamond \langle \textsc{s}_1 , \dots , \textsc{s}_n \rangle = \textsc{t}[\textsc{s}_1 , \dots , \textsc{s}_n] & : \overrightarrow{c} \\[2ex]
\text{(SP)} & \textsc{t} : \overrightarrow{c} \quad \rhd \vdash & \langle (\overrightarrow{y}.y_1) \diamond \textsc{t}, \dots , (\overrightarrow{y}.y_n) \diamond \textsc{t} \rangle = \textsc{t} & : \overrightarrow{c} \\[2ex]
\text{(dinat)} & \begin{array}{l}\textsc{s}: \overrightarrow{a} \to \overrightarrow{c},\\ \textsc{t}: \overrightarrow{c} \to \overrightarrow{a}\end{array} \rhd \vdash & \mathsf{cy}(\overrightarrow{x}.\textsc{s}[\textsc{t}[\overrightarrow{x}]]) = (\overrightarrow{z}.\textsc{s}[\overrightarrow{z}]) \diamond \mathsf{cy}(\overrightarrow{z}.\textsc{t}[\textsc{s}[\overrightarrow{z}]])] & : \overrightarrow{c} \\[2ex]
\text{(Beki\v{c})} & \begin{array}{l}\textsc{t} : \overrightarrow{c}, \overrightarrow{a} \to \overrightarrow{c},\\ \textsc{s} : \overrightarrow{c}, \overrightarrow{a} \to \overrightarrow{a}\end{array} \rhd \vdash & \mathsf{cy}(\overrightarrow{x}, \overrightarrow{y}. \langle \widehat{\textsc{t}}, \widehat{\textsc{s}} \rangle) = \begin{array}{l}\langle\ \mathsf{cy}(\overrightarrow{x}.(\overrightarrow{y}.\widehat{\textsc{t}}) \diamond \mathsf{cy}(\overrightarrow{y}.\widehat{\textsc{s}})),\\ \quad \mathsf{cy}(\overrightarrow{y}.(\overrightarrow{x}.\widehat{\textsc{s}}) \diamond \mathsf{cy}(\overrightarrow{x}.(\overrightarrow{y}.\widehat{\textsc{t}}) \diamond \mathsf{cy}(\overrightarrow{y}.\widehat{\textsc{s}}))) \ \rangle\end{array} & : \overrightarrow{c}, \overrightarrow{a} \\[2ex]
\text{(CI)} & \textsc{t} : \overrightarrow{a} \to \overrightarrow{a} \quad \rhd \vdash & \mathsf{cy}(\overrightarrow{y}.\langle \textsc{t}[\rho_1], \dots , \textsc{t}[\rho_m] \rangle) = \langle \mathsf{cy}(y.\tilde{\textsc{t}}\,), \dots , \mathsf{cy}(y.\tilde{\textsc{t}}\,) \rangle & : \overrightarrow{a}
\end{array}$$

In (CI), $\rho_i = \langle q_1 , \dots , q_m \rangle$, each q_j is one of y_i for $i = 1, \dots , m$, and $\tilde{\textsc{t}}$ is short for $\textsc{t}[y, \dots , y]$. In (Bekič), $\widehat{\textsc{t}}$ and $\widehat{\textsc{s}}$ are short for $\textsc{t}[\overrightarrow{x}, \overrightarrow{y}]$ and $\textsc{s}[\overrightarrow{x}, \overrightarrow{y}]$, respectively.

Axioms AxBr($[\,], +$) for branching

$$\begin{array}{lllcll}
\text{(del)} & \textsc{t} : c & \rhd \vdash & \mathsf{cy}(x^c.\textsc{t} + x) & = & \textsc{t} & : c \\
\text{(unitL)} & \textsc{t} : c & \rhd \vdash & [\,] + \textsc{t} & = & \textsc{t} & : c \\
\text{(unitR)} & \textsc{t} : c & \rhd \vdash & \textsc{t} + [\,] & = & \textsc{t} & : c \\
\text{(assoc)} & \textsc{s}, \textsc{t}, \textsc{u} : c & \rhd \vdash & (\textsc{s} + \textsc{t}) + \textsc{u} & = & \textsc{s} + (\textsc{t} + \textsc{u}) & : c \\
\text{(comm)} & \textsc{s}, \textsc{t} : c & \rhd \vdash & \textsc{s} + \textsc{t} & = & \textsc{t} + \textsc{s} & : c \\
\text{(degen)} & \textsc{t} : c & \rhd \vdash & \textsc{t} + \textsc{t} & = & \textsc{t} & : c
\end{array}$$

Figure 3 Axioms.

▶ **Definition 3.1** ([11, 5]). A *Conway operator* in a cartesian category $\mathcal{C}$ is a family of functions $(-)^\dagger : \mathcal{C}(A \times X, X) \to \mathcal{C}(A, X)$ satisfying:

$$(f \circ (g \times \mathrm{id}_X))^\dagger = f^\dagger \circ g\,, \qquad (f^\dagger)^\dagger = (f \circ (\mathrm{id}_A \times \Delta))^\dagger\,,$$
$$f \circ \langle \mathrm{id}_A, (g \circ \langle \pi_1, f \rangle)^\dagger \rangle = (f \circ \langle \pi_1, g \rangle)^\dagger.$$

An *iteration category* is a cartesian category having a Conway operator satisfying the "commutative identities" law [5]

$$\langle f \circ (\mathrm{id}_A \times \rho_1), \dots , f \circ (\mathrm{id}_A \times \rho_m) \rangle^\dagger = \Delta_m \circ (f \circ (\mathrm{id}_A \times \Delta_m))^\dagger : A \to X$$

where

- $f : A \times X^m \to X$
- diagonal $\Delta_m \triangleq \langle \mathrm{id}_X, \cdots , \mathrm{id}_X \rangle : X \to X^m$
- $\rho_i : X^m \to X^m$ such that $\rho_i = \langle q_{i1}, \dots , q_{im} \rangle$ where each q_{ij} is one of projections $\pi_1, \dots , \pi_m : X^m \to X$ for $i = 1, \dots , m$ (see also [31]).

An *iteration functor* between iteration categories is a cartesian functor that preserves Conway operators.

A typical example of iteration category is the category of complete partial orders (cpos) and continuous functions [5, 23], where the least fixed point operator is a Conway operator.

▶ **Definition 3.2.** Let Σ be a signature. A Σ-*structure* M in an iteration category $\mathcal{C}$ is specified by giving for each base type $b \in \mathcal{B}$, an object $[\![b]\!]^M$ (or simply written $[\![b]\!]$) in $\mathcal{C}$, and for each function symbol $f : (\overrightarrow{a_1} \to \overrightarrow{b_1}), \dots , (\overrightarrow{a_m} \to \overrightarrow{b_m}) \to \overrightarrow{c}$, a function

$$[\![f]\!]^M_A : \mathcal{C}(A \times [\![\overrightarrow{a_1}]\!], [\![\overrightarrow{b_1}]\!]) \times \cdots \times \mathcal{C}(A \times [\![\overrightarrow{a_n}]\!], [\![\overrightarrow{b_n}]\!]) \longrightarrow \mathcal{C}(A, [\![\overrightarrow{c}]\!]) \qquad (1)$$

$$\frac{y : b \in \Gamma}{\Theta \ \triangleright\ \Gamma \vdash\ y \ : b} \qquad \frac{(\mathrm{M} : a_1, \ldots, a_m \to \overrightarrow{b}\,) \in \Theta \quad \Theta \ \triangleright\ \Gamma \vdash\ t_1 \ : a_1 \quad \Theta \ \triangleright\ \Gamma \vdash\ t_m \ : a_m}{\Theta \ \triangleright\ \Gamma \vdash\ \mathrm{M}[t_1, \ldots, t_m] \ : \overrightarrow{b}}$$

$$\frac{\Theta \ \triangleright\ \Gamma, \overrightarrow{x_1 : a_1} \vdash\ t_1 \ : \overrightarrow{b_1} \quad \cdots \quad \Theta \ \triangleright\ \Gamma, \overrightarrow{x_m : a_m} \vdash\ t_m \ : \overrightarrow{b_m}}{\Theta \ \triangleright\ \Gamma \vdash\ f(\,\overrightarrow{x_1^{a_1}}.t_1, \ldots \overrightarrow{x_m^{a_m}}.t_m\,) \ : \overrightarrow{c}}$$

where $f : (\overrightarrow{a_1} \to \overrightarrow{b_1}), \ldots, (\overrightarrow{a_m} \to \overrightarrow{b_m}) \to \overrightarrow{c}$.

Figure 4 Typing rules of meta-terms.

(Sub)

$$\frac{\begin{array}{l} \mathrm{M}_1 : (\overrightarrow{a_1} \to \overrightarrow{b_1}), \ldots, \mathrm{M}_k : (\overrightarrow{a_k} \to \overrightarrow{b_k}) \ \triangleright\ \Gamma \vdash\ t = t' \ : \overrightarrow{c} \\ \Theta \ \triangleright\ \Gamma', \overrightarrow{x_i : a_i} \vdash\ s_i = s_i' \ : \overrightarrow{b_i} \quad (1 \le i \le k) \end{array}}{\Theta \ \triangleright\ \Gamma, \Gamma' \vdash\ t\,[\,\overrightarrow{\mathrm{M} := s}\,] = t'\,[\,\overrightarrow{\mathrm{M} := s'}\,] \ : \overrightarrow{c}}$$

(Ax) $$\frac{(\Theta \ \triangleright\ \Gamma \vdash\ s = t \ : \overrightarrow{c}\,) \in \mathcal{E}}{\Theta \ \triangleright\ \Gamma \vdash\ s = t \ : \overrightarrow{c}}$$

(Ref) $$\frac{}{\Theta \ \triangleright\ \Gamma \vdash\ t = t \ : \overrightarrow{c}}$$

(Sym) $$\frac{\Theta \ \triangleright\ \Gamma \vdash\ s = t \ : \overrightarrow{c}}{\Theta \ \triangleright\ \Gamma \vdash\ t = s \ : \overrightarrow{c}}$$

(Tr) $$\frac{\Theta \ \triangleright\ \Gamma \vdash\ s = t \ : \overrightarrow{c} \quad \Theta \ \triangleright\ \Gamma \vdash\ t = u \ : \overrightarrow{c}}{\Theta \ \triangleright\ \Gamma \vdash\ s = u \ : \overrightarrow{c}}$$

Figure 5 Cartesian second-order equational logic.

which is natural in A, where $\llbracket b_1, \ldots, b_n \rrbracket \triangleq \llbracket b_1 \rrbracket \times \ldots \times \llbracket b_n \rrbracket$. Also given a context $\Gamma = x_1 : b_1, \ldots, x_n : b_n$ we set $\llbracket \Gamma \rrbracket \triangleq \llbracket b_1, \ldots, b_n \rrbracket$. The superscript of $\llbracket - \rrbracket$ may be omitted hereafter.

Interpretation. Let M be a Σ-structure in an iteration category $\mathcal{C}$. We give the categorical meaning of a term judgment $\Gamma \vdash t \ : \overrightarrow{c}$ (where there are no metavariables) as a morphism $\llbracket t \rrbracket^M : \llbracket \Gamma \rrbracket \to \llbracket \overrightarrow{c} \rrbracket$ in $\mathcal{C}$ defined by

$$\begin{aligned} \llbracket \Gamma \vdash y_i \ : c \rrbracket^M &= \pi_i : \llbracket \Gamma \rrbracket \to \llbracket c \rrbracket \\ \llbracket \Gamma \vdash f(\,\overrightarrow{x_1^{a_1}}.t_1, \ldots \overrightarrow{x_n^{a_n}}.t_n\,) \ : \overrightarrow{c} \rrbracket^M &= \llbracket f \rrbracket^M_{\llbracket \Gamma \rrbracket}(\,\llbracket t_1 \rrbracket^M, \ldots, \llbracket t_n \rrbracket^M\,). \end{aligned}$$

We assume the following interpretations in any Σ_{def}-structure:

$$\begin{array}{lcl@{\qquad}l} \llbracket \langle\,\rangle \rrbracket^M_A &=& ! \ : A \to \mathbf{1} & \llbracket \langle -, \ldots, - \rangle \rrbracket^M_A (t_1, \ldots, t_n) = \langle t_1, \ldots, t_n \rangle \\ \llbracket \diamond \rrbracket^M_A (t, s) &=& t \circ \langle \mathrm{id}_A, s \rangle & \llbracket \mathsf{cy} \rrbracket^M_A (t) \ = \ t^{\dagger} \end{array}$$

Importantly, these satisfy the axioms AxCy because $\mathcal{C}$ is an iteration category.

▶ **Definition 3.3.** A $(\Sigma, \mathcal{E})$-*structure* is a Σ-structure M in $\mathcal{C}$ satisfying $\llbracket s \rrbracket^M = \llbracket t \rrbracket^M$ for every axiom $\Gamma \vdash s = t \ : c$ in $\mathcal{E}$. Let N be a $(\Sigma, \mathcal{E})$-structure in an iteration category $\mathcal{D}$. We say that an iteration functor $F : \mathcal{C} \to \mathcal{D}$ *preserves* $(\Sigma, \mathcal{E})$-*structures* if $F(\llbracket - \rrbracket^M) = \llbracket - \rrbracket^N$.

A *c-structure* (M, α) for a datatype declaration $(c, \Sigma_c, \mathcal{E})$ is a $(\Sigma_c, \mathcal{E})$-structure M with a family of morphisms of $\mathcal{C}$; $\quad \alpha \ \triangleq \ (\ (\![f]\!)^M \ : \llbracket b_1 \rrbracket \times \ldots \times \llbracket b_n \rrbracket \to \llbracket c \rrbracket\ )_{f : b_1, \ldots, b_n \to c \in \Sigma_c}$. It

Let $\Gamma = y_1 : \overrightarrow{b_1} \cdots , y_k : \overrightarrow{b_1}$. Suppose $\Theta \triangleright \Gamma', \overrightarrow{x_i : a_i} \vdash s_i : \overrightarrow{b_i}$ and

$$\mathrm{M}_1 : \overrightarrow{a}_1 \to \overrightarrow{b}_1, \ldots, \mathrm{M}_k : \overrightarrow{a}_k \to \overrightarrow{b}_k \triangleright \Gamma \vdash e : \overrightarrow{c}$$

where $\overrightarrow{x_i : a_i} = x_i^1 : a_i^1, \ldots, x_i^m : a_i^m$ for $m = |\overrightarrow{a_i}|$ and each $i = 1, \ldots, k$. Then a substitution $\Theta \triangleright \Gamma, \Gamma' \vdash e\,[\,\overrightarrow{\mathrm{M} := s}\,] : \overrightarrow{c}$ is inductively defined as follows.

$$\begin{aligned}
x\,[\,\overrightarrow{\mathrm{M} := s}\,] &\triangleq x \\
\mathrm{M}_i[t_1, \ldots, t_{m_i}]\,[\,\overrightarrow{\mathrm{M} := s}\,] &\triangleq s_i\,[x_1 \mapsto t_1\,[\,\overrightarrow{\mathrm{M} := s}\,], \ldots, x_{m_i} \mapsto t_{m_i}\,[\,\overrightarrow{\mathrm{M} := s}\,]] \\
f(\overrightarrow{y_1}.s_1, \ldots, \overrightarrow{y_m}.s_m)\,[\,\overrightarrow{\mathrm{M} := s}\,] &\triangleq f(\overrightarrow{y_1}.s_1\,[\,\overrightarrow{\mathrm{M} := s}\,], \ldots, \overrightarrow{y_m}.s_m\,[\,\overrightarrow{\mathrm{M} := s}\,])
\end{aligned}$$

where $[\,\overrightarrow{\mathrm{M} := s}\,]$ denotes $[\mathrm{M}_1 := s_1, \ldots, \mathrm{M}_k := s_k]$.

Figure 6 Substitution for metavariables.

actually defines a $(\Sigma_c, \mathcal{E})$-structure by $[\![f]\!]_A^M(t_1, \ldots, t_n) \triangleq (\![f]\!)^M \circ \langle t_1, \ldots, t_n \rangle$ for any A in $\mathcal{C}$. We say that an iteration functor $F : \mathcal{C} \to \mathcal{D}$ *preserves c-structures* if $F([\![c]\!]^M) = [\![c]\!]^N$, and $F((\![f]\!)^M) = (\![f]\!)^N$ for every $f \in \Sigma_c$.

▶ **Example 3.4** (The cyclic list datatype `CList`)**.** A `CList`-structure is given by a $(\Sigma_{\texttt{CList}}, \mathsf{AxCy})$-structure M having the interpretations of "[ ]" and "::".

▶ **Example 3.5** (The cyclic tree type `CTree`)**.** A `CTree`-structure is given by a $(\Sigma_{\texttt{CTree}}, \mathsf{AxBr} \cup \mathsf{AxBr}([\,], +))$-structure M where $[\![\texttt{CTree}]\!]^M = N$ and N is a commutative monoid object $(N,\ \eta{:}\mathbf{1} \to N,\ \mu{:}N \times N \to N)$ in $\mathcal{C}$ satisfying $\mu^\dagger = \mathrm{id}_N$, where $(\![\,[\,]\,]\!)^M = \eta$, $(\![+]\!)^M = \mu$. It satisfies $\mathsf{AxBr}([\,], +)$. Note that any `CTree`-structure is always a *degenerated commutative bialgebra* (cf. [16]) in a cartesian category $\mathcal{C}$, i.e. N is also a comonoid $(N, !, \Delta)$ that satisfies the compatibility

$$\Delta \circ \eta = \eta \times \eta, \quad \Delta \circ \mu \;=\; (\mu \times \mu) \circ (\mathrm{id} \times \langle \pi_2, \pi_1 \rangle \times \mathrm{id}) \diamond (\Delta \times \Delta), \quad \mu \circ \Delta \;=\; \mathrm{id}.$$

The last equation is by $\mu \circ \Delta = \mu \circ \langle \mathrm{id}, \mathrm{id} \rangle = \mu \circ \langle \mathrm{id}, (\mu)^\dagger \rangle =^{(\mathrm{dinat})} (\mu)^\dagger = \mathrm{id}$. Thus, a `CTree`-structure models branch and sharing of cyclic sharing trees.

We next give a syntactic category and a Σ-structure to prove categorical completeness. Let Σ be a signature, and $\mathcal{E}$ a set of axioms which is the union of AxCy and axioms for all datatype declarations of base types c. Given axioms $\mathcal{E}$, all proved equations $\Gamma \vdash s = t : \overrightarrow{c}$ (which must be the empty metavariable context) by the second-order equational logic (Fig. 5), defines an equivalence relation $=_\mathcal{E}$ on well-typed terms, where we also identify renamed terms by bijective renaming of free and bound variables. We write an equivalence class of terms by $=_\mathcal{E}$ as $[\Gamma \vdash t : \overrightarrow{c}]_\mathcal{E}$. We define the category $\mathbf{Tm}(\mathcal{E})$ of terms by taking

- objects: sequences of base types $\overrightarrow{c}$
- morphisms: $[\Gamma \vdash t : \overrightarrow{c}]_\mathcal{E} : [\![\Gamma]\!] \to [\![\overrightarrow{c}]\!]$, the identity: $[\overrightarrow{x : c} \vdash \langle \overrightarrow{x} \rangle : \overrightarrow{c}]_\mathcal{E}$
- composition: $[\overrightarrow{x : b} \vdash s : \overrightarrow{c}]_\mathcal{E} \circ [\Gamma \vdash t : \overrightarrow{b}]_\mathcal{E} \triangleq [\Gamma \vdash (\overrightarrow{x}.s) \diamond t : \overrightarrow{c}]_\mathcal{E}$

▶ **Proposition 3.6.** *$\mathbf{Tm}(\mathcal{E})$ is an iteration category, and has a $(\Sigma, \mathcal{E})$-structure* U.

Proof. We define a Σ-structure U by $[\![c]\!]^{\mathrm{U}} \triangleq c$ for each $c \in \mathcal{B}$ and $[\![f]\!]^{\mathrm{U}}_{\overrightarrow{a}} \triangleq f$ for each function symbol f and arbitrary base types $\overrightarrow{a}$. We take

- terminal object: () • pair: $\langle [s]_\mathcal{E}, [t]_\mathcal{E} \rangle \triangleq [\Gamma \vdash \langle s\,,\, t \rangle : \overrightarrow{c_1}, \overrightarrow{c_2}]_\mathcal{E}$
- product: concatenation of sequences
- Conway: $([\Gamma, \overrightarrow{x : c} \vdash t : \overrightarrow{c}]_\mathcal{E})^\dagger = [\Gamma \vdash \mathsf{cy}(\overrightarrow{x^c}.t) : \overrightarrow{c}]_\mathcal{E}$
- projections: $[x_1 : c_1, x_2 : c_2 \vdash x_i : c_i]_\mathcal{E}$

Then these data satisfy that $\mathbf{Tm}(\mathcal{E})$ is an iteration category and U forms a $(\Sigma,\mathcal{E})$-structure because of the axioms $\mathcal{E}$ for each $c \in \mathcal{B}$. Moreover, $(\llbracket c\rrbracket^{\mathrm{U}}, (f)_{f\in\Sigma_c})$ is a c-structure. ◀

Then $\llbracket t\rrbracket^{\mathrm{U}} = [t]_{\mathcal{E}}$ holds for all well-typed terms t. Using it, we have the following.

▶ **Theorem 3.7** (Categorical soundness and completeness). *$\Gamma \vdash s = t \ : \overrightarrow{c}$ is derivable iff $\llbracket s\rrbracket^M_{\mathcal{C}} = \llbracket t\rrbracket^M_{\mathcal{C}}$ holds for all iteration categories $\mathcal{C}$ and all $(\Sigma,\mathcal{E})$-structures in $\mathcal{C}$.*

▶ **Theorem 3.8.** *For a $(\Sigma,\mathcal{E})$-structure M in an iteration category $\mathcal{C}$, there exists a unique iteration functor $\Psi^M : \boldsymbol{Tm}(\mathcal{E}) \longrightarrow \mathcal{C}$ that preserves $(\Sigma,\mathcal{E})$-structures. Pictorially, it is expressed as the following picture, where* Tm *denotes the set of all terms (without quotient).*

$$\begin{tikzcd}
\mathrm{Tm} \arrow[r, "\llbracket - \rrbracket^{\mathrm{U}}"] \arrow[d, "\llbracket - \rrbracket^{M}"'] & \boldsymbol{Tm}(\mathcal{E}) \arrow[dl, "\Psi^M"] \\
\mathcal{C} &
\end{tikzcd}$$

Proof. We write simply Ψ for Ψ^M. Since Ψ preserves $(\Sigma,\mathcal{E})$-structures, $\Psi(\llbracket - \rrbracket^{\mathrm{U}}) = \llbracket - \rrbracket^{M}$ holds. Hence $\Psi(\llbracket t\rrbracket^{\mathrm{U}}) = \Psi([t]_{\mathcal{E}}) = \llbracket t\rrbracket^M$ for any t, meaning that the mapping Ψ is required to satisfy

$$\begin{array}{lll}
\Psi(\ [\Gamma \vdash y_i \ \ : c]_{\mathcal{E}} &) = \ \pi_i & \qquad \Psi([\Gamma \vdash \langle\,\rangle \ \ : ()]_{\mathcal{E}}) = ! \\
\Psi(\ [\Gamma \vdash \langle s\,,\, t\rangle \ \ : \overrightarrow{c_1}, \overrightarrow{c_2}]_{\mathcal{E}} &) = \langle \Psi[\Gamma \vdash s \ \ : \overrightarrow{c_1}]_{\mathcal{E}}, \Psi[\Gamma \vdash t \ \ : \overrightarrow{c_2}]_{\mathcal{E}}\rangle & \\
\Psi(\ [\Gamma \vdash \mathsf{cy}(\overrightarrow{x^{c}}.t) \ \ : \overrightarrow{c}]_{\mathcal{E}} &) = (\Psi[\Gamma, \overrightarrow{x : c} \vdash t \ \ : \overrightarrow{c}]_{\mathcal{E}})^{\dagger} & \\
\Psi(\ [\Gamma \vdash f(\ \overrightarrow{x_1^{a_1}}.t_1, \ldots, \overrightarrow{x_m^{a_m}}.t_m) \ \ : c]_{\mathcal{E}} &) = \llbracket f\rrbracket^M_{\llbracket\Gamma\rrbracket}(\Psi[\Gamma, \overrightarrow{x_1 : a_1} \vdash t_1 \ \ : b_1]_{\mathcal{E}}, \ldots) & \\
\Psi(\ [\Gamma \vdash (\overrightarrow{x^{b}}.t) \diamond s \ \ : c]_{\mathcal{E}} &) = \Psi[\Gamma, \overrightarrow{x : b}, \vdash t \ \ : c]_{\mathcal{E}} \circ \langle \mathrm{id}_{\llbracket\Gamma\rrbracket}, \Psi[\Gamma \vdash s \ \ : \overrightarrow{b}]_{\mathcal{E}}\rangle &
\end{array} \tag{2}$$

The above equations mean that Ψ is an iteration functor that sends the $(\Sigma,\mathcal{E})$-structure U to M. Such Ψ is uniquely determined by these equations because U is a $(\Sigma,\mathcal{E})$-structure. ◀

4 Fold on Cyclic Datatype

Fix a cyclic datatype c (say, the type `CList` of cyclic lists). By the previous theorem, for a c-structure M, the interpretation $\llbracket - \rrbracket^M$ determines a c-structure preserving iteration functor Ψ^M. If we take the target category $\mathcal{C}$ as also $\mathbf{Tm}(\mathcal{E})$, M should be another cyclic datatype b (say, the `CNat` of cyclic natural numbers), where the constructors of c are interpreted as terms of type b. For example, the sum of a cyclic list in Introduction is understood in this way. Thus the functor Ψ^M determined by $\llbracket - \rrbracket^M$ can be understood as a transformation of cyclic data from terms of type c to terms of type b.

Along this idea, we formulate the fold operation from the cyclic datatype c to b by the functor Ψ^M. Let (M, α) be an arbitrarily c-structure in $\mathbf{Tm}(\mathcal{E})$, where $\llbracket c\rrbracket^M = b \in \mathcal{B}$. We write the arrow part function Ψ^M on hom-sets as the fold, i.e.

$$\mathbf{fold}^c_b(\alpha) : \mathbf{Tm}(\mathcal{E})(\llbracket\Gamma\rrbracket^{\mathrm{U}},\ c) \longrightarrow \mathbf{Tm}(\mathcal{E})(\llbracket\Gamma\rrbracket^{M},\ b).$$

4.1 Formalising fold as a second-order algebraic theory

The **fold** is a function on equivalence classes of term judgments modulo $\mathcal{E}$ including $\mathsf{AxCy} \cup \mathsf{AxBr}$ characterised by (2). Equivalently, we regard it as a function on terms (judgments) that preserves $=_{\mathcal{E}}$, i.e. $s =_{\mathcal{E}} t \quad \Rightarrow \quad \mathbf{fold}^c_b(\alpha)(s) \ =_{\mathcal{E}} \ \mathbf{fold}^c_b(\alpha)(t)$. In this subsection, we axiomatise the function $\mathbf{fold}^c_b$ as the laws of fold within second-order equational logic using (2).

Formalising a c-structure (M, α). To give $\alpha = ((\!|f|\!)^M : [\![a_1]\!] \times \ldots \times [\![a_n]\!] \to [\![c]\!])_{f:a_1,\ldots,a_n \to c \in \Sigma_c}$ is to give terms $x_1 : [\![a_1]\!], \ldots, x_n : [\![a_n]\!] \vdash e_f \ : b$ for all $f : a_1, \ldots, a_n \to c \in \Sigma_c$ such that $(\!|f|\!)^M = [e_f]_{\mathcal{E}}$. We represent α as a tuple of terms e_f according to function symbols in Σ_c by the order of datatype constructors listed in a `ctype` declaration of c.

Formalising fold. We next formalise the fold operation in second-order algebraic theory. The type of fold may be chosen as $\mathsf{fold}^c_b : (\overrightarrow{a_1} \to b), \ldots, (\overrightarrow{a_k} \to b), (c^m \to c) \to (b^m \to b)$, where the first k-arguments correspond to the c-structure α. But in second-order algebraic theory, the codomain of function symbol *must be* a sequence of base types (§2.1), so the current codomain $(b^m \to b)$ is inappropriate. To solve it, we introduce a new base type jty^b_m as the type of "encoded judgments" and a function symbol $\mathsf{judgmt}^b_m : (b^m \to b) \to \mathsf{jty}^b_m$ for each $m \in \mathbb{N}, b \in \mathcal{B}$. We encode a judgment $\overrightarrow{y : b} \vdash t \ : b$ as a term $\mathsf{judgmt}^b_m(\overrightarrow{y}.t)$, for $m = |\overrightarrow{y}|$, which will be denoted by $\overrightarrow{y} \Vdash t$ for readability. In case of $m = 0$, $\mathsf{jty}^b_0 = b$ and we do not use the constructor judgmt^b_0. In summary, the fold is formalised as the function symbol of the type

$$\mathsf{fold}^c_b : (\overrightarrow{a_1} \to b), \ \ldots, (\overrightarrow{a_k} \to b), \mathsf{jty}^c_m \to \mathsf{jty}^b_m$$

and the mathematical expression $\mathbf{fold}^c_b(\alpha)([\Gamma \vdash t \ : c]_{\mathcal{E}})$ at the level of semantics is formalised as a term $\mathsf{fold}^c_b(e_1, \ldots, e_k, \ \Gamma \Vdash t)$ in second-order algebraic theory, where each e_i corresponds to $(\!|f_i|\!)^M$ for $f_i \in \Sigma_c$ in α.

Finally, we axiomatise fold by using the characterisation (2) in case of particular category $\mathcal{C} = \mathbf{Tm}(\mathcal{E})$ and a c-structure. Here we assume an additional function symbol $\mathsf{app} : (\overrightarrow{a} \to b), \overrightarrow{a} \to b$. We give the axioms FOLD in Fig. 7, which is straightforward formalisation of (2) in case of the target c-structure is given by terms of type b. The arguments of fold expressing the c-structure are abbreviated as E for simplicity. We also include the axioms and theorems (8)-(12) taken from AxCy and AxBr for simplification. This importation of several axioms from $\mathsf{AxCy} \cup \mathsf{AxBr}$ to the second-order algebraic theory FOLD is harmless because our general framework is second-order equational logic under $\mathcal{E} \cup \mathsf{FOLD}$ which includes $\mathsf{AxCy} \cup \mathsf{AxBr}$. The following is immediate by construction.

▶ **Proposition 4.1.** *Using the above formalisation process, the following are equivalent.*

- $\mathbf{fold}^c_b(\alpha)([\Gamma \vdash t \ : c]_{\mathcal{E}}) = [\Gamma' \vdash u \ : b]_{\mathcal{E}}$
- $\vdash \mathit{fold}(e_1, \ldots, e_k, \ \Gamma \Vdash t) = (\Gamma' \Vdash u) \ : \mathit{jty}^b_m$ *is derived from the axioms* $\mathcal{E} \cup \mathit{FOLD}$ *using the second-order equational logic.*

where α, e_i *and* t *are* fold *free,* $\Gamma = x_1 : c, \ldots, x_m : c$, $\Gamma' = x_1 : b, \ldots, x_m : b$.

▶ **Example 4.2.** The plus function on `CNat` can be defined as fold as follows.

```
plus : CNat,CNat → CNat
spec plus(m, n) = pl(m)
  where pl(0)    = n
        pl(S(m)) = S(pl(m))
fun plus(m, n) = fold (n, x.S(x)) m
```

Fold

(1) $\mathsf{fold}(E,\ \overrightarrow{y}^{c} \Vdash y_i) \quad = \overrightarrow{y}^{b} \Vdash y_i \qquad (\text{for } y_i \in \{\overrightarrow{y}\})$

(2) $\mathsf{fold}(E,\ \overrightarrow{y} \Vdash \langle\,\rangle) \quad = \overrightarrow{y} \Vdash \langle\,\rangle$

(3) $\mathsf{fold}(E,\ \overrightarrow{y} \Vdash \langle \textsc{s}[\overrightarrow{y}], \textsc{t}[\overrightarrow{y}]\rangle) \quad = \overrightarrow{y} \Vdash \langle \mathsf{app}(\mathsf{fold}(E,\ \overrightarrow{y} \Vdash \textsc{s}[\overrightarrow{y}]), \overrightarrow{y}),\ \mathsf{app}(\mathsf{fold}(E,\ \overrightarrow{y} \Vdash \textsc{t}[\overrightarrow{y}]), \overrightarrow{y})\rangle$

(4) $\mathsf{fold}(E,\ \overrightarrow{y} \Vdash \mathsf{cy}(\overrightarrow{x}.\textsc{t}[\overrightarrow{y}, \overrightarrow{x}])) = \overrightarrow{y} \Vdash \mathsf{cy}(\overrightarrow{x}.\mathsf{app}(\mathsf{fold}(E,\ \overrightarrow{y}, \overrightarrow{x} \Vdash \textsc{t}[\overrightarrow{y}, \overrightarrow{x}]), \overrightarrow{y}, \overrightarrow{x}))$

(5) $\mathsf{fold}(E,\ \overrightarrow{y} \Vdash d(\overrightarrow{\textsc{a}}, \textsc{t}_1[\overrightarrow{y}], \dots, \textsc{t}_n[\overrightarrow{y}])) = \overrightarrow{y} \Vdash (\overrightarrow{x}.\textsc{e}_d[\overrightarrow{\textsc{a}}, \overrightarrow{x}]) \diamond \langle \mathsf{app}(\mathsf{fold}(E,\ \overrightarrow{y} \Vdash \textsc{t}_1[\overrightarrow{y}]), \overrightarrow{y}), \dots, \rangle$

(6) $\mathsf{fold}(E,\ x \Vdash (\overrightarrow{y}.\textsc{t}[\overrightarrow{y}]) \diamond \textsc{s}[\overrightarrow{x}]) = \overrightarrow{x} \Vdash \overrightarrow{y}.\mathsf{app}(\mathsf{fold}(E,\ \overrightarrow{y} \Vdash \textsc{t}[\overrightarrow{y}]), \overrightarrow{y}) \ \diamond\ \mathsf{app}(\mathsf{fold}(E,\ \overrightarrow{x} \Vdash \textsc{s}[\overrightarrow{x}]), \overrightarrow{x})$

(7) $\mathsf{app}(\overrightarrow{x} \Vdash \textsc{s}[\overrightarrow{x}], z_1, \dots, z_m) \quad = \textsc{s}[z_1, \dots, z_m]$

Bekič and cycle cleaning

(8) $\mathsf{cy}(\overrightarrow{x}, \overrightarrow{y}.\,\langle \widehat{\textsc{t}},\ \widehat{\textsc{s}}\rangle) \quad = \langle\, \mathsf{cy}(\overrightarrow{x}.\,(\overrightarrow{y}.\widehat{\textsc{t}}) \diamond \mathsf{cy}(\overrightarrow{y}.\widehat{\textsc{s}})),\ \ \mathsf{cy}(\overrightarrow{y}.\,(\overrightarrow{x}.\widehat{\textsc{s}}) \diamond \mathsf{cy}(\overrightarrow{x}.\,(\overrightarrow{y}.\widehat{\textsc{t}}) \diamond \mathsf{cy}(\overrightarrow{y}.\widehat{\textsc{s}})))\rangle$

(9) $\mathsf{cy}(\overrightarrow{y}.\textsc{t}) \quad = \textsc{t} \qquad (\text{NB. } \textsc{t} \text{ cannot contain } y)$

(10) $\mathsf{cy}(x^c.x) = [\,] \quad \mathsf{cy}(x^c.\textsc{t} + x) \quad = \textsc{t} \qquad (\text{if a type } c \text{ has } [\,] \text{ and “+” satisfying } \mathsf{AxBr})$

Composition

(11) $(\overrightarrow{y}.\textsc{t}[\overrightarrow{y}]) \diamond \langle \textsc{s}_1\,, \dots, \textsc{s}_n\rangle \quad = \textsc{t}[\textsc{s}_1\,, \dots, \textsc{s}_n]$

Here E is a sequence $(\textsc{e}_d)_{d\in\Sigma_c}$ of metavariables and $d \in \Sigma_c$.
In (8), $\widehat{\textsc{t}}$ and $\widehat{\textsc{s}}$ are short for $\textsc{t}[\overrightarrow{x}, \overrightarrow{y}]$ and $\textsc{s}[\overrightarrow{x}, \overrightarrow{y}]$, respectively.

Figure 7 Second-order algebraic theory FOLD of fold from the datatype c to b.

In specification, we understand `plus` in terms of a unary function `pl` which recurses on the first argument m and gives the second argument n if $m = 0$. Hence it is fold where the parameter n is passed to the Σ-structure of fold.

4.2 Primitive recursion by fold

The fold formalised above covers the ordinary fold on algebraic datatypes. Thus, we expect that various techniques on fold developed in functional programming, such as the fold fusion technique and representation of recursion principles such as [28] may be transferred to the current setting. Here we consider a way to implement a particular pattern of recursion appearing often in specifications as a fold. Consider a specification having a clause

$$\mathtt{spec}\ f(d(t)) = e$$

where e contains $f(t)$ as well as t solely (cf. examples in §6). (If e constrains only the recursive call $f(t)$, it is merely a pattern of structural recursion, so it can be implemented by fold using the structure $x.e'$ where all the recursive calls $f(t)$ in e are abstracted to x as Example 4.2.)

The above specification (i.e. the clause with `spec` keyword) can be seen as describing primitive recursion, because it is similar the primitive recursion on natural numbers $f(S(n)) = e(f(n), n)$, where both n and $f(n)$ can be used at the right-hand side. In functional programming, it is known that primitive recursion on algebraic datatypes can be represented as fold, called paramorphism [27]. We sketch how we can import this technique (see also [21, §3.5], [22, §4.2]). For the above case, we take the fold where the target Σ-structure is the product b, b of types, i.e. $\mathsf{fold}^{c}_{b,b}$. In this case, variables in context are doubled at the right-hand sides of the axioms FOLD, e.g. for (1) $\mathsf{fold}(E,\ \overrightarrow{y} \Vdash y_i) = (\overrightarrow{y}, \overrightarrow{y'} \Vdash \langle y_i, y'_i\rangle)$. Let $\pi_1 = (x, y.x)$. We implement f as

$$f(\Gamma \Vdash t) \triangleq \pi_1 \diamond \mathsf{fold}(\cdots,\ \langle x, y.e',\ d(y)\rangle,\ \cdots,\ \Gamma \Vdash t)$$

where e' is obtained from e in the specification by replacing every “$f(t)$” with x and every “t” not in the form $f(t)$ with y. The other components of the c-structure for fold are implemented

Fold

$$\begin{array}{lll}
(1) & \mathsf{fold}(E,\ \overrightarrow{y^{\mathsf{Var}_c}}.\,\mathsf{v}(y_i)) & \to \overrightarrow{y^{\mathsf{Var}_b}}.\,\mathsf{v}(y_i) \\
(2) & \mathsf{fold}(E,\ \overrightarrow{y}.\,\langle\,\rangle) & \to \overrightarrow{y}.\,\langle\,\rangle \\
(3) & \mathsf{fold}(E,\ \overrightarrow{y}.\,\langle \mathrm{S}[\overrightarrow{y}], \mathrm{T}[\overrightarrow{y}]\rangle) & \to \overrightarrow{y}.\,\langle \mathsf{fold}(E,\ \overrightarrow{y}.\,\mathrm{S}[\overrightarrow{y}])@\overrightarrow{y},\ \mathsf{fold}(E,\ \overrightarrow{y}.\,\mathrm{T}[\overrightarrow{y}])@\overrightarrow{y}\rangle \\
(4) & \mathsf{fold}(E,\ \overrightarrow{y}.\,\mathsf{cy}^1(x.\mathrm{T}[\overrightarrow{y}, x])) & \to \overrightarrow{y}.\,\mathsf{cy}^1(x.\mathsf{fold}(E,\ \overrightarrow{y}, x.\,\mathrm{T}[\overrightarrow{y}, x])@\overrightarrow{y}, x) \\
(5) & \mathsf{fold}(E,\ \overrightarrow{y}.\,d(\overrightarrow{\mathrm{A}}, \mathrm{T}_1[\overrightarrow{y}], \ldots, \mathrm{T}_n[\overrightarrow{y}])) & \to \overrightarrow{y}.\,(\overrightarrow{x}.\mathrm{E}_d[\overrightarrow{\mathrm{A}},\ \overrightarrow{x}]) \diamond \langle\,\mathsf{fold}(E,\ \overrightarrow{y}.\,\mathrm{T}_1[\overrightarrow{y}])@\overrightarrow{y}, \ldots\rangle \\
(6) & \mathsf{fold}(E,\ \overrightarrow{x}.\,(\overrightarrow{y}.\mathrm{T}[\overrightarrow{y}]) \diamond \mathrm{S}[\overrightarrow{x}]) & \to \overrightarrow{x}.\,(\overrightarrow{y}.\mathsf{fold}(E,\ \overrightarrow{y}.\,\mathrm{T}[\overrightarrow{y}])@\overrightarrow{y})\ \diamond\ \mathsf{fold}(E,\ \overrightarrow{x}.\,\mathrm{S}[\overrightarrow{x}])@\overrightarrow{x}
\end{array}$$

Bekič and cycle cleaning (for $m, n \geq 1$)

$$\begin{array}{lll}
(8) & \mathsf{cy}^{m+n}(\overrightarrow{x}, \overrightarrow{y}.\,\langle\,\widehat{\mathrm{T}},\ \widehat{\mathrm{S}}\,\rangle) & \to \langle\,\mathsf{cy}^m(\overrightarrow{x}.\,(\overrightarrow{y}.\widehat{\mathrm{T}}) \diamond \mathsf{cy}^n(\overrightarrow{y}.\widehat{\mathrm{S}})), \\
 & & \quad \mathsf{cy}^n(\overrightarrow{y}.\,(\overrightarrow{x}.\widehat{\mathrm{S}}) \diamond \mathsf{cy}^m(\overrightarrow{x}.\,(\overrightarrow{y}.\widehat{\mathrm{T}}) \diamond \mathsf{cy}^n(\overrightarrow{y}.\widehat{\mathrm{S}})))\rangle \\
(9) & \mathsf{cy}(\overrightarrow{y}.\mathrm{T}) & \to \mathrm{T} \\
(10) & \mathsf{cy}(x.\mathsf{v}(x)) \to [\,] \quad \mathsf{cy}(x.\mathrm{T} + \mathsf{v}(x)) & \to \mathrm{T} \qquad \text{(if a type } c \text{ has } [\,] \text{ and "+" satisfying } \mathsf{AxBr})
\end{array}$$

Composition

$$\begin{array}{lll}
(11) & (\overrightarrow{y}.\,\mathsf{v}(y_i)) \diamond \langle\,\overrightarrow{\mathrm{S}}\,\rangle & \to \mathrm{S}_i \\
(12) & (\overrightarrow{y}.\,d(\overrightarrow{x_1}.\,\mathrm{T}_1[\overrightarrow{y}, \overrightarrow{x_1}], \ldots)) \diamond \langle\,\overrightarrow{\mathrm{S}}\,\rangle & \to d(\overrightarrow{x}.\,(\overrightarrow{y}.\,\mathrm{T}_1[\overrightarrow{y}, \overrightarrow{x_1}]) \diamond \langle\,\overrightarrow{\mathrm{S}}\,\rangle, \ldots, (\overrightarrow{y}.\,\mathrm{T}_n[\overrightarrow{y}, \overrightarrow{x_n}]) \diamond \langle\,\overrightarrow{\mathrm{S}}\,\rangle) \\
 & & \quad \text{(for each constructor } d)
\end{array}$$

Figure 8 Rewrite system FOLDr.

by the same way, according to the specification. Then by induction on the structure of terms t, we have $\mathsf{fold}^c_{b,b}(E, \overrightarrow{y} \Vdash t) = \overrightarrow{y} \Vdash \langle\,\mathsf{app}(f(\overrightarrow{y} \Vdash t), \overrightarrow{y}),\ t\,\rangle$ for closed t using FOLD. By the characterisation (2), we have $f(d(t)) = (x, y.\,e') \diamond \langle f(t), t\rangle = e$, thus it satisfies the specification. We use extensively this technique in §6.

5 Strongly Normalising Computation Rules for FOLD

We expect that FOLD provides strong normalising computation rules. An immediate idea is to regard the axioms FOLD as rewrite rules by orienting each axiom from left to right.

But proving strong normalisation (SN) of FOLD is not straightforward. The sizes of both sides of equations in FOLD are not decreasing in most axioms. So, assigning some "measure" to the rules in FOLDr that is strictly decreasing is difficult for this case. If the axioms (regarded as rewrite rules) are a binding CRS [18] (meaning that every meta-application $\mathrm{M}[t_1, \ldots, t_n]$ is of the form $\mathrm{M}[\overrightarrow{x}]$), then it is possible to use a simple polynomial interpretation to prove termination of second-order rules [18]. Unfortunately, this is not the case because in (5) and (11) there are meta-applications violating the condition. Existence of meta-application means that it essentially involves the β-reduction, thus it has the same difficulty as proving strong normalisation of the simply-typed λ-calculus.

We use a general established method of *the General Schema*[4, 3], which is based on Tait's computability method to show SN. The General Schema has succeeded to prove SN of various recursors such as the recursor in Gödel's System T. The basic idea of the General Schema is to check whether the arguments of recursive calls in the the right-hand side of a rewrite rule are "smaller" than the left-hand sides' ones. It is similar to Coquand's notion of "structurally smaller" [8], but more relaxed and extended.

Rewrite rules using strictly positive types. In order to apply the General Schema criterion, we refine the second-order algebraic theory FOLD to the rewrite rule FOLDr. The General Schema in [3] is formulated for a framework of rewrite rules called inductive datatype systems, whose (essentially) second-order fragment is almost the same as the present formulation given in §2. Minor differences are as follows.

(i) The target of function symbols must be a single (not necessary base) type in inductive datatype systems. Hence we introduce the product type constructor ×, assume that $b_1 \times b_2$ is again a base type in the sense of §2.2, and use it for the target type.

(ii) Instead of term $x_1, \ldots, x_n.t$ that binds a sequence of variables and is of sort $a_1, \ldots, a_n \to b$ in second-order algebraic theory, we use $x_1. \cdots .x_n.t$ that repeatedly binds single variables and is of type $a_1 \to \cdots \to a_n \to b$. Now the abbreviation $\overrightarrow{x}.t$ denotes $x_1. \cdots .x_n.t$.

(iii) We assume that a function symbol $@ : (a \to b), a \to b$ and a rule $(x.\mathrm{T}[x])@\mathrm{S} \to \mathrm{T}[\mathrm{S}]$ for application ([3] Def. 2, β-IDTS). We write $(\overrightarrow{x}.t)@\overrightarrow{s}$ for $(\overrightarrow{x}.t)@s_1 \cdots @s_n$.

The General Schema requires a notion of strictly positivity. Crucially, the constructors used in FOLD are *not strictly positive*, as cy and $\diamond$ involve negative occurrence of c in $(c \to c)$. We can overcome this problem by modifying the type $(c \to c)$ to a restricted $(\mathsf{Var}_c \to c)$, where Var_c is a base type having no constructor considered as a type of "variables" of type c. We assume the constructor v which embeds a "variable" into a term. We modify the types of constructors as follows:

$$\begin{array}{llll}
\langle -, \cdots, - \rangle & : c_1, \ldots, c_n \to \overrightarrow{c}, & \mathsf{cy} & : (\overrightarrow{\mathsf{Var}_c} \to \overrightarrow{c}) \to \overrightarrow{c}, \\
\mathsf{v} & : \mathsf{Var}_c \to c, & - \diamond - & : (\overrightarrow{\mathsf{Var}_a} \to \overrightarrow{c}), a_1 \times \cdots \times a_n \to \overrightarrow{c},
\end{array}$$

where $\overrightarrow{c}$ denotes $c_1 \times \cdots \times c_n$, c_i's and a are base types, $\overrightarrow{\mathsf{Var}_a} \to \tau$ is short for $\mathsf{Var}_{a_1} \to \cdots \to \mathsf{Var}_{a_n} \to \tau$ and similarly for $\overrightarrow{\mathsf{Var}_c} \to \tau$. The use of a type $\mathsf{Var}_\sigma \to \tau$ to represent binders is known in the field of mechanized reasoning, sometimes called *(weak) higher-order abstract syntax* [9]. Accordingly, the type of fold is now

$$\mathsf{fold} : (\overrightarrow{\mathsf{Var}_{a_1}} \to b), \ldots, (\overrightarrow{\mathsf{Var}_{a_k}} \to b), (\mathsf{Var}_c^m \to c) \to (\mathsf{Var}_b^m \to b),$$

and rules are modified to FOLDr giving in Fig. 8. In case of inductive data type system, a term of the form $\overrightarrow{y}.t$ is allowed and well-typed (although not allowed as a sole term in case of second-order algebraic theory), thus we can now write the binder $\overrightarrow{y}.-$ directly at the right-hand side. FOLDr is correct.

▶ **Lemma 5.1.** *If $t \to^+_{\mathsf{FOLDr}} t'$ where t' does not involve @, then $\check{t} = \check{t'}$ is derivable from FOLD without using (Sym) where $\check{t}, \check{t'}$ recovers the original term notation from the encoding we gave above.*

▶ **Theorem 5.2.** *The rewrite system FOLDr is strongly normalising.*

Proof. Since FOLDr fits into the General Schema using the well-founded order

$$\mathsf{fold} > \mathsf{cy}^m > \mathsf{cy}^n > \diamond > \mathsf{v} > \text{any other constructors}, @$$

for natural numbers $m > n$, it is strongly normalising. Note that the superscript of cy in (8) indicates the number of arguments (cf. §2.1). This kind of indication of an "invariant" is similar to the idea of higher-order semantic labelling [19], but here we just make the existing superscript explicit rather than labelling. ◀

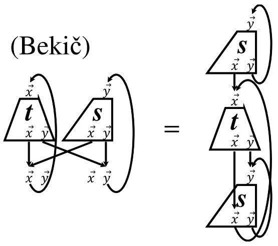
6 Computing by Fold on Cyclic Datatypes

In this section, we demonstrate fold computation on cyclic data by several examples.

▶ **Example 6.1.** As an example of primitive recursion on cyclic datatypes mentioned in §4.2. we consider the tail of a cyclic list, which we call `ctail`. It should satisfy the specification below right. But how to define the tail of cy-term is not immediately clear. For example, what should be the result of `ctail (cy(x.1::2::x))`? This case may need unfolding of cycle as in [17]. A naive unfolding by using the fixed point law $\mathsf{cy}(x.t) = t\,[\mathsf{cy}(x.t)/x]$ violates strong normalisation because it copies the original term. It actually *increases* complexity.

```
ctail : CList → CNat
spec ctail ([])      = []
     ctail (k::t)    = t
     ctail (cy(x. t))= ??
```

(Bekič)



Rather than the fixed point law, we use another important principle of cyclic structures known as *Bekič law*, given by the axiom (Bekič) in AxCy or (8) in FOLDr. It says that the fixed point of a pair can be obtained by computing the fixed point of each of its components independently and composing them suitably (see the right figure). It can be seen as *decreasing* complexity of cyclic computation because looking at the argument of cy, the number of components of tuple is reduced. We define `ctail` by fold.

```
fun ctail(t) = π1 ◇ fold (<[],[]>, k.x.y.<y, k::y>) t
```

```
ctail(cy(x.1::2::x)) →+ π1 ◇ cy(x.y. <2::y, 1::2::y>)
     →+ π1 ◇ <cy(x.2::cy(y.1::2::y)), cy(y.1::2::y)>
     →  cy(x.2::cy(y.1::2::y)) → 2::cy(y.1::2::y)      (Normal form)
```

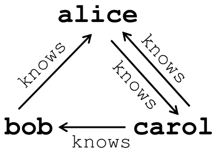
Note that the above normal form does not mean a head normal form and we do not rely on lazy evaluation. The highlighted step uses Bekič law.

▶ **Example 6.2.** This example shows that our cyclic datatype has ability to express directed graphs. The graph shown below right represents friend relationship, which describes Alice knows Carol, Bob knows Alice, and Carol knows Alice and Bob. This is represented as a term

```
  cy(a.b.c.<name("alice")+knows(c), name("bob")+knows(a),
            name("carol")+knows(a)+knows(b)>)
```

which we call `g`. The term `g` is of type `FriendGraph` defined as follows.

```
   ctype FriendGraph where
     knows : FriendGraph → FriendGraph
     name  : String → FriendGraph
     [] : FriendGraph
     + : FriendGraph,FriendGraph → FriendGraph
   with axioms AxCy,AxBr([],+)
```



We define a function `collect` that collects all names in a graph as a name list of type `Names`.

```
ctype Names where
  nm : String → Names
  [] : Names
  + : Names,Names → Names
with axioms AxCy,AxBr([],+)
```

```
collect : FriendGraph → Names
spec collect (knows(t)) = collect(t)
     collect (name(p))  = nm(p)

fun collect t = π1◇ folde t
```

Then we collect certainly all names by FOLDr as follows, where `folde` is short for `fold (x.y.<x,knows(y)>, x.y.<nm(y),name(y)>)`.

```
collect g = π1◇ folde g
→ π1◇ (cy(a.a'.b.b'.c.c'.
    <folde(a.b.c.name("alice")+knows(c)), folde(a.b.c.name("bob")+knows(a)),
     folde(a.b.c.name("carol")+knows(a)+knows(b)>)))
→+ π1◇ (cy(a.a'.b.b'.c.c'.
  < <nm("alice"),name("alice")>, <nm("bob"),name("bob")>, <nm("carol"),name("carol")> >)
→+ nm("alice")+nm("bob")+nm("carol")
```

7 Related Work

There has been various work to deal with graph computation and cyclic data structures in functional programming and foundational calculi including [12, 30, 6, 24, 26, 2, 1]. Several work [12, 30, 26] relies on lazy evaluation to deal with cycles. The present paper is different in this respect. We do not assume any particular operational semantics nor strategy to obtain strongly normalising fold on cyclic data. This point may be useful to deal with cyclic datatypes in proof assistance like Coq or Agda.

Foundational graph rewriting calculi, such as equational term graph rewriting systems [2], are general frameworks of graph computation. The fold on cyclic datatype in this paper is more restricted than general graph rewriting. However, our emphasis is clarification of the categorical and algebraic structure of cyclic datatypes and the computation fold on them by regarding fold as a structure preserving map, rather than unrestricted rewriting. It was a key to obtain strong normalisation. We also hope that it will be useful for further optimisation such as the fold fusion based on semantics as done in [22] Sec. 4.3. The general study of graph rewriting was also important for our study at the foundational level. The unit “[ ]” of branching in AxBr corresponds to the black hole constant “•” considered in [2], due to [5]. This observation has been used to give an effective operational semantics of graph transformation in [26].

In [17, 20], the present author aimed to capture the unique representations of cyclic sharing data structures (without any quotient) in order to obtain efficient functional programming concept. The approach taken in this paper is different. We have assumed the axioms AxCy and AxBr to equate bisimilar graphs. The point is that bisimulation on graphs can be efficiently decidable [10], thus now we regard that uniqueness of representation is not quite serious.

In [21, 22], the author and collaborators gave algebraic and categorical semantics of a graph transformation language UnCAL [6, 24] using iteration categories [5]. The graph data of UnCAL corresponds to cyclic sharing trees of type `CTree` in the present paper. UnCAL does not have the notion of types. Hence structural recursive functions in UnCAL are always transformations from general graphs to graphs, thus typing such as `sum:CList→CNat` (in Introduction) or `collect:FriendGraph→Names` (in §6) could not be formulated. The present paper advanced one step further by developing a suitable algebraic framework that captures datatypes supporting cycles and sharing. We have used a rewriting technique of the General Schema [3] to show strong normalisation (not merely termination of a particular computation strategy or algorithm) of fold. Such a direction has not been pursued so far.

Acknowledgments. I am grateful to Kazutaka Matsuda and Kazuyuki Asada for various discussions about calculi and programming languages about graphs.

References

1 Z. M. Ariola and S. Blom. Cyclic lambda calculi. In *Theoretical Aspects of Computer Software, LNCS 1281*, pages 77–106, 1997.

2 Z. M. Ariola and J. W. Klop. Equational term graph rewriting. *Fundam. Inform.*, 26(3/4):207–240, 1996.

3 F. Blanqui. Termination and confluence of higher-order rewrite systems. In *Rewriting Techniques and Application (RTA 2000)*, LNCS 1833, pages 47–61. Springer, 2000.

4 F. Blanqui, J.-P. Jouannaud, and M. Okada. Inductive data type systems. *Theoretical Computer Science*, 272:41–68, 2002.

5 S. L. Bloom and Z. Ésik. *Iteration Theories – The Equational Logic of Iterative Processes*. EATCS Monographs on Theoretical Computer Science. Springer, 1993.

6 P. Buneman, M. F. Fernandez, and D. Suciu. UnQL: A query language and algebra for semistructured data based on structural recursion. *VLDB J.*, 9(1):76–110, 2000.

7 C. Calcagno, P. Gardner, and U. Zarfaty. Context logic and tree update. In *Proc. of POPL'05*, pages 271–282, 2005. doi:10.1145/1040305.1040328.

8 T. Coquand. Pattern matching with dependent types. In *Proc. of the 3rd Work. on Types for Proofs and Programs*, 1992.

9 J. Despeyroux, A. Felty, and A. Hirschowitz. Higher-order abstract syntax in Coq. In *Typed Lambda Calculi and Applications, LNCS 902*, pages 124–138, 1995.

10 A. Dovier, C. Piazza, and A. Policriti. An efficient algorithm for computing bisimulation equivalence. *Theoretical Computer Science*, 311:221–256, 2004. Issues 1-3.

11 Z. Ésik. Axiomatizing iteration categories. *Acta Cybernetica*, 14:65–82, 1999.

12 L. Fegaras and T. Sheard. Revisiting catamorphisms over datatypes with embedded functions (or, programs from outer space). In *POPL'96*, pages 284–294, 1996. doi:10.1145/237721.237792.

13 M. Fiore and C.-K. Hur. Second-order equational logic. In *Proc. of CSL'10*, LNCS 6247, pages 320–335, 2010.

14 M. Fiore and O. Mahmoud. Second-order algebraic theories. In *Proc. of MFCS'10*, LNCS 6281, pages 368–380, 2010.

15 M. Fiore and S. Staton. Substitution, jumps, and algebraic effects. In *Proc. of LICS'14*, 2014.

16 M. P. Fiore and M. D. Campos. The algebra of directed acyclic graphs. In *Computation, Logic, Games, and Quantum Foundations*, LNCS 7860, pages 37–51, 2013.

17 N. Ghani, M. Hamana, T. Uustalu, and V. Vene. Representing cyclic structures as nested datatypes. In *Proc. of TFP'06*, pages 173–188, 2006.

18 M. Hamana. Universal algebra for termination of higher-order rewriting. In *Proc. of RTA'05*, LNCS 3467, pages 135–149, 2005.

19 M. Hamana. Higher-order semantic labelling for inductive datatype systems. In *Proc. of PPDP'07*, pages 97–108. ACM Press, 2007. doi:10.1145/1273920.1273933.

20 M. Hamana. Initial algebra semantics for cyclic sharing tree structures. *Logical Methods in Computer Science*, 6(3), 2010.

21 M. Hamana. Iteration algebras for UnQL graphs and completeness for bisimulation. In *Proc. of Fixed Points in Computer Science (FICS'15)*, Electronic Proceedings in Theoretical Computer Science 191, pages 75–89, 2015.

22 M. Hamana, K. Matsuda, and K. Asada. The algebra of recursive graph transformation language UnCAL: Complete axiomatisation and iteration categorical semantics. submitted.

23 M. Hasegawa. *Models of Sharing Graphs: A Categorical Semantics of let and letrec.* PhD thesis, University of Edinburgh, 1997.

24 S. Hidaka, Z. Hu, K. Inaba, H. Kato, K. Matsuda, and K. Nakano. Bidirectionalizing graph transformations. In *Proc. of ICFP 2010*, pages 205–216, 2010.

25 A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):447–468, 1996.

26 K. Matsuda and K. Asada. Graph transformation as graph reduction: A functional reformulation of graph-transformation language UnCAL. Technical Report GRACE-TR 2015-01, National Institute of Informatics, January 2015.

27 L. G. L. T. Meertens. Paramorphisms. *Formal Asp. Comput.*, 4(5):413–424, 1992.

28 E. Meijer, M. M. Fokkinga, and R. Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In *Proc of FPCA'91*, pages 124–144, 1991.

29 R. Milner. A complete inference system for a class of regular behaviours. *J. Comput. Syst. Sci.*, 28(3):439–466, 1984.

30 B. C.d.S. Oliveira and W. R. Cook. Functional programming with structured graphs. In *Proc. of ICFP'12*, pages 77–88, 2012.

31 A. K. Simpson and G. D. Plotkin. Complete axioms for categorical fixed-point operators. In *Proc. of LICS'00*, pages 30–41, 2000.

32 S. Staton. An algebraic presentation of predicate logic. In *Proc. of FOSSACS 201*, pages 401–417, 2013.

33 S. Staton. Algebraic effects, linearity, and quantum programming languages. In *Proc. of POPL'15*, pages 395–406, 2015.

34 G. Winskel. *The Formal Semantics of Programming Languages.* The MIT Press, 1993.

Non-ω-Overlapping TRSs are UN

Stefan Kahrs[1] **and Connor Smith**[2]

1 **School of Computing, University of Kent, Canterbury, United Kingdom**
S.M.Kahrs@kent.ac.uk

2 **School of Computing, University of Kent, Canterbury, United Kingdom**
cls204@kent.ac.uk

Abstract

This paper solves problem #79 of RTA's list of open problems [14] – in the positive. If the rules of a TRS do not overlap w.r.t. substitutions of infinite terms then the TRS has unique normal forms. We solve the problem by reducing the problem to one of consistency for "similar" constructor term rewriting systems. For this we introduce a new proof technique. We define a relation $\Downarrow$ that is consistent by construction, and which – if transitive – would coincide with the rewrite system's equivalence relation $=_R$.

We then prove the transitivity of $\Downarrow$ by coalgebraic reasoning. Any concrete proof for instances of this relation only refers to terms of some finite coalgebra, and we then construct an equivalence relation on that coalgebra which coincides with $\Downarrow$.

Keywords and phrases consistency, omega-substitutions, uniqueness of normal forms

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.22

1 Introduction

For over 40 years [13] it has been known that TRSs that are left-linear and non-overlapping are confluent, and for over 30 years [8] that non-overlapping on its own may not even give us unique normal forms:

▶ **Example 1.** By Huet [8]: $\{F(x,x) \to A, F(x,G(x)) \to B, C \to G(C)\}$. The term $F(C,C)$ possesses two distinct normal forms, A and B.

However, in a certain sense the first two rules overlap semantically: the infinite term $G(G(\cdots))$ provides such an overlap, and in the world of infinitary rewriting [9] the term C even rewrites to that term in the limit.

The notion of overlap is based on the notion of substitution. By changing the codomain of the substitutions of concern from the set of finite terms to the set of infinitary (finite or infinite) ones we arrive at the notion of ω-overlap.

This creates the question: *do non-ω-overlapping TRSs have unique normal forms*? This was first conjectured 27 years ago by Ogawa [11], with an incomplete proof, and the problem is still listed as open problem 79 in RTA's list of open problems.

When making the step from a rewrite relation $\to_R$ to its equivalence closure $=_R$ one is typically interested in its consistency [3, p32ff], i.e. are there terms t, u such that $\neg(t =_R u)$?

Both uniqueness of normal forms (UN) and consistency (CON) can be looked at as properties of open terms or ground terms. We stick in this paper to the versions on open terms, as these notions are unaffected by signature extensions; for the versions on ground terms, UN can be lost and CON gained when we extend the signature. Moreover, on open terms UN implies CON.

For non-ω-overlapping systems UN and CON are closely related, as we can extend non-UN systems in a seemingly harmless way to make them fail CON too:

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 22; pp. 22:1–22:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Example 2.** Add to the system of Example 1 the rewrite rules $H(A,x,y) \to x$ and $H(B,x,y) \to y$. The system remains non-overlapping but it is now inconsistent.

Even if a TRS is non-ω-overlapping, the reduction relation $\to_R$ may still not be confluent (and so we need a different approach to show consistency); this follows from a well-known example by Klop [10]:

▶ **Example 3.** $\{A \to C(A), C(x) \to D(x, C(x)), D(x,x) \to E\}$.

In this system we have $A \to_R^* E$ and $A \to_R^* C(E)$, but $C(E)$ and E have no common reduct.

1.1 Translation of TRSs to Constructor TRSs

We are going to show how TRSs can be translated into Constructor TRSs, without affecting its equivalence in a substantial way, in particular: consistency is both preserved and reflected by the translation, as is strong normalisation.

The translation works by (i) doubling up the signature, so that for each function symbol F we have both a constructor version F_c and a destructor F_d; (ii) translate the rewrite rules to make them comply with the regime of Constructor TRSs; (iii) add further rules that make former patterns regain pattern status.

▶ **Example 4.** If we take the rewrite rules of Combinatory Logic, $A(A(K,x),y) \to x$ and $A(A(A(S,x),y),z) \to A(A(x,z),A(y,z))$ and apply the translation, we end up with the following system:

$$\begin{array}{cc}
A_d(A_c(K_c,x),y) \to x & A_d(A_c(A_c(S_c,x),y),z) \to A_d(A_d(x,z),A_d(y,z)) \\
A_d(K_c,x) \to A_c(K_c,x) & A_d(S_c,x) \to A_c(S_c,x) \\
K_d \to K_c & A_d(A_c(S_c,x),y) \to A_c(A_c(S_c,x),y) \\
 & S_d \to S_c
\end{array}$$

The top two rules are the translated versions of the original rules, the ones below are their respective pattern rules.

In Example 4, an orthogonal TRS was translated into an orthogonal Constructor TRS. In general, this will not quite be the case, and non-ω-overlapping TRSs will not remain non-ω-overlapping either. However, all overlaps created by the translation are benign.

1.2 Consistency of Constructor Rewriting

At the heart of our overall proof is showing (for our rewrite systems in question) that the equivalence closure $=_R$ of single rewrite steps is a subrelation of a consistent relation $\Downarrow$ and therefore itself consistent. This relation $\Downarrow$ is defined using slightly stronger closure principles than those that characterise the joinability relation $\downarrow$, however they remain weak enough to ensure (for arbitrary TRSs) that $\Downarrow$ is consistent. Because $\Downarrow$ is closed under the same operations as $=_R$, *except for transitivity*, proving consistency of $=_R$ can be reduced to proving that $\Downarrow$ is transitive.

Our proof idea is then based on the following fundamental observations: (i) (inductive, finitely-branching) proofs are finite objects, (ii) therefore each proof can only refer to finitely many terms. Instead of asking the question: "is $t \Downarrow u$ true?" we consider its provability relative to some finite set of terms A ($t \Downarrow_A u$); we need A to be closed under subterms which implies that it is a coalgebra of the signature. We show that – provided the TRS is "suitably well-behaved" – such finite coalgebras give rise to a single structure one might call a *universal proof* for A that proves $t \Downarrow_A u$ whenever it holds. This universal proof also exhibits the property that $\Downarrow_A$ is an equivalence relation. We have that $t \Downarrow u$ iff $t \Downarrow_A u$ for some finite A. Since these coalgebras are closed under union, and $A \subseteq B \wedge t \Downarrow_A u \Rightarrow t \Downarrow_B u$, we have that $\Downarrow$ itself is transitive.

2 Preliminaries

We assume familiarity with the standard notions of term rewriting and infinitary term rewriting [18, 2], but use this section to fix some notation.

A signature Σ is a pair $(\mathcal{F}, \#)$ comprising a set $\mathcal{F}$ of function symbols and a function $\# : \mathcal{F} \to \mathcal{N}$ assigning to each function symbol an arity. We write $\mathit{Ter}(\Sigma, X)$ for the set of finite terms over the variable set X, and $\mathit{Ter}^{\omega}(\Sigma, X)$ and $\mathit{Ter}^{\infty}(\Sigma, X)$ for the corresponding sets of rational and infinitary terms. Given a rewrite rule $l \to r$ we write $\xrightarrow{l \to r}$ for the substitutive closure of the rule, and $\xrightarrow{\epsilon}$ for the union of $\xrightarrow{l \to r}$ for all rewrite rules $l \to r$ of a TRS.

We say that a relation R on $\mathit{Ter}(\Sigma, X)$ is consistent if $\forall x, y \in X.\, x \mathrel{R} y \Rightarrow x = y$. We say that a TRS is consistent (has the CON property) if the congruence closure of $\xrightarrow{\epsilon}$ is consistent on $\mathit{Ter}(\Sigma, Y)$, for an infinite set Y.

A substitution is a map $\sigma : V \to \mathit{Ter}(\Sigma, X)$ which we homomorphically extend to $\mathit{Ter}(\Sigma, V) \to \mathit{Ter}(\Sigma, X)$. Two terms $t \in \mathit{Ter}(\Sigma, V), u \in \mathit{Ter}(\Sigma, W)$ are said to be *unifiable* iff there is a pair of substitutions $\sigma : V \to \mathit{Ter}(\Sigma, X)$, $\theta : W \to \mathit{Ter}(\Sigma, X)$ such that $\sigma(t) = \theta(u)$. A pair of terms is said to be *ω-unifiable* if these conditions hold for substitutions with infinite terms in their codomain. Unifiability implies ω-unifiability, as all finite terms inhabit the infinite term universe as well.

As an aside, ω-unifiability of finite terms coincides with their unifiability w.r.t. substitutions with rational terms. This was first studied by Huet [7], and is these days usually implemented via union/find structures [16], which incidentally provide some inspiration for the notion of "proof graph" we consider later on.

2.1 Constructor Rewriting

A TRS is a *Constructor TRS* if the signature Σ is a *constructor signature*, i.e. it splits into two disjoint subsignatures Σ_c and Σ_d such that for any rewrite rule $F(p_1, \ldots, p_n) \to r$ we have $F \in \Sigma_d$ and $p_1, \ldots, p_n \in \mathit{Ter}(\Sigma_c, X)$.

The standard notion of non-overlapping TRSs is based on the notion of unifiability, and it can be simplified for Constructor TRSs. A Constructor TRS is non-overlapping iff the left-hand sides of any two different rules are not unifiable. Replacing 'unifiability' in that setting with 'ω-unifiability' provides the analogous (stronger) notion of non-ω-overlapping. A similar notion is that of *almost* non-ω-overlapping TRSs, which means that non-variable proper subterms of left-hand sides of rules are not ω-unifiable with left-hand sides of rules, and that $\xrightarrow{\epsilon}$ is deterministic.

2.2 Term-Coalgebras

In order to consider coalgebras of signatures Σ we would have to view signatures as functors on the category Set. However, we only need the following special instance of this concept later, which helps to keep the proofs short:

▶ **Definition 5.** Given a signature Σ, a *term-coalgebra* is a set $A \subseteq \mathit{Ter}^\infty(\Sigma, \emptyset)$ which is closed under subterms. It is called *finite* if it is a finite set, and *strongly finite* if in addition $A \subseteq \mathit{Ter}(\Sigma, \emptyset)$. We refer to the elements of a coalgebra as *nodes*.

More generally, Σ-coalgebras A would be characterized by a function $\upsilon : A \to \Sigma(A)$ which maps a node to a structure containing its root function symbol and the list of its subnodes. In that setting two nodes are bisimilar if their repeated unfolding via υ yield the same infinitary term. In a term-coalgebra this is unique, so bisimilarity coincides there with equality.

We also allow for variables in term-coalgebras by "freezing" them, i.e. using the canonical isomophism between $\mathit{Ter}^\infty(\Sigma_d + \Sigma_c, X)$ and $\mathit{Ter}^\infty(\Sigma_d + (\Sigma_c + X), \emptyset)$. Thus, when considered as a member of a term-coalgebra a variable is a nullary constructor. For heterogeneous relations between term-coalgebras we must therefore have that the variable set X is the same, so that they are coalgebras of the same functor. Relations between term-coalgebras can be consistent simply due to the lack of variables occurring in them as nodes, or indeed any other nodes: the empty set is a term-coalgebra that can only be consistently related to other term-coalgebras.

2.3 Relational Algebra

We use some standard constructions from relational algebra; in particular, we write $R \cdot S$ for relational composition in diagrammatical order, i.e. $a\ (R \cdot S)\ b \iff \exists c.\, a\ R\ c \wedge c\ S\ b$. As constants, we also use the empty relation $\emptyset$, and the identity relation id.

Binary relations on any set form a complete lattice, and so Tarski's fixpoint theorems [17] apply – any monotonic function f on these relation domains has a smallest fixpoint, $\mu(f)$, and a largest fixpoint $\nu(f)$. One usually writes $\mu x.f(x)$ for $\mu(\lambda x.f(x))$, etc. Most operations in relational algebra are monotonic (with the notable exception of complement, which we are not using here), as are the smallest/largest fixpoint constructions themselves [1, Proposition 1.2.18]. Thus any composition of these operations will result in a monotonic function on relations that therefore has both of these fixpoints. In the following, we will tacitly exploit that any function arrived by these means is monotonic.

▶ **Definition 6.** A predicate P on a complete lattice L is called *sup-continuous* iff for any function $f : I \to L$ such that $\forall x \in I.\, P(f(x))$ we also have $P(\bigsqcup_{i \in I} f(i))$.

Note that – as the definition also applies when the index set is empty – we would also necessarily have $P(\bot)$.

▶ **Proposition 7.** *Let P be a sup-continuous predicate on a complete lattice L, and f a monotonic function on L that preserves P, i.e. $\forall x \in L.\, P(x) \Rightarrow P(f(x))$. Then $P(\mu x.f(x))$.*

Proof. This follows from [1, Theorem 1.2.11]. That theorem defines for any ordinal β, $x_\beta = \bigsqcup\{f(x_\alpha) \mid \alpha < \beta\}$, and shows that for some β that is sufficiently large $x_\beta = \mu x.f(x)$. Hence the result follows by ordinal induction on the ordinal $\beta + 1$. ◀

3 Constructor Translation

We first demonstrate that a TRS can, in a sense, be viewed as a Constructor TRS, by translating it into a Constructor TRS with similar properties. This similarity is particularly strong for non-ω-overlapping TRSs.

To translate TRSs we use the concept of signature morphism – see [15] for a more general and modern version of the concept; we specialise it here for the standard signatures used in TRSs, as this concept rarely shows up in term rewriting literature.

▶ **Definition 8.** A signature morphism between signatures $\Sigma = (\mathcal{F}_\Sigma, \#_\Sigma)$ and $\Theta = (\mathcal{F}_\Theta, \#_\Theta)$ is a function $f : \mathcal{F}_\Sigma \to \mathcal{F}_\Theta$ such that $\#_\Theta(f(G)) = \#_\Sigma(G)$. Each signature morphism $f : \Sigma \to \Theta$ induces a map $T_f : \mathit{Ter}(\Sigma, X) \to \mathit{Ter}(\Theta, X)$ given as $T_f(F(t_1, \ldots, t_n)) = f(F)(T_f(t_1), \ldots, T_f(t_n))$ and $T_f(x) = x$ for $x \in X$.

Signatures and signature morphisms form a category, and this category clearly has coproducts, given by the disjoint union of signatures.

▶ **Definition 9.** Given a signature Σ we write $\Sigma 2$ for the coproduct $\Sigma + \Sigma$, which we view as a constructor signature; the images of Σ under the injections ι_1 and ι_2 give us Σ_c and Σ_d, respectively. We write F_c and F_d for the function symbols $\iota_1(F)$ and $\iota_2(F)$, respectively. We use the abbreviations $\lfloor t \rfloor$ for $T_{\iota_1}(t)$ and $\lceil t \rceil$ for $T_{\iota_2}(t)$.

So $\Sigma 2$ contains two copies of every function symbol, one as a constructor, and one as a destructor. The two embedding signature morphisms induce two different embeddings of terms, labelling all symbols as constructors or destructors, respectively.

▶ **Definition 10.** Let $\gamma : \Sigma 2 \to \Sigma$ be the signature morphism $[id, id]$, i.e. $\gamma(F_c) = F$, $\gamma(F_d) = F$. We write $|t|$ for $T_\gamma(t)$.

Thus, given a "labelled" term $t \in \mathit{Ter}(\Sigma 2, X)$, $|t| \in \mathit{Ter}(\Sigma, X)$ is the term we get when we erase the labels from t. Clearly, we have $|\lceil t \rceil| = t = |\lfloor t \rfloor|$, but no corresponding property when we go the other way, e.g. u and $\lceil |u| \rceil$ can differ.

▶ **Definition 11.** Given a TRS $T = (\Sigma, R)$, a *pattern* is a proper subterm of the left-hand side of a rule in R. We write $\mathrm{Pat}(T)$ for the set of all patterns of the TRS T.

Recall that Constructor TRSs are characterised by having all their patterns confined to $\mathit{Ter}(\Sigma_c, X)$. Therefore, patterns play a special role in the translation of TRSs into Constructor TRSs:

▶ **Definition 12.** Let T be a TRS with ruleset R and signature Σ. The *constructor translation of T* is a Constructor TRS $T' = (\Sigma 2, R')$ built as follows. $R' = R'_d \cup R'_c$, where $R'_d = \{F_d(\lfloor t_1 \rfloor, \ldots, \lfloor t_n \rfloor) \to \lceil r \rceil \mid F(t_1, \ldots, t_n) \to r \in R\}$ and $R'_c = \{F_d(\lfloor t_1 \rfloor, \ldots, \lfloor t_n \rfloor) \to F_c(\lfloor t_1 \rfloor, \ldots, \lfloor t_n \rfloor) \mid F(t_1, \ldots, t_n) \in \mathrm{Pat}(T)\}$.

Any rule of the original TRS becomes a rule in R'_d by turning its patterns into constructor patterns, and every non-variable pattern of T becomes a rule in R'_c. We have already seen the translation of Combinatory Logic (Example 4) as an example for this translation in the introduction. For simplicity, the constructor translation does not make a distinction which symbols already acted like constructors, such as the constants K and S.

We can relate a TRS to its constructor translation. First, when terms lose pattern status via the destructor translation then they can regain it through rewriting:

▶ **Lemma 13.** *Let T be a TRS and T' its constructor translation. For any $p \in \mathrm{Pat}(T)$ we have $\lceil p \rceil \to^*_{T'} \lfloor p \rfloor$.*

Proof. By induction on the term structure of p. If p is a variable then $\lceil p \rceil = \lfloor p \rfloor$. Otherwise, $p = F(t_1, \ldots, t_n)$ and $\lceil p \rceil = F_d(\lceil t_1 \rceil, \ldots, \lceil t_n \rceil)$. By induction hypothesis $\lceil t_i \rceil \to^*_{R'} \lfloor t_i \rfloor$, for all i. Therefore, $F_d(\lceil t_1 \rceil, \ldots, \lceil t_n \rceil) \to^*_{T'} F_d(\lfloor t_1 \rfloor, \ldots, \lfloor t_n \rfloor)$.

Moreover, $F_d(\lfloor t_1 \rfloor, \ldots, \lfloor t_n \rfloor) \to F_c(\lfloor t_1 \rfloor, \ldots, \lfloor t_n \rfloor)$ is a rule in R' and therefore overall $\lceil p \rceil = F_d(\lceil t_1 \rceil, \ldots, \lceil t_n \rceil) \to^*_{T'} F_d(\lfloor t_1 \rfloor, \ldots, \lfloor t_n \rfloor) \to_{T'} F_c(\lfloor t_1 \rfloor, \ldots, \lfloor t_n \rfloor) = \lfloor p \rfloor$. ◀

▶ **Lemma 14.** *Let T be a TRS and T' its constructor translation. If $t \to_T u$ then $\lceil t \rceil \to^+_{T'} \lceil u \rceil$. If $p \to_{T'} q$ then $\lfloor p \rfloor \to_T \lfloor q \rfloor \vee \lfloor p \rfloor = \lfloor q \rfloor$.*

Proof. If $t \to_T u$ then we must have that for some context C, substitution σ and rewrite rule $F(p_1, \ldots, p_n) \to r$ in T, $t = C[F(\sigma(p_1), \ldots, \sigma(p_n))]$ and $u = C[\sigma(r)]$. We clearly have $\lceil t \rceil = \lceil C \rceil [F_d(\lceil \sigma \rceil(\lceil p_1 \rceil), \ldots, \lceil \sigma \rceil(\lceil p_n \rceil))]$ and $\lceil u \rceil = \lceil C \rceil [\lceil \sigma \rceil(\lceil r \rceil)]$, where $\lceil C \rceil$ and $\lceil \sigma \rceil$ are straightforward extensions of the signature morphism to contexts and substitutions. By Lemma 13 we have $\lceil p_i \rceil \to^*_{T'} \lfloor p_i \rfloor$, hence by substitutivity of rewriting $\lceil \sigma \rceil(\lceil p_1 \rceil) \to^*_{T'} \lceil \sigma \rceil(\lfloor p_i \rfloor)$. Compatibility of rewriting gives us $\lceil t \rceil \to^*_{T'} \lceil C \rceil [F_d(\lceil \sigma \rceil(\lfloor p_1 \rfloor), \ldots, \lceil \sigma \rceil(\lfloor p_n \rfloor))]$. The latter term then rewrites in one step to $\lceil u \rceil$.

In the case of $p \to_{T'} q$ a rewrite step with a rule from R'_c gives us $\lfloor p \rfloor = \lfloor q \rfloor$, otherwise it is a translated rule from the old system and we have $\lfloor p \rfloor \to_T \lfloor q \rfloor$. ◀

▶ **Proposition 15.** *Let T be a TRS and T' be its constructor translation. For $t, u \in Ter(\Sigma, X)$, if $t =_T u$ then $\lceil t \rceil =_{T'} \lceil u \rceil$. For $p, q \in Ter(\Sigma 2, Y)$, if $p =_{T'} q$ then $\lfloor p \rfloor =_T \lfloor q \rfloor$.*

Proof. Either way we split the equational proof into individual rewrite steps, and then rebuild these using Lemma 14. ◀

Proposition 15 tells us that we can translate equations back and forth between a TRS and its constructor translation. This has a consequence on consistency.

▶ **Corollary 16.** *Let T be a TRS and T' its constructor translation. Then T is consistent iff T' is.*

Proof. If, say, T is inconsistent, then $x =_T y$, for distinct variables x and y. By Proposition 15 we have $\lceil x \rceil =_{T'} \lceil y \rceil$. But $\lceil x \rceil = x$ and $\lceil y \rceil = y$ and so T' is inconsistent too. The other direction is analogous. ◀

So, the constructor translation preserves and reflects consistency – in the following we really only need that it reflects that property. Aside: regarding termination and confluence, the constructor translation preserves and reflects the former, but only reflects the latter. In general, it does not even preserve weak confluence.

Notice that our construction can fail to produce an almost non-ω-overlapping TRS if our original TRS was merely almost-non-ω-overlapping.

▶ **Example 17.** Consider the following rules describing an if-and-only-if operator on the Booleans: $\{Iff(F, x) \to N(x), Iff(x, F) \to N(x), Iff(x, x) \to N(F), N(N(F)) \to F\}$.

The system is almost non-ω-overlapping, with trivial overlaps between any of the first three rules. However, the constructor translation makes some of the trivial overlaps non-trivial, because F becomes F_c on the left and F_d on the right.

4 Strongly Almost non-ω-overlapping Constructor TRSs

In the Introduction we were mentioning that overlaps created by the constructor translation are "benign". We will now characterise how benign they are more precisely.

▶ **Definition 18.** Two rewrite rules $l_1 \to r_1$, $l_1, r_1 \in Ter(\Sigma, X)$, and $l_2 \to r_2$, $l_2, r_2 \in Ter(\Sigma, Y)$, have a *common generalisation* $l_3 \to r_3$ iff there are substitutions $\sigma_1 : Z \to Ter(\Sigma, X)$, $\sigma_2 : Z \to Ter(\Sigma, Y)$ such that:

- $\sigma_1(l_3) = l_1$ and $\sigma_2(l_3) = l_2$, and $\sigma_1(r_3) = r_1$ and $\sigma_2(r_3) = r_2$,
- all variables in r_3 occur in l_3.

The idea goes back to Plotkin's concept of generalisation and anti-unifiers [12]. Indeed we can check whether two rules have a common generalisation by computing the anti-unifier of the terms $R(l_1, r_1)$ and $R(l_2, r_2)$, and then checking whether the result – which must have the form $R(l_3, r_3)$ – satisfies the final condition on variables.

▶ **Lemma 19.** *If two rewrite rules of a Constructor TRS have a common generalisation $l_3 \to r_3$ then this is either a legal rewrite rule for a Constructor TRS over the same signature, or l_3 is a variable.*

Proof. Proper subterms of l_3 must be constructor terms, otherwise $\sigma_1(l_3) = l_1$ must have non-constructor subterms, contradicting the premise. ◀

We do not need the concrete rewrite system a common generalisation would be part of; all we need is that the rule behaves like a rewrite rule from a Constructor TRS.

▶ **Definition 20.** A TRS is called *strongly almost non-ω-overlapping* iff (i) all ω-overlaps are in root position, (ii) whenever two left-hand sides are ω-unifiable then their rules have a common generalisation.

To justify the chosen terminology:

▶ **Proposition 21.** *Any non-ω-overlapping TRS is strongly almost non-ω-overlapping. Any strongly almost non-ω-overlapping TRS is almost non-ω-overlapping.*

Proof. A non-ω-overlapping TRS clearly satisfies the conditions of being strongly non-ω-overlapping, as its rules can only overlap with themselves, and that at the root.

For the second part, assume we have a strongly almost non-ω-overlapping TRS. Let $\langle\theta_1, \theta_2\rangle$ be a ω-unifier for the left-hand sides l_1, l_2. Then we have $\theta_1(l_1) = (\theta_1 \circ \sigma_1)(l_3)$, and similarly $\theta_2(l_2) = (\theta_2 \circ \sigma_2)(l_3)$. Thus, the composite substitutions $\theta_1 \circ \sigma_1$ and $\theta_2 \circ \sigma_2$ must agree on all variables occurring in l_3. The variable condition on r_3 then gives us $(\theta_1 \circ \sigma_1)(r_3) = (\theta_2 \circ \sigma_2)(r_3)$, and as $(\theta_2 \circ \sigma_2)(r_3) = \theta_2(r_2)$ and $(\theta_1 \circ \sigma_1)(r_3) = \theta_1(r_1)$ we have that $\langle\theta_1, \theta_2\rangle$ is also a ω-unifier for the right-hand sides r_1, r_2. ◀

▶ **Proposition 22.** *The constructor translation of an almost non-ω-overlapping TRS is strongly almost non-ω-overlapping.*

Proof. Because the constructor translation produces a Constructor TRS all ω-overlaps between left-hand sides of rules, if any, are at root position. Let $l_1 \to r_1$ and $l_2 \to r_2$ be two rules in the constructor translation R', such that l_1 and l_2 are ω-unifiable. That implies that $|l_1|$ and $|l_2|$ are ω-unifiable too.

If both rules are translated rules then we can only avoid a contradiction by $|l_1| = |l_2|$ and $|r_1| = |r_2|$ which implies $l_1 = l_2$ and $r_1 = r_2$, as the translation of rules is injective.

If the first rule is a translated rule and the second a pattern rule then $|l_2|$ is a non-variable subterm of a left-hand side in R, and is ω-unifiable with $|l_1|$ which contradicts our assumption about R.

If both rules are pattern rules then both rules clearly have the common generalisation $F_d(x_1, \ldots, x_n) \to F_c(x_1, \ldots, x_n)$, where F is the root symbol of the pattern. ◀

Note: it is not generally true that the constructor translation of a strongly almost non-ω-overlapping TRS is itself strongly almost non-ω-overlapping, because the constructor translation breaks the sharing of subterms of left-hand and right-hand sides, e.g. for the rules $F(C(x), y) \to G(C(x))$ and $F(y, B) \to G(y)$ – the common generalisation $F(y, z) \to G(y)$ of the two rules is not preserved by the constructor translation. One could fix this by providing a more sophisticated translation that maintains the sharing of common subexpressions between left-hand and right-hand sides.

5 Reasoning with Term-Coalgebras

The main purpose of this section is to establish some tools to reason about consistency. These tools are largely relation-algebraic, for relations operating on term-coalgebras, though they could be generalised to arbitrary Σ-coalgebras.

As an additional ingredient to define relations between or on term-coalgebras for a signature Σ we use the following notation: if $R \subset A \times B$, where A and B are term-coalgebras A and B then $\widetilde{R} \subseteq A \times B$ is defined as follows:

$$\forall t \in A.\, \forall u \in B.\, t\ \widetilde{R}\ u \iff \exists F \in \Sigma.\, \exists a_1, \ldots, a_n \in A.\, \exists b_1, \ldots, b_n \in B.$$
$$t = F(a_1, \ldots, a_n) \wedge u = F(b_1, \ldots, b_n) \wedge \forall i.\, a_i\ R\ b_i$$

This concept was first used in [5, 6]; we modified it slightly by removing the reflexivity case. For constructur signatures, we use the notations $\overline{R}$ and $\widehat{R}$ to mean $\widetilde{R}$ for the subsignatures Σ_d and Σ_c, respectively. In particular, $t\ \widehat{id}\ t$ iff the root symbol of t is a constructor, and so $\widehat{R} \cdot \overline{S} = \emptyset$. We still use $\widetilde{R}$ for constructor signatures, to refer to the combined signature; hence $\widetilde{R} = \overline{R} \cup \widehat{R}$.

One can generalise this to arbitrary Σ-coalgebras where the conditions $t = F(a_1, \ldots, a_n)$ and $u = F(b_1, \ldots, b_n)$ would be replaced by $\upsilon_A(t) = F(a_1, \ldots, a_n)$ and $\upsilon_B(u) = F(b_1, \ldots, b_n)$ where υ_A and υ_B are the unfolding maps of their respective coalgebras.

▶ **Proposition 23.** *Some general relation-algebraic properties of $\widetilde{R}$:*

1. $\widetilde{R \cap S} = \widetilde{R} \cap \widetilde{S}$, *which moreover implies that the function $x \mapsto \widetilde{x}$ is monotonic.*
2. $\widetilde{R^{-1}} = \widetilde{R}^{-1}$.
3. $\widetilde{R \cdot S} \supseteq \widetilde{R} \cdot \widetilde{S}$. *Therefore also: $\widetilde{R^*} \supseteq \widetilde{R}^*$.*

Proof. Trivial. ◀

We have $\widetilde{R \cup S} \supseteq \widetilde{R} \cup \widetilde{S}$ by monotonicity, and it is not an equation because a signature can contain function symbols of arity greater than 1. Also, the relation $\widehat{\emptyset}$ is generally not the empty relation – it will relate all bisimilar nodes that have no subnodes and are topped with a constructor, and therefore also variables.

For term-coalgebras we have $id = \widetilde{id}$, but arbitrary Σ-coalgebras would only give us $id \subseteq \widetilde{id}$, because a coalgebra can contain distinct bisimilar nodes with identical subnodes.

▶ **Definition 24.** A relation R between term-coalgebras is called Σ-closed iff $\widetilde{R} \subseteq R$.

Note: this is standard terminology taken from [2], except that we generalise it to coalgebras.

▶ **Definition 25.** Let $V = \wp(A \times B)$ be the set of relations between term-coalgebras A and B. Then the function $\mathrm{CT} : V \to V$ is defined by $\mathrm{CT}(R) \doteq \mu x.\, R \cup \widetilde{x}$. Thus $\mathrm{CT}(R)$ is the smallest Σ-closed relation containing R.

We can use the $\widetilde{R}$ notation to define $=_R$ in a relation-algebraic way:

▶ **Definition 26.** The inductive congruence closure $\mathrm{CGI}(R)$ of a relation R on a term-coalgebra is defined as: $\mathrm{CGI}(R) \doteq \mu x.\, R \cup x^{-1} \cup (x \cdot x) \cup id \cup \widetilde{x}$.

Thus $=_R$ is then $\mathrm{CGI}(\xrightarrow{\epsilon})$ on the coalgebra $\mathit{Ter}(\Sigma, X)$. Notice that for rewrite systems this is in general not the same as the equivalence closure of rewrite steps, because a coalgebra might lack the intermediate terms. To reason about pattern matching we will later need a stronger notion of consistency, that includes reasoning about constructors:

▶ **Definition 27.** A relation R between term-coalgebras is called *constructor-compatible* iff $\widehat{id} \cdot R \cdot \widehat{id} \subseteq \widehat{R}$.

▶ **Lemma 28.** *Every constructor-compatible relation R between any two term-coalgebras A and B is consistent.*

Proof. Let $x\ R\ y$ where x and y are variables. Since variables in term-coalgebras are viewed as constructors we have $x\ \widehat{id_A}\ x$ and $y\ \widehat{id_B}\ y$. Hence $x\ \widehat{id_A}\ x\ R\ y\ \widehat{id_B}\ y$. Constructor-compatibility of R then gives us $x\ \widehat{R}\ y$ which means $x = y$ since they have no subterms. ◀

▶ **Lemma 29.** *Constructor-compatible relations are closed under arbitrary union. Relational inverse also preserves constructor-compatibility.*

Proof. Let R_i with $i \in I$ be a family of constructor-compatible relations.

$$\widehat{id} \cdot (\bigcup_{i\in I} R_i) \cdot \widehat{id} = \bigcup_{i\in I} (\widehat{id} \cdot R_i \cdot \widehat{id}) \subseteq \bigcup_{i\in I} \widehat{R_i} \subseteq \widehat{\bigcup_{i\in I} R_i}$$

For inverse, for relations between term-coalgebras A and B:

$$\widehat{id_B} \cdot R^{-1} \cdot \widehat{id_A} = ((\widehat{id_B} \cdot R^{-1} \cdot \widehat{id_A})^{-1})^{-1} = (\widehat{id_A} \cdot R \cdot \widehat{id_B})^{-1} \subseteq \widehat{R}^{-1} = \widehat{R^{-1}}$$ ◀

Lemma 29 means that constructor-compatibility is a sup-continuous predicate on the lattice of binary relations between two coalgebras. We also have that any relation between term-coalgebras has a constructor-compatible interior – the union of all its subrelations that have this property.

▶ **Definition 30.** Given a Constructor TRS over a signature Σ, a *consistency invariant* is a consistent and Σ-closed relation S on a term-coalgebra A such that for any constructor-compatible equivalence $=_S \subseteq S$ we have $\xleftarrow{\epsilon} \cdot \overline{=_S} \cdot \xrightarrow{\epsilon} \subseteq \mathrm{CT}(=_S)$.

Explanation: if we have $a_1 \xleftarrow{\epsilon} a_2 \overline{=_S}\ a_3 \xrightarrow{\epsilon} a_4$ then the pair $\langle a_1, a_4 \rangle$ can be viewed as a form of "semantical critical pair", because it has been obtained by root-rewrite-steps from $\langle a_2, a_3 \rangle$ which share their root symbols and are "semantically equal" below the root. Thus a consistency invariant is characterised by the property that semantical critical pairs stay within the invariant. That this is relative to a term-coalgebra A matters insofar as rewrite steps with contracta outside A are simply discarded.

The reason $=_S$ is locally quantified in the definition of consistency invariant is that although constructor-compatible relations are closed under union, equivalence relations are not, so we cannot simply compute a suitable interior relation. The reason the definition uses constructor-compatible equivalences is that we can turn them into functions that unify the nodes in their equivalence classes.

▶ **Definition 31.** Given a term-coalgebra A, a function $f : A \to Ter^\infty(\Sigma, \emptyset)$ is called *constructor-preserving* iff

$$\forall a_1, \ldots a_n \in A. \forall C \in \Sigma_c.\ f(C(a_1, \ldots, a_n)) = C(f(a_1), \ldots, f(a_n))$$

▶ **Proposition 32.** *Let $=_\varrho$ be a constructor-compatible equivalence relation on a term-coalgebra A. Given some well-ordering on $=_\varrho$-equivalence classes, there is a function $U(=_\varrho) : A \to Ter^\infty(\Sigma, \emptyset)$ such that: (i) $U(=_\varrho)$ is constructor-preserving; (ii) $\forall a, b \in A.\ a =_\varrho b \Rightarrow U(=_\varrho)(a) = U(=_\varrho)(b)$.*

Proof. Let $\min D$ denote the minimum element of any non-empty subset D of any equivalence class w.r.t. that well-order. Let $[a] \subseteq A$ be the $=_\varrho$-equivalence class of some node $a \in A$. Let $B = \{b \in [a] \mid b \;\widehat{id}\; b\}$. Then we define $U(=_\varrho)(a)$ as follows:

$$U(=_\varrho)(a) = \begin{cases} C(U(=_\varrho)(b_1), \ldots, U(=_\varrho)(b_n)) & \text{if } B \neq \emptyset \wedge \min B = C(b_1, \ldots, b_n) \\ \min[a] & \text{if } B = \emptyset \end{cases}$$

Clearly, $U(=_\varrho)$ satisfies condition (ii) because its definition only depends on the equivalence class $[a]$, not on a directly. Let $c \in [a]$ be constructor-topped, i.e. $c \;\widehat{id}\; c$. Thus we have $c \;\widehat{id}\; c =_\varrho \min B \;\widehat{id}\; \min B$: and constructor-compatibility of $=_\varrho$ gives us: $c \;\widehat{=_\varrho}\; \min B$. Therefore $c = C(c_1, \ldots, c_n)$ and $c_i =_\varrho b_i$ and the result follows. ◀

Notice that even if the coalgebra A only contains finite terms the function $U(=_\varrho)$ may still have infinite terms in its range; e.g. this would be the case for the equivalence class $[K, C(K)]$ if C is a constructor.

6 Rewrite-Related Reasoning

We now study some properties of our relations in the presence of pattern matching and rewrite rules. The aim is to establish invariants that "survive" the parallel application of rewrite rules at the root of a term (node).

Besides giving us an ω-unifier (for equivalences), constructor-compatible relations give us an invariant in pattern matching:

▶ **Lemma 33.** *Let $t \in Ter(\Sigma_c, X)$, $s = \sigma(t)$, $u = \theta(t)$, and $s \mathrel{R} u$ where R is a constructor-compatible relation between two term-coalgebras A and B. Then for any $x \in X$ that occurs in t, $\sigma(x) \mathrel{R} \theta(x)$.*

Proof. Let $x \in X$ be any variable occurring in t, i.e. there is some position $p \in \mathrm{Pos}(t)$ such that $t|_p = x$. The proof goes by induction on the length of p.

If $p = \langle\,\rangle$ (the empty position) then $\sigma(t) = \sigma(x)$; similarly, $\theta(u) = \theta(x)$, and so the result follows.

Otherwise, $p = i \cdot p'$ and $t = C(t_1, \ldots, t_n)$, for some constructor $C \in \Sigma_c$. $s = \sigma(t)$ implies $s = C(\sigma(t_1), \ldots, \sigma(t_n))$. Similarly, $u = C(\theta(t_1), \ldots, \theta(t_n))$. Thus, s and u are both constructor-topped, therefore $s \mathrel{R} u$ implies $s \mathrel{\widehat{R}} u$ by constructor-compatibility of R. Hence $\sigma(t_i) \mathrel{R} \theta(t_i)$, and we can apply the induction hypothesis to t_i w.r.t. position p'. ◀

For applying a substitution after matching we have the following result:

▶ **Lemma 34.** *Let R be a Σ-closed relation between two term-coalgebras A and B. Let $t \in Ter(\Sigma, X)$, $s = \sigma(t) \in A$, $u = \theta(t) \in B$. If for all variables $x \in X$ that occur in t we have $\sigma(x) \mathrel{R} \theta(x)$ then $s \mathrel{R} u$.*

Proof. By induction on the term structure of t. If $t \in X$ (it is a variable) then $s = \sigma(t) \mathrel{R} \theta(t) = u$ and the result follows from the assumption.

Otherwise, $t = F(t_1, \ldots, t_n)$, $s = F(\sigma(t_1), \ldots, \sigma(t_n))$, $u = F(\theta(t_1), \ldots, \theta(t_n))$. By induction hypothesis we have $s_i \mathrel{R} u_i$ (for all i), thus $s \mathrel{\widetilde{R}} u$ which entails the result, because R is Σ-closed. ◀

As a direct consequence we can characterise "how safe" parallel rewrite steps with the same rule are in a constructor rewrite system:

▶ **Corollary 35.** *Let S be constructor-compatible, and let $l \to r$ be a rewrite rule of a Constructor TRS. If $t_1 \xleftarrow{l\to r} t_2 \; \overline{S} \; t_3 \xrightarrow{l \to r} t_4$ then t_1, t_4 are related by* $\mathrm{CT}(S)$.

Proof. Because l is the left-hand side of a rule of a Constructor TRS, its direct subterms are constructor terms. Thus Lemma 33 applies: the matching substitutions are pointwise related by S. Hence they are also related by $\mathrm{CT}(S)$ and the result then follows from Lemma 34. ◀

Besides non-determinism of $\xrightarrow{\epsilon}$ what might also introduce constructor-inconsistency if between root-rewrite steps we rewrite on subterms in some way that might allow the adjacent root steps to create an inconsistency. However, the situations allowing us to rewrite in opposite directions are limited by the following observation:

▶ **Lemma 36.** *Let $a \to b, c \to d$ be two rewrite rules of a Constructor TRS. Let $=_\varrho$ be a constructor-compatible equivalence on some term-coalgebra A. If $t_1 \xleftarrow{a \to b} t_2 \; \overline{=_\varrho} \; t_3 \xrightarrow{c \to d} t_4$ then a and c are ω-unifiable.*

Proof. We know $t_2 = \sigma(a)$ and $t_3 = \theta(c)$ for some substitutions σ and θ. We know from Proposition 32 that $U(=_\varrho)$ maps the direct subnodes of t_2 and t_3 to the same result.

Because that function is constructor-preserving we have $U(=_\varrho)(t_2|_i) = U(=_\varrho)(\sigma(a|_i)) = (U(=_\varrho) \circ \sigma)(a|_i)$ and similarly $U(=_\varrho)(t_3|_i) = (U(=_\varrho) \circ \theta)(c|_i)$. Thus the pair of maps $\langle U(=_\varrho) \circ \sigma, U(=_\varrho) \circ \theta \rangle$ is an ω-unifier for a and c. ◀

Lemma 36 means that if the reasoning between root steps is done safely, i.e. via a constructor-compatible equivalence on subterms then the subsequent rewrite steps were done with rules with ω-overlapping patterns.

▶ **Theorem 37.** *For a strongly almost non-ω-overlapping Constructor TRS, any Σ-closed consistent relation R on a term-coalgebra A is a consistency invariant.*

Proof. By Lemma 36 parallel rule applications are with ω-unifiable left-hand sides. As the system is strongly almost non-ω-overlapping both are therefore instances of a common generalisation $l_3 \to r_3$, and we can apply Corollary 35. ◀

The standard equivalence relation $=_R$ associated with a Constructor TRS can be expressed as $\mathrm{CGI}(\xrightarrow{\epsilon})$. We want to show that this relation is consistent. Instead, we are going to prove the stronger property that it is constructor-compatible.

To define the right kind of invariant we need another auxiliary function on binary relations over a term-coalgebra which allows us to compose rewrite steps and reasoning on subterms "in a safe way" with another relation.

▶ **Definition 38.** The unary function Ind_A on binary relations over a term-coalgebra A is defined as follows: $\mathrm{Ind}_A(x) \doteq (\xrightarrow{\epsilon}_A \cup \overline{x}) \cdot x$.

▶ **Lemma 39.** $\mathrm{Ind}_A(R)$ *is constructor-compatible.*

Proof. Constructor-topped terms are not in the domain of either $\xrightarrow{\epsilon}_A$ or $\overline{R}$. Hence $\widehat{id_A} \cdot \mathrm{Ind}_A(R) \cdot \widehat{id_A}$ is the empty relation. ◀

Using Ind_A we can construct a suitable consistent relation:

▶ **Definition 40.** Given a TRS with signature Σ, and a term-coalgebra A, the relation $\Downarrow_A$ is a relation on A defined as follows:

$$\begin{aligned} \Downarrow_A &\doteq \mu x.\, f_A(x) \\ f_A(x) &\doteq x^{-1} \cup \mathrm{Ind}_A(x) \cup id_A \cup \widehat{x} \cup x \end{aligned}$$

We omit the index if $A = Ter^\infty(\Sigma + X, \emptyset)$.

▶ **Lemma 41.** *Let A and B be term-coalgebras with $A \subseteq B$. Then $\Downarrow_A \subseteq \Downarrow_B$.*

Proof. We have $id_A \subseteq id_B$ and $\overset{\epsilon}{\to}_A \subseteq \overset{\epsilon}{\to}_B$ simply because $A \subseteq B$. Hence $\mathrm{Ind}_A(x) \subseteq \mathrm{Ind}_B(x)$ and $f_A(x) \subseteq f_B(x)$. The result follows by monotonicity of fixpoint constructions (Proposition 1.2.18 in [1]). ◀

▶ **Proposition 42.** *$\Downarrow_A$ is Σ-closed.*

Proof. $\overline{\Downarrow_A} = \overline{\Downarrow_A} \cdot id_A \subseteq \overline{\Downarrow_A} \cdot \Downarrow_A \subseteq \mathrm{Ind}_A(\Downarrow_A) \subseteq \Downarrow_A$, and $\widehat{\Downarrow_A} \subseteq \Downarrow_A$ is immediate. ◀

In contrast to joinability, the direction of rewrite steps only matters here at root position. Below root we do not rewrite, we just look for the invariant again. This change makes the relation $\Downarrow$ strictly more expressive than $\downarrow$ (in non-confluent systems):

▶ **Example 43.** Recall that in Example 3 we had $A \to_R^* E$ and $A \to_R^* C(E)$, without a common reduct for the two terms. However, we do have $E \Downarrow C(E)$, even $E \Downarrow_B C(B)$ in a strictly finite term-coalgebra B: $B = \{A, E, C(A), C(E), D(A, C(A)), D(C(A), C(A))\}$. Because $A \to_R^* E$ and $C(A) \to_R^* E$ we also have $A \Downarrow_B E$ (and $C(A) \Downarrow_B E$), by symmetry $E \Downarrow_B A$ and so $C(E) \overline{\Downarrow_B} C(A)$. Because $\Downarrow_B$ is closed under prefixing with $\overline{\Downarrow_B}$ we get $C(E) \Downarrow_B E$.

▶ **Proposition 44.** *For any term-coalgebra A and w.r.t. any Constructor TRS, the relation $\Downarrow_A$ is constructor-compatible.*

Proof. First note that the function f_A preserves constructor-compatibility: We have that the indivual parts of f_A preserve constructor-compatibility (Lemmas 29 and 39), hence $\widehat{id_A} \cdot f_A(x) \cdot \widehat{id_A} \subseteq \widehat{x^{-1}} \cup \widehat{\mathrm{Ind}_A(x)} \cup \widehat{id_A} \cup \widehat{x} \subseteq \widehat{f_A(x)}$.

From Lemma 29 we get that constructor-compatibility is sup-continuous, hence $\mu x.f_A(x)$ is constructor-compatible by Proposition 7. ◀

▶ **Corollary 45.** *Given a strongly almost non-ω-overlapping Constructor TRS, $\Downarrow_A$ is a consistency invariant on any term-coalgebra A.*

Proof. This follows directly from Theorem 37 and Propositions 44 and 42. ◀

7 Proof Graphs

We introduce the new concept of *proof graphs*. The immediate purpose of these structures is to permit us to reason about consistency proofs, and manipulate them, if necessary. The overall goal is to show that $\Downarrow_A$ is an equivalence.

We assume throughout a fixed Constructor TRS $(\Sigma, \mathcal{R})$, and a fixed strongly finite term-coalgebra A.

▶ **Definition 46.** A *proof graph* $\varrho = (\overset{\varrho}{\to}, =_\varrho)$ is given by a binary relation $\overset{\varrho}{\to}$ on A with the following properties:
1. $(\overset{\varrho}{\to} \cup \overset{\varrho}{\leftarrow})^* = =_\varrho \subseteq \Downarrow_A$;
2. $\overset{\varrho}{\to}$ is deterministic, i.e. $\overset{\varrho}{\leftarrow} \cdot \overset{\varrho}{\to} \subseteq id_A$;
3. $\overset{\varrho}{\to}$ is terminating;
4. $\overset{\varrho}{\to} \subseteq \overset{\epsilon}{\to}_A \cup \overline{\Downarrow_A} \cup \widehat{=_\varrho}$

Explanation: the first condition means that a proof graph represents an equivalence relation which is a subrelation of $\Downarrow_A$; the second and third condition means that this representation is a forest of trees (a union/find structure); the fourth condition means that these edges have "good properties" when we want to extend the proof graph and merge equivalence classes.

▶ **Lemma 47.** *Let $\varrho = (\overset{\varrho}{\to}, =_\varrho)$ be a proof graph. Then $=_\varrho$ is constructor-compatible.*

Proof. Let $t =_\varrho u$, where $t \; \widehat{id_A} \; t$ and $u \; \widehat{id_A} \; u$. The first condition of the definition gives us $t \; (\overset{\varrho}{\to} \cup \overset{\varrho}{\leftarrow})^* \; u$; by the second and third condition $\overset{\varrho}{\to}$ there must a common reduct $s \in A$ with $t \overset{\varrho}{\to}{}^* s$ and $s \overset{\varrho}{\leftarrow}{}^* u$. Because the only outgoing edges for constructor-topped nodes are of the relation $\widehat{=}_\varrho$ we have $t \; \widehat{=}_\varrho{}^* \; u$, but $\widehat{=}_\varrho{}^* \subseteq \widehat{(=_\varrho)^*} = \widehat{=}_\varrho$, and so $t \; \widehat{=}_\varrho \; u$. ◀

7.1 Extensions of a Proof Graph

▶ **Definition 48.** A node $t \in A$ is called a *normal form* of ϱ iff it is a normal form of the relation $\overset{\varrho}{\to}$. We write NF_ϱ for the set of normal forms of ϱ. We write $[\varrho](a)$ for the normal form of a node a.

The normal forms of a proof graph represent its equivalence classes. We want a way to merge equivalences classes of a proof graph. In its simplest form (without allowing for "rewiring") this means the following:

▶ **Definition 49.** Given two proof graphs α and β, β is an extension of α iff $\overset{\alpha}{\to} \subseteq \overset{\beta}{\to}$.

▶ **Lemma 50.** *Let β be an extension of α. Then for all $a, b \in A$ with $a \overset{\beta}{\to} b \wedge \neg(a \overset{\alpha}{\to} b)$ we must have: $\neg(a =_\alpha b)$ and $a \in \mathrm{NF}_\alpha$.*

Proof. By contradiction: If $a \notin \mathrm{NF}_\alpha$ then $a \overset{\alpha}{\to} b'$ for some b', hence $a \overset{\beta}{\to} b'$ because $\overset{\alpha}{\to} \subseteq \overset{\beta}{\to}$. But then $\overset{\beta}{\to}$ fails to be deterministic, so β could not be a proof graph. If $a =_\alpha b$ then $b \overset{\alpha}{\to}{}^* a$, because $\overset{\alpha}{\to}$ is deterministic and terminating. Therefore $b \overset{\beta}{\to}{}^* a \overset{\beta}{\to} b$. Thus $\overset{\beta}{\to}$ is not terminating, so β could not be a proof graph. ◀

In addition to that one needs that the new merged equivalence class in $=_\beta$ is still contained in $\Downarrow_A$. However, that turns out not to be an issue because of our restrictions on edges.

▶ **Definition 51.** Given a proof graph $\varrho = (\overset{\varrho}{\to}, =_\varrho)$ the *grey edge relation* on nodes is defined as $\rightsquigarrow_\varrho = \overset{\epsilon}{\to}_A \cup \overline{\Downarrow_A} \cup \widehat{=}_\varrho$.

So grey edges include those that are in that proof graph and those that "might be".

▶ **Lemma 52.** *Let ϱ be a proof graph. Let $\to_\beta \subseteq \rightsquigarrow_\varrho$ such that $\to_\beta$ is deterministic and terminating, and let $=_\beta$ be the equivalence closure of $\to_\beta$. Then $=_\beta \subseteq \Downarrow_A$.*

Proof. If $t =_\beta u$ we must have an $s \in A$ with $t \to_\beta^* s$ and $u \to_\beta^* s$, because $\to_\beta$ is deterministic and terminating. We prove this by induction on the number of $\to_\beta$ steps. Moreover, we strengthen the claim by requiring that if $t \; \widehat{id_A} \; t$ and $u \; \widehat{id_A} \; u$ then $t \; \widehat{=}_\varrho \; u$.

If $t = u$ then by reflexivity of $\Downarrow_A$ we have $t \Downarrow_A u$. If in addition $t \; \widehat{id_A} \; t$ then $t \; \widehat{=}_\varrho \; t$ by reflexivity of $=_\varrho$.

If $t \; (\overset{\epsilon}{\to}_A \cup \overline{\Downarrow_A}) \; t' =_\beta u$ then $t' \Downarrow_A u$ by induction hypothesis and so $t \; \mathrm{Ind}_A(\Downarrow_A) \; u$ which implies $t \Downarrow_A u$.

If $t =_\beta t' (\overset{\epsilon}{\to}_A \cup \overline{\Downarrow_A})^{-1} u$ then $t \Downarrow_A t'$ by induction hypothesis, $t' \Downarrow_A t$ by symmetry of $\Downarrow_A$, $u \Downarrow_A t$ by the previous argument, and $t \Downarrow_A u$ by symmetry.

Otherwise, we must have $t \; \widehat{id_A} \; t$ and $u \; \widehat{id_A} \; u$ and $t \; \widehat{=}_\varrho \; t' =_\beta u$. Thus t' is constructor-topped and $t' \; \widehat{=}_\varrho \; u$ by induction hypothesis. This implies $t \; \widehat{=_\varrho \cdot =_\varrho} \; u$ and so $t \; \widehat{=}_\varrho \; u$ by transitivity of $=_\varrho$. Since $=_\varrho \subseteq \Downarrow_A$ we have $\widehat{=}_\varrho \subseteq \widehat{\Downarrow_A} \subseteq \Downarrow_A$ and so $t \Downarrow_A u$. ◀

▶ **Corollary 53.** *Let* $\varrho = (\overset{\varrho}{\to}, =_\varrho)$ *be a proof graph, let* $a \rightsquigarrow_\varrho b$ *where* $a \in \mathrm{NF}_\varrho$ *and* $\neg(a =_\varrho b)$*. Then* $\beta = (\overset{\varrho}{\to} \cup \{(a,b)\}, =_\beta)$ *is an extension of* ϱ*.*

Proof. The conditions on a and b mean that $\overset{\beta}{\to}$ remains deterministic and terminating. Therefore, and because of $\overset{\beta}{\to} \subseteq \rightsquigarrow_\varrho$ we can apply Lemma 52 and get $=_\beta \subseteq \Downarrow_A$. Finally, because $=_\varrho \subset =_\beta$ we also have $\widehat{=}_\varrho \subseteq \widehat{=}_\beta$ and so all constructor edges can be retained. ◀

▶ **Definition 54.** A proof graph is called *complete* if it has no extensions other than itself.

▶ **Proposition 55.** *Every proof graph has a complete extension.*

Proof. Trivial, as each proper extension merges equivalence classes, and A is finite. ◀

▶ **Lemma 56.** *For every complete proof graph* ϱ *the relation* $=_\varrho$ *is* Σ_c*-closed.*

Proof. By contradiction. Suppose ϱ were complete and we had $t \mathrel{\widehat{=}_\varrho} u$ and $\neg(t =_\varrho u)$ then we must also have $t \mathrel{\widehat{=}_\varrho} [\alpha](t)$. This implies $[\alpha](t) \mathrel{\widehat{=}_\varrho} u$. By Corollary 53 $\beta = (\overset{\varrho}{\to} \cup \{(a,b)\}, =_\beta)$ is an extension of ϱ, which contradicts completeness of ϱ. ◀

The corresponding property is generally not true for Σ-closure, because we might only be able to attach new edges from $\overline{\Downarrow_A}$ to nodes that are redexes. But if a proof graph does not contain “too many redexes” then such a conflict does not materialise.

We can characterise proof graphs for which this is possible by considering the relation $\overline{\Downarrow_A}^*$ – which is an equivalence since $\Downarrow_A$ (and therefore $\overline{\Downarrow_A}$) is symmetric.

▶ **Definition 57.** For any $t \in A$ we define $E_t \doteq \{u \in A \mid t \mathrel{\overline{\Downarrow_A}^*} u\}$. We also define $D_A \doteq \{E_t \mid t \in A\}$. $R_t \doteq \{u \in E_t \mid \exists v \in A.\, u \overset{\epsilon}{\to}_A v\}$.

Thus E_t is the equivalence class of t for the relation $\overline{\Downarrow_A}^*$, R_t the subset of redexes of E_t and D_A the collection of equivalence classes. Note that if t is constructor-topped then E_t is a singleton and R_t is empty. We use these notions to build a proof a graph in which redexes and Σ-closure are not in conflict.

▶ **Definition 58.** A *target* is a function $\mathrm{targ} : D_A \to A$ such that the following properties hold: (i) $\forall t \in A.\, \mathrm{targ}(E_t) \in E_t$ and (ii) $\forall t \in A.\, R_t \neq \emptyset \Rightarrow \mathrm{targ}(E_t) \in R_t$.

Thus a target singles out a member of each equivalence class, which moreover must be a redex if the class contains any redexes. The motivation is to build a proof graph whose subgraph on E_t is a tree with root $\mathrm{targ}(E_t)$.

▶ **Definition 59.** A *targeted proof graph* is a pair $(\varrho, \mathrm{targ}_\varrho)$ such that ϱ is a proof graph, targ_ϱ is a target, and we have:

$$\forall t \in A.\, t\, (\overline{\Downarrow_A} \cap \overset{\varrho}{\to})^*\, \mathrm{targ}(E_t) \vee t \in \mathrm{NF}_\varrho$$

$$\forall t \in A, u \in A.\, \mathrm{targ}(E_t) \overset{\varrho}{\to} u \Rightarrow u \notin E_t$$

Thus, in a targeted proof graph a subset of E_t is already connected with root $\mathrm{targ}(E_t)$ whilst all other nodes in E_t are not linked to anything.

▶ **Definition 60.** A *targeted extension* of a proof graph α is a targeted proof graph $(\beta, \mathrm{targ}_\beta)$ such that β is an extension of α.

▶ **Lemma 61.** *Any targeted proof graph has a targeted extension which is complete.*

Proof. Suppose ϱ was targeted and there is a $t \in \mathrm{NF}_\varrho$ such that with $t \neq \mathrm{targ}(E_t)$. Then there are nodes $t = t_0, \ldots, t_n = \mathrm{targ}(E_t)$ such that $\forall i.\, t_i \overline{\Downarrow_A} t_{i+1}$ and there must be some $j < n$ such that $t_j \in \mathrm{NF}_\varrho$ and $t_{j+1}\ (\overline{\Downarrow_A} \cap \overset{\varrho}{\to})^*\ \mathrm{targ}(E_t)$. Thus we can extend ϱ with the edge (t_j, t_{j+1}), keep the same target, and then complete the extension.

If there is no such node then any extension will remain targeted, and so we can apply Proposition 55. ◀

▶ **Lemma 62.** *If $(\alpha, \mathrm{targ}_\alpha)$ is a complete targeted proof graph then $=_\alpha$ is Σ-closed.*

Proof. By Lemma 56 we know that $=_\alpha$ is Σ_c-closed. Let $t \overline{=_\alpha} u$. Because α is complete and targeted $t \overset{\alpha}{\to}^* \mathrm{targ}_\alpha(E_t)$ and $u \overset{\alpha}{\to}^* \mathrm{targ}_\alpha(E_u)$ Since $t \overline{=_\alpha} u$ we have $t \overline{\Downarrow_A} u$, hence $E_t = E_u$. ◀

7.2 The Universal Proof Graph

The kind of proof graph we want to build is one whose equivalence is the full relation $\Downarrow_A$, because that would show that $\Downarrow_A$ is transitive.

▶ **Definition 63.** A proof graph ϱ is *universal* iff $=_\varrho = \Downarrow_A$.

▶ **Lemma 64.** *If $\Downarrow_A$ is a consistency invariant for our rewrite system then any targeted proof graph which is complete is universal.*

Proof. Let $(\varrho, \mathrm{targ}_\varrho)$ a target proof graph which is complete. This is universal iff $=_\varrho$ and $\mathrm{CG}(\overset{\epsilon}{\to}_A)$ coincide. We write $=_R$ for $\mathrm{CG}(\overset{\epsilon}{\to}_A)$.

We prove the implication $\forall t, u \in A.\, t \Downarrow_A u \Rightarrow t =_\varrho u$ by induction on the term structure. Because $=_\varrho$ is a Σ-closed equivalence this reduces to $\forall t, u \in A.\, t \overset{\epsilon}{\to}_A u \Rightarrow t =_\varrho u$.

If $t \overset{\epsilon}{\to}_A u$ then $R_t \neq \emptyset$ and so $\mathrm{targ}_\varrho(E_t) \in R_t$ and we have $t\ (\overline{\Downarrow_A} \cap \overset{\varrho}{\to})^*\ \mathrm{targ}_\varrho(E_t)$ by completeness of ϱ. Since $\overline{\Downarrow_A}^* \subseteq \overline{=_R}^* \subseteq \overline{=_R}$ we get $t \overline{=_R} \mathrm{targ}_\varrho(E_t)$ and so by induction hypothesis $t \overline{=_\varrho} \mathrm{targ}_\varrho(E_t)$. Because $\mathrm{targ}_\varrho(E_t) \in R_t$ there is some $r \in A$ with $\mathrm{targ}_\varrho(E_t) \overset{\epsilon}{\to}_A r$ and because of completeness we have $\mathrm{targ}_\varrho(E_t) =_\varrho r$. The consistency invariant property then gives us $u\ \mathrm{CT}(=_\varrho)\ r$. Because $=_\varrho$ is Σ-closed (Lemma 62) we get $u =_\varrho r$ and $t =_\varrho \mathrm{targ}_\varrho(E_t)$. Overall $t =_\varrho \mathrm{targ}\varrho(E_t) =_\varrho r =_\varrho u$. ◀

▶ **Lemma 65.** *If $\Downarrow_A$ is a consistency invariant for our rewrite system then there is a universal proof graph.*

Proof. Let $\mathrm{targ} : D_A \to A$ be any target function. Then $((\emptyset, \mathit{id}), \mathrm{targ})$ is a targeted proof graph. We can then apply Lemma 61 and Lemma 65. ◀

▶ **Theorem 66.** *Let (Σ, R) be a Constructor TRS such that $\Downarrow_A$ is a consistency invariant for any strongly finite term-coalgebra A. Then $=_R$ coincides with $\Downarrow$ on $Ter(\Sigma, \emptyset)$ (and is therefore constructor-compatible).*

Proof. Moreover, we even have that $t =_R u$ implies $t \Downarrow_B u$ for some strongly finite B.

Since $t =_R u$, we must have a sequence of distinct terms $s_1, \ldots, s_n$ with $t = s_1$ and $u = s_n$ such that for all $i < n$ either $s_i \to_R s_{i+1}$ or $s_{i+1} \to_R s_i$. Closing the set $\{s_1, \ldots, s_n\}$ under subterms then gives us the term-coalgebra B. By assumption $\Downarrow_B$ is a consistency invariant for B, therefore there is a universal proof graph for B by Lemma 65 and thus $t \Downarrow_B u$. By the coalgebra-inclusion argument (Lemma 41) we have $t \Downarrow u$. ◀

▶ **Corollary 67.** *Strongly almost non-ω-overlapping Constructor TRSs have a consistent equational theory.*

Proof. Follows from Theorem 66 and Corollary 45. ◀

8 Consequences

Now we can put these results together to deliver the main theorems.

▶ **Theorem 68.** *Non-ω-overlapping TRSs have a consistent equational theory.*

Proof. By Corollary 16 a TRS is consistent iff its constructor translation is. By Proposition 22 the constructor translation of a non-ω-overlapping TRS is strongly almost non-ω-overlapping, which – by Corollary 67 – means that it is consistent. ◀

From this, we also easily get uniqueness of normal forms:

▶ **Theorem 69.** *Non-ω-overlapping TRSs have unique normal forms.*

Proof. By contradiction. Suppose $T = (\Sigma, R)$ were a non-ω-overlapping TRS, $t, u \in Ter(\Sigma, X)$ were normal forms with $t =_R u$ and $t \neq u$. Then they remain normal forms in the TRS $U = (\Sigma + X + \{F\}, R \cup F(t,x,y) \to x, F(u,x,y) \to y)$. The system U is also non-ω-overlapping, as the new rules do not ω-overlap with each other or any old rule. But $x =_U F(t,x,y) =_U F(u,x,y) =_U y$, i.e. U is inconsistent which contradicts Theorem 68. ◀

9 Future Work

We would like to extend the result to wider ranges of TRSs. In particular, it would be nice to be able to extend it to almost non-ω-overlapping Constructor TRSs, as these are the kind of TRSs that are of concern in the Glasgow Haskell compiler which originated our interest [4]. This is almost certainly doable with just slight extensions of the techniques displayed here, though extending this further to arbitrary non-ω-overlapping TRSs might not be as straightforward.

Also of special interest are semi-equational Conditional TRSs as the can be used to turn TRSs into equivalent left-linear CTRSs, and non-duplicating TRSs into linear CTRSs, by transforming variable sharing into equational constraints.

10 Conclusion

We have proved that non-ω-overlapping TRS have a consistent equational theory, as well as unique normal forms. More important than the result itself is the novel proof technique that makes use of finite Σ-coalgebras, in order to show that certain relations are invariants across equational reasoning. The technique is related to the notion of "equivalent reductions" of orthogonal term rewriting, but in contrast does not require "the creation of" additional terms – the terms participating in the consistency proof are the same as the ones of the original equational proof.

References

1 André Arnold and Damian Niwiński. *Rudiments of μ-calculus*. North-Holland, 2001.

2 Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

3 Hendrik P. Barendregt. *The Lambda-Calculus, its Syntax and Semantics*. North-Holland, Amsterdam, 1984.

4 Richard Eisenberg, Dimitrios Vytiniotis, Simon Peyton Jones, and Stephanie Weirich. Closed type families with overlapping equations. In *Principles of Programming Languages*, pages 671–684. ACM, 2014.

5 Jörg Endrullis, Helle Hvid Hansen, Dimitri Hendriks, Andrew Polonsky, and Alexandra Silva. A coinductive treatment of infinitary rewriting. In *Workshop on Infinitary Rewriting*, page 8, 2013.

6 Jörg Endrullis, Dimitri Hendriks, Helle Hvid Hansen, Andrew Polonsky, and Alexandra Silva. A coinductive framework for infinitary rewriting and equational reasoning. In *Rewriting Techniques and Applications*, 2015.

7 Gérard Huet. *Résolution d'Equations dans les Langages d'Ordre 1,2,...,ω*. PhD thesis, Université de Paris VII, 1976.

8 Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27:797–821, 1980.

9 Richard Kennaway and Fer-Jan de Vries. *Term Rewriting Systems*, chapter Infinitary Rewriting, pages 668–711. Cambridge University Press, 2003.

10 Jan Willem Klop. *Combinatory Reduction Systems*. PhD thesis, Centrum voor Wiskunde en Informatica, 1980.

11 Mizuhito Ogawa and Satoshi Ono. On the uniquely converging property of nonlinear term rewriting systems. Technical Report IEICE COMP89-7, NTT Software Laboratories, 1989.

12 Gordon D. Plotkin. A note on inductive generalisation. *Machine Intelligence*, 5:153–163, 1970.

13 Barry K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *Journal of the ACM*, 20:160–187, 1973.

14 RTA. The RTA list of open problems. `http://www.cs.tau.ac.il/~nachum/rtaloop`

15 Donald Sannella and Andrzej Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Springer, 2012.

16 Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, April 1975.

17 Alfred Tarski. A lattice-theoretic fixed point theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

18 Terese, editor. *Term Rewriting Systems*. Cambridge University Press, 2003.

Complexity Hierarchies and Higher-Order Cons-Free Rewriting*

Cynthia Kop[1] and Jakob Grue Simonsen[2]

1 **Department of Computer Science, University of Copenhagen (DIKU), Copenhagen, Denmark**
`kop@di.ku.dk`

2 **Department of Computer Science, University of Copenhagen (DIKU), Copenhagen, Denmark**
`{simonsen@di.ku.dk`

Abstract

Constructor rewriting systems are said to be cons-free if, roughly, constructor terms in the right-hand sides of rules are subterms of constructor terms in the left-hand side; the computational intuition is that rules cannot build new data structures. It is well-known that cons-free programming languages can be used to characterize computational complexity classes, and that cons-free first-order term rewriting can be used to characterize the set of polynomial-time decidable sets.

We investigate cons-free higher-order term rewriting systems, the complexity classes they characterize, and how these depend on the order of the types used in the systems. We prove that, for every $k \geq 1$, left-linear cons-free systems with type order k characterize $\mathrm{E}^k\mathrm{TIME}$ if arbitrary evaluation is used (i.e., the system does not have a fixed reduction strategy).

The main difference with prior work in implicit complexity is that (i) our results hold for non-orthogonal term rewriting systems with possible rule overlaps with no assumptions about reduction strategy, (ii) results for such term rewriting systems have previously only been obtained for $k = 1$, and with additional syntactic restrictions on top of cons-freeness and left-linearity.

Our results are apparently among the first implicit characterizations of the hierarchy $\mathrm{E} = \mathrm{E}^1\mathrm{TIME} \subsetneq \mathrm{E}^2\mathrm{TIME} \subsetneq \cdots$. Our work confirms prior results that having full non-determinism (via overlaps of rules) does not directly allow characterization of non-deterministic complexity classes like NE. We also show that non-determinism makes the classes characterized highly sensitive to minor syntactic changes such as admitting product types or non-left-linear rules.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases higher-order term rewriting, implicit complexity, cons-freeness, ETIME hierarchy

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.23

1 Introduction

In [14], Jones introduces *cons-free programming*: working with a small functional programming language, cons-free programs are exactly those where function bodies cannot contain use of data constructors (the "cons" operator on lists). Put differently, a cons-free program is

[*] The authors are supported by the Marie Skłodowska-Curie action "HORIP", program H2020-MSCA-IF-2014, 658162 and by the Danish Council for Independent Research Sapere Aude grant "Complexity via Logic and Algebra" (COLA).

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Brigitte Pientka and Delia Kesner; Article No. 23; pp. 23:1–23:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

read-only: data structures cannot be created or altered, only read from the input; and any data passed as arguments to recursive function calls must thus be part of the original input.

The interest in such programs lies in their applicability to computational complexity: by imposing cons-freeness, the resulting set of programs can only decide the sets in a proper subclass of the Turing-decidable sets, indeed are said to *characterize* the subclass. Jones goes on to show that adding further restrictions such as type order or enforcing tail recursion lowers the resulting expressiveness to known classes. For example, cons-free programs with data order 0 can decide exactly the sets in PTIME, while tail-recursive cons-free programs with data order 1 can decide exactly the sets in PSPACE. The study of such restrictions and the complexity classes characterized is a research area known as *implicit complexity* and has a long history with many distinct approaches (see, e.g., [4, 6, 5, 7, 8, 12, 17]).

Rather than a toy language, it is tantalizing to consider *term rewriting* instead. Term rewriting systems have no fixed evaluation order (so call-by-name or call-by-value can be introduced as needed, but are not *required*); and term rewriting is natively non-deterministic, allowing distinct rules to be applied ("functions to be invoked") to the same piece of syntax, hence could be useful for extensions towards non-deterministic complexity classes. Implicit complexity using term rewriting has seen significant advances using a plethora of approaches (e.g. [1, 2, 3]). Most of this research has, however, considered fixed evaluation orders (most prominently innermost reduction), and if not, then systems which are either orthogonal, or at least confluent (e.g. [2]). Almost all of the work considers only first-order rewriting.

The authors of [11] provide a first definition of cons-free term rewriting without constraints on evaluation order or confluence requirements, and prove that this class – limited to *first-order* rewriting – characterizes PTIME. However, they impose a rather severe partial linearity restriction on the programs. This paper seeks to answer two questions: (i) what happens if *no* restrictions beyond left-linearity and cons-freeness are imposed? And (ii) what if *higher-order* term rewriting – including bound variables as in the lambda calculus – is allowed? We obtain that k^{th}-order cons-free term rewriting exactly characterizes E^kTIME. This is surprising because in Jones' rewriting-like language, k^{th}-order programs characterize $\text{EXP}^{k-1}\text{TIME}$: surrendering both determinism and evaluation order thus significantly increases expressivity.

An extended version, including appendices with full proofs, is available online [16].

2 Preliminaries

2.1 Computational complexity

We presuppose introductory working knowledge of computability and complexity theory (corresponding to standard textbooks, e.g., [13]). Notation is fixed below.

Turing Machines (TMs) are triples (A, S, T) where A is a finite set of tape symbols such that $A \supseteq I \cup \{\text{␣}\}$, where $I \supseteq \{0, 1\}$ is a set of *initial symbols* and $\text{␣} \notin I$ is the special *blank* symbol; $S \supseteq \{\texttt{start}, \texttt{accept}, \texttt{reject}\}$ is a finite set of states, and T is a finite set of transitions (i, r, w, d, j) with $i \in S \setminus \{\texttt{accept}, \texttt{reject}\}$ (the *original state*), $r \in A$ (the *read symbol*), $w \in A$ (the *written symbol*), $d \in \{\texttt{L}, \texttt{R}\}$ (the *direction*), and $j \in S$ (the *result state*). We sometimes write this transition as $i \xRightarrow{r/w\ d} j$. All TMs in the paper are deterministic and (which we can assume wlog.) do not get stuck: for every pair (i, r) with $i \in S \setminus \{\texttt{accept}, \texttt{reject}\}$ and $r \in A$ there is exactly one transition (i, r, w, d, j). Every TM has a single, right-infinite tape.

A *valid tape* is a right-infinite sequence of tape symbols with only finitely many not ␣. A *configuration* of a TM is a triple (t, p, s) with t a valid tape, $p \in \mathbb{N}$ and $s \in S$. The transitions T induce a binary relation $\Rightarrow$ between configurations in the obvious way.

A TM with input alphabet I *decides* $X \subseteq I^+$ if for any string $x \in I^+$, we have $x \in X$ iff $(\llcorner x_1 \dots x_n \llcorner\llcorner \dots, 0, \texttt{start}) \Rightarrow^* (t, i, \texttt{accept})$ for some t, i, and $(\llcorner x_1 \dots x_n \llcorner\llcorner \dots, 0, \texttt{start}) \Rightarrow^* (t, i, \texttt{reject})$ otherwise (i.e., the machine halts on all inputs, ending in `accept` or `reject` depending on whether $x \in X$). If $f : \mathbb{N} \longrightarrow \mathbb{N}$ is a function, a (deterministic) TM *runs in time* $\lambda n.f(n)$ if, for each $n \in \mathbb{N} \setminus \{0\}$ and each $x \in I^n$: $(\llcorner x \llcorner\llcorner \dots, 0, \texttt{start}) \Rightarrow^{\leq f(n)} (t, i, \underline{\texttt{s}})$ for $\underline{\texttt{s}} \in \{\texttt{accept}, \texttt{reject}\}$, where $\Rightarrow^{\leq f(n)}$ denotes a sequence of at most $f(n)$ transitions.

Complexity and the ETIME hierarchy

For $k, n \geq 0$, let $\exp_2^0(n) = n$ and $\exp_2^{k+1}(n) = 2^{\exp_2^k(n)} = \exp_2^k(2^n)$.

▶ **Definition 1.** Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ be a function. Then, $\mathrm{TIME}\,(f(n))$ is the set of all $S \subseteq \{0,1\}^+$ such that there exist $a > 0$ and a deterministic TM running in time $\lambda n.a \cdot f(n)$ that decides S (i.e., S is decidable in time $O(f(n))$). For $k \geq 1$ define: $\mathrm{E}^k\mathrm{TIME} \triangleq \bigcup_{a \in \mathbb{N}} \mathrm{TIME}\left(\exp_2^k(an)\right)$

Observe in particular that $\mathrm{E}^1\mathrm{TIME} = \bigcup_{a \in \mathbb{N}} \mathrm{TIME}\left(\exp_2^1(an)\right) = \bigcup_{a \in \mathbb{N}} \mathrm{TIME}\left(2^{an}\right) = \mathrm{E}$ (where E is the usual complexity class of this name, see e.g., [19, Ch. 20]).

Note that for any $d, k \geq 1$, we have $(\exp_2^k(x))^d = 2^{d \cdot \exp_2^{k-1}(x)} \leq 2^{\exp_2^{k-1}(dx)} = \exp_2^k(dx)$. Hence, if P is a polynomial with non-negative integer coefficients and the set $S \subseteq \{0,1\}^+$ is decided by an algorithm running in time $O(P(\exp_2^k(an)))$ for some $a \in \mathbb{N}$, then $S \in \mathrm{E}^k\mathrm{TIME}$.

Using the Time Hierarchy Theorem [20], it is easy to see that $\mathrm{E} = \mathrm{E}^1\mathrm{TIME} \subsetneq \mathrm{E}^2\mathrm{TIME} \subsetneq \mathrm{E}^3\mathrm{TIME} \subsetneq \cdots$. The union $\bigcup_{k \in \mathbb{N}} \mathrm{E}^k\mathrm{TIME}$ is the set ELEMENTARY of elementary languages.

2.2 Higher-order rewriting

Unlike first-order term rewriting, there is no single, unified approach to higher-order term rewriting, but rather a number of different co-extensive systems with distinct syntax; for an overview of basic issues, see [21]. We will use *Algebraic Functional Systems* (AFSs) [15, 9], in the simplest form (which disallows partial applications). However, our proofs do not use any particular features of AFSs that preclude using different higher-order formalisms.

Types and Terms

We assume a non-empty set $\mathcal{S}$ of *sorts*, and define types and type orders as follows: (i) every $\iota \in \mathcal{S}$ is a type of order *0*; (ii) if σ, τ are types of order n and m respectively, then $\sigma \Rightarrow \tau$ is a type of order $\max(n+1, m)$. Here $\Rightarrow$ is right-associative, so $\sigma \Rightarrow \tau \Rightarrow \pi$ should be read $\sigma \Rightarrow (\tau \Rightarrow \pi)$. A *type declaration* of order $k \geq 0$ is a tuple $[\sigma_1 \times \cdots \times \sigma_n] \Rightarrow \iota$ with all σ_i types of order at most $k-1$, and $\iota \in \mathcal{S}$; if $n = 0$ this declaration may simply be denoted ι.

We additionally assume given disjoint sets $\mathcal{F}$ of *function symbols* and $\mathcal{V}$ of *variables*. Each symbol in $\mathcal{F}$ is associated with a unique type declaration, and each variable in $\mathcal{V}$ with a unique type. The set $\mathcal{T}(\mathcal{F}, \mathcal{V})$ of *terms over* $\mathcal{F}$ *and* $\mathcal{V}$ consists of those expressions s such that $\vdash s : \sigma$ can be derived for some type σ using the following clauses:

`(var)`	$\vdash x : \sigma$	if $x : \sigma \in \mathcal{V}$
`(app)`	$\vdash s \cdot t : \tau$	if $s : \sigma \Rightarrow \tau$ and $t : \sigma$
`(abs)`	$\vdash \lambda x.s : \sigma \Rightarrow \tau$	if $x : \sigma \in \mathcal{V}$ and $s : \tau$
`(fun)`	$\vdash f(s_1, \dots, s_n) : \iota$	if $f : [\sigma_1 \times \dots \times \sigma_n] \Rightarrow \iota \in \mathcal{F}$ and $s_1 : \sigma_1, \dots, s_n : \sigma_n$

Clearly, each term has a *unique* type. Note that a function symbol $f : [\sigma_1 \times \dots \times \sigma_n] \Rightarrow \iota$ takes exactly n arguments, and its output type ι is a sort. The *abstraction* construction $\lambda x.s$ binds occurrences of x in s as in the λ-calculus, and α-conversion is defined for terms *mutatis mutandis*; we identify terms modulo α-conversion, renaming bound variables if necessary.

Application is left-associative. The set of variables of s which are not bound is denoted $FV(s)$. A term s is *closed* if $FV(s) = \emptyset$. We say that a term s *has base type* if $\vdash s : \iota \in \mathcal{S}$.

▶ **Example 2.** We will often use extensions of the signature $\mathcal{F}_{\mathtt{string}}$, given by:

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
\mathtt{true} & : & \mathtt{bool} & \mathtt{0} & : & [\mathtt{string}] \Rightarrow \mathtt{string} & \rhd & : & \mathtt{string} \\
\mathtt{false} & : & \mathtt{bool} & \mathtt{1} & : & [\mathtt{string}] \Rightarrow \mathtt{string} & & &
\end{array}$$

Terms are for instance $\mathtt{true}$, $\lambda x.\mathtt{0}(\mathtt{1}(x))$ and $(\lambda x.\mathtt{0}(x)) \cdot \mathtt{1}(y)$. The first and last of these terms have base type, and the first two are closed; the last one has y as a free variable.

A *substitution* is a type-preserving map from $\mathcal{V}$ to $\mathcal{T}(\mathcal{F}, \mathcal{V})$ which is the identity on all but finitely many variables. Substitutions γ are extended to arbitrary terms s, notation $s\gamma$, by using α-conversion to rename all bound variables in s to fresh ones, then replacing each unbound variable x by $\gamma(x)$. A *context* $C[]$ is a term in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ in which a single occurrence of a variable is replaced by a symbol $\Box \notin \mathcal{F} \cup \mathcal{V}$. The result of replacing $\Box$ in $C[]$ by a term s (of matching type) is denoted $C[s]$. Free variables may be captured; e.g. $(\lambda x.\Box)[x] = \lambda x.x$. If $s = C[t]$ we say that t is a *subterm* of s, notation $t \trianglelefteq s$, or $t \lhd s$ if $C[] \neq \Box$.

Rules and Rewriting

A *rule* is a pair $\ell \to r$ of terms in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ with the same *sort* (i.e. $\vdash \ell : \iota$ and $\vdash r : \iota$ for some $\iota \in \mathcal{S}$), such that ℓ has the form $f(\ell_1, \dots, \ell_n)$ with $f \in \mathcal{F}$ and such that $FV(r) \subseteq FV(\ell)$. A rule $\ell \to r$ is *left-linear* if every variable occurs at most once in ℓ. We assume given a set $\mathcal{R}$ of rules, and define the one-step rewrite relation $\to_{\mathcal{R}}$ on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ as follows:

$$\begin{array}{rcl@{\qquad}l}
C[\ell\gamma] & \to_{\mathcal{R}} & C[r\gamma] & \text{with } \ell \to r \in \mathcal{R},\ C \text{ a context, } \gamma \text{ a substitution} \\
C[(\lambda x.s) \cdot t] & \to_{\mathcal{R}} & C[s[x := t]] &
\end{array}$$

We may write $s \to_{\beta} t$ for a rewrite step using (beta). Let $\to_{\mathcal{R}}^{+}$ denote the transitive closure of $\to_{\mathcal{R}}$ and $\to_{\mathcal{R}}^{*}$ the transitive-reflexive closure. We say that s *reduces to* t if $s \to_{\mathcal{R}} t$. A term s is in *normal form* if there is no t such that $s \to_{\mathcal{R}} t$, and t *is a normal form of* s if $s \to_{\mathcal{R}}^{*} t$ and t is in normal form. An AFS is a pair $(\mathcal{F}, \mathcal{R})$, generating a set of terms and a reduction relation. The *order* of an AFS is the maximal order of any type declaration in $\mathcal{F}$.

▶ **Example 3.** Recall the signature $\mathcal{F}_{\mathtt{string}}$ from Example 2; let $\mathcal{F}_{\mathtt{count}}$ be its extension with $\mathtt{succ} : [\mathtt{string}] \Rightarrow \mathtt{string}$. We consider the AFS $(\mathcal{F}_{\mathtt{count}}, \mathcal{R}_{\mathtt{count}})$ with the following rules:

$$\begin{array}{lrcl@{\quad}lrcl}
(\mathsf{A}) & \mathtt{succ}(\rhd) & \to & \mathtt{1}(\rhd) & (\mathsf{B}) & \mathtt{succ}(\mathtt{0}(xs)) & \to & \mathtt{1}(xs) \\
 & & & & (\mathsf{C}) & \mathtt{succ}(\mathtt{1}(xs)) & \to & \mathtt{0}(\mathtt{succ}(xs))
\end{array}$$

This is a *first-order* AFS, implementing the successor function on a binary number expressed as a bitstring with the least significant digit first. For example, 5 is represented by $\mathtt{1}(\mathtt{0}(\mathtt{1}(\rhd)))$, and indeed $\mathtt{succ}(\mathtt{1}(\mathtt{0}(\mathtt{1}(\rhd)))) \to_{\mathcal{R}} \mathtt{0}(\mathtt{succ}(\mathtt{0}(\mathtt{1}(\rhd)))) \to_{\mathcal{R}} \mathtt{0}(\mathtt{1}(\mathtt{1}(\rhd)))$, which represents 6.

▶ **Example 4.** Alternatively, we may define a bit-sequence as a *function*: let $\mathcal{F}_{\mathtt{hocount}}$ be the extension of $\mathcal{F}_{\mathtt{string}}$ with $\mathtt{not} : [\mathtt{bool}] \Rightarrow \mathtt{bool}$, $\mathtt{ite} : [\mathtt{bool} \times \mathtt{bool} \times \mathtt{bool}] \Rightarrow \mathtt{bool}$ and $\mathtt{all}$, $\mathtt{succ} : [(\mathtt{bool} \Rightarrow \mathtt{bool}) \times \mathtt{string}] \Rightarrow \mathtt{string}$. Let $\mathcal{R}_{\mathtt{hocount}}$ consist of:

$$\begin{array}{lrcl}
(\mathsf{A}) & \mathtt{ite}(\mathtt{true}, x, y) & \to & x \quad (\mathsf{C}) \quad \mathtt{not}(x) \to \mathtt{ite}(x, \mathtt{false}, \mathtt{true}) \\
(\mathsf{B}) & \mathtt{ite}(\mathtt{false}, x, y) & \to & y \quad (\mathsf{D}) \quad \mathtt{all}(F, \rhd) \to F \cdot \rhd \\
(\mathsf{E}) & \mathtt{all}(F, \underline{\mathtt{a}}(xs)) & \to & \mathtt{ite}(F \cdot \underline{\mathtt{a}}(xs), \mathtt{all}(F, xs), \mathtt{false}) \qquad [\![\text{for } \underline{\mathtt{a}} \in \{\mathtt{0}, \mathtt{1}\}]\!] \\
(\mathsf{F}) & \mathtt{succ}(F, \rhd) & \to & \mathtt{not}(F \cdot \rhd) \\
(\mathsf{G}) & \mathtt{succ}(F, \underline{\mathtt{a}}(xs)) & \to & \mathtt{ite}(\mathtt{all}(F, xs), \mathtt{not}(F \cdot \underline{\mathtt{a}}(xs)), F \cdot \underline{\mathtt{a}}(xs)) \ [\![\text{for } \underline{\mathtt{a}} \in \{\mathtt{0}, \mathtt{1}\}]\!]
\end{array}$$

Note that (E) and (G) each represent *two* rules: one for each choice of $\underline{\mathtt{a}}$. This AFS is second-order, due to `all` and `succ`. A function F represents a (potentially infinite) binary number, with the i^{th} bit given by $F \cdot t$ for any bitstring t of length i (counting from $i = 0$, so $t = \rhd$). Thus, the number 0 is represented by, e.g., $\lambda x.\mathtt{false}$, and 1 by $\mathtt{ONE} ::= \lambda x.\mathtt{succ}(\lambda y.\mathtt{false}, x)$. Indeed $\mathtt{ONE} \cdot \rhd = (\lambda x.\mathtt{succ}(\lambda y.\mathtt{false}, x)) \cdot \rhd \to_\beta \mathtt{succ}(\lambda y.\mathtt{false}, \rhd) \to_{\mathcal{R}} \mathtt{not}((\lambda y.\mathtt{false}) \cdot \rhd) \to_\beta \mathtt{not}(\mathtt{false}) \to_{\mathcal{R}} \mathtt{true}$, and $\mathtt{ONE} \cdot \mathtt{0}^k(\rhd) \to_{\mathcal{R}}^* \mathtt{false}$ for $k > 0$.

We fix a partitioning of $\mathcal{F}$ into two disjoint sets, $\mathcal{D}$ of *defined symbols* and $\mathcal{C}$ of *constructor symbols*, such that $f \in \mathcal{D}$ for all $f(\vec{\ell}) \to r \in \mathcal{R}$. A term s is a *constructor term* if it is in $\mathcal{T}(\mathcal{C}, \mathcal{V})$ and a *proper constructor term* if it also contains no applications or abstractions. A *closed* proper constructor term is also called a *data term*. The set of data terms is denoted $\mathcal{DA}$. Note that data terms are built using only clause (fun). A term $f(s_1, \dots, s_n)$ with $f \in \mathcal{D}$ and each $s_i \in \mathcal{DA}$ is called a *basic term*. A *constructor rewriting system* is an AFS where each rule $f(\ell_1, \dots, \ell_n) \to r \in \mathcal{R}$ satisfies that all ℓ_i are proper constructor terms (and $f \in \mathcal{D}$). An AFS is a *left-linear constructor rewriting system* if moreover each rule is left-linear.

In a constructor rewriting system, β-reduction steps can always be done prior to other steps: if s has a normal form q and $s \to_\beta t$, then also $t \to_{\mathcal{R}}^* q$. Therefore we can (and will!) safely assume that the right-hand sides of rules are in normal form with respect to $\to_\beta$.

▶ **Example 5.** The AFSs from Examples 3 and 4 are left-linear constructor rewriting systems. In Example 3, $\mathcal{C} = \mathcal{F}_{\mathtt{string}}$ and $\mathcal{D} = \{\mathtt{succ}\}$. If a rule $\mathtt{0}(\rhd) \to \rhd$ were added to $\mathcal{R}_{\mathtt{count}}$, it would no longer be a constructor system, as this would force `0` to be in $\mathcal{D}$, conflicting with rule (B). A rule such as $\mathtt{equal}(xs, xs) \to \mathtt{true}$ would break left-linearity.

▶ Remark. Constructor rewriting systems – typically left-linear – are very common both in the literature on term rewriting and in functional programming, where similar restrictions are imposed. Left-linear systems are well-behaved: contraction of non-overlapping redexes cannot destroy redexes that they themselves are arguments of. Constructor systems avoid non-root overlaps, and allow for a clear split between data and intermediate terms.

They are, however, less common in the literature on *higher-order* term rewriting, and the notion of a *proper* constructor term is new for AFSs (although the exclusion of abstractions and applications in the left-hand sides roughly corresponds to *fully extended pattern HRSs* in Nipkow's style of higher-order rewriting [18]).

Deciding problems using rewriting

Like Turing Machines, an AFS can decide a set $X \subseteq I^+$ (where I is a finite set of symbols). Consider AFSs with a signature $\mathcal{F} = \mathcal{C} \cup \mathcal{D}$ where $\mathcal{C}$ contains symbols $\rhd : \mathtt{string}$, $\mathtt{true} : \mathtt{bool}$, $\mathtt{false} : \mathtt{bool}$ and $\mathtt{a} : [\mathtt{string}] \Rightarrow \mathtt{string}$ for all $a \in I$. There is an obvious correspondence between elements of I^+ and data terms of sort `string`; if $x \in I^+$, we write $\overline{\mathtt{x}}$ for the corresponding data term. The AFS *accepts* $D \subseteq I^+$ if there is a designated defined symbol $\mathtt{decide} : [\mathtt{string}] \Rightarrow \mathtt{bool}$ such that, for every $x \in I^+$ we have $\mathtt{decide}(\overline{\mathtt{x}}) \to_{\mathcal{R}}^* \mathtt{true}$ iff $x \in D$. More generally, we are interested in the reductions of a given *basic term* to a *data term*.

We use the acceptance criterion above – reminiscent of the acceptance criterion of non-deterministic Turing machines – because term rewriting is inherently non-deterministic unless further constraints (e.g., orthogonality) are imposed. Thus, an input x is "rejected" by a rewriting system iff there is no reduction to `true` from $\mathtt{decide}(\overline{\mathtt{x}})$; and as evaluation is non-deterministic, there may be many distinct reductions starting from $\mathtt{decide}(\overline{\mathtt{x}})$.

3 Cons-free rewriting

Since the purpose of this research is to find groups of programs which can handle *restricted* classes of Turing-computable problems, we will impose certain limitations. In particular, we will limit interest to *cons-free left-linear constructor rewriting systems.*

▶ **Definition 6.** A rule $\ell \to r$, presented using α-conversion in a form where all binders are distinct from $FV(\ell)$, is *cons-free* if for all subterms $s = f(s_1, \dots, s_n) \trianglelefteq r$ with $f \in \mathcal{C}$, we have $s \triangleleft \ell$ or $s \in \mathcal{DA}$. A left-linear constructor AFS $(\mathcal{F}, \mathcal{R})$ is cons-free if all rules in $\mathcal{R}$ are.

This definition corresponds largely to the definitions of cons-freeness appearing in [11, 14]. In a cons-free AFS, it is not possible to create more data, as we will see in Section 3.1.

▶ **Example 7.** The AFS from Example 3 is not cons-free due to rules (B) and (C). The AFS from Example 4 *is* cons-free (in rules (E) and (G), $\underline{\mathtt{a}}(xs)$ is allowed to occur on the right despite the constructor $\underline{\mathtt{a}}$, because it also occurs on the left). However, there are few interesting basic terms, as we do not consider for instance $\mathtt{succ}(\lambda x.\mathtt{false}, \triangleright)$ basic.

▶ Remark. The limitation to left-linear constructor AFSs is standard, but also *necessary*: if either restriction is dropped, our limitation to cons-free AFSs becomes meaningless. In the case of constructor systems, this is obvious: if defined symbols are allowed to occur within a left-hand side, then we could simply let $\mathcal{D} := \mathcal{F}$ and have a "cons-free" system. The case of left-linearity is a bit more sophisticated; this we will study in more detail in Section 6.

As the first two restrictions are necessary to give meaning to the third, we will consider the limitation to left-linear constructor AFSs implicit in the notion "cons-free".

3.1 Properties of Cons-free Term Rewriting

As mentioned, cons-free term rewriting cannot create new data. This means that the set of data terms that might occur during a reduction starting in some basic term s are exactly the data terms occurring in s, or those occurring in the right-hand side of some rule. Formally:

▶ **Definition 8.** Let $(\mathcal{F}, \mathcal{R})$ be a constructor AFS. For a given term s, the set $\mathcal{B}_s$ contains all data terms t such that (i) $s \trianglerighteq t$, or (ii) $r \trianglerighteq t$ for some rule $\ell \to r \in \mathcal{R}$.

$\mathcal{B}_s$ is a set of data terms, is closed under subterms and, since we have assumed $\mathcal{R}$ to be fixed, has a linear number of elements in the size of s. The property that no new data is generated by reducing s is formally expressed by the following result:

▶ **Definition 9** ($\mathcal{B}$-safety)**.** Let $\mathcal{B} \subseteq \mathcal{DA}$ be a set which (i) is closed under taking subterms, and (ii) contains all data terms occurring as a subterm of the right-hand side of a rule in $\mathcal{R}$. A term s is *$\mathcal{B}$-safe* if for all t with $s \trianglerighteq t$: if t has the form $c(t_1, \dots, t_m)$ with $c \in \mathcal{C}$, then $t \in \mathcal{B}$.

▶ **Lemma 10.** *If s is $\mathcal{B}$-safe and $s \to_{\mathcal{R}} t$, then t is $\mathcal{B}$-safe.*

Proof Sketch. By induction on the form of s; the result follows trivially by the induction hypothesis if the reduction does not take place at the root, leaving only the base cases $s = (\lambda x.u) \cdot v \to_{\mathcal{R}} u[x := v] = t$ and $s = \ell\gamma \to_{\mathcal{R}} r\gamma = t$. The first of these is easy by induction on the form of the ($\mathcal{B}$-safe!) term u, the second follows by induction on the form of r (which, as the right-hand side of a cons-free rule, has convenient properties). ◀

Thus, if we start with a basic term $f(\vec{s})$, any data terms occurring in a reduction $f(\vec{s}) \to_{\mathcal{R}}^* t$ (directly or as subterms) are in $\mathcal{B}_{f(\vec{s})}$. This insight will be instrumental in Section 5.

▶ **Example 11.** By Lemma 10, functions in a cons-free AFSs cannot build recursive data. To code around this, we might use subterms of the input as a measure of length. Consider the decision problem whether a given bitstring is a palindrome. We cannot use a rule such as $\mathtt{decide}(cs) \to \mathtt{equal}(cs, \mathtt{reverse}(cs))$ since, by Lemma 10, it is impossible to define $\mathtt{reverse}$. Instead, a typical solution uses a string ys of length k to find $\overline{\mathtt{c_k}}$ in $\overline{\mathtt{c_0} \dots \mathtt{c_{n-1}}}$:

$$\begin{array}{rcll}
\mathtt{decide}(cs) & \to & \mathtt{palindrome}(cs, cs) & \\
\mathtt{palindrome}(cs, \rhd) & \to & \mathtt{true} & \\
\mathtt{palindrome}(cs, \underline{\mathtt{a}}(ys)) & \to & \mathtt{and}(\mathtt{palindrome}(cs, ys), \mathtt{chk}_{\underline{\mathtt{a}}}(cs, ys)) & [\![\underline{\mathtt{a}} \in \{\mathtt{0}, \mathtt{1}\}]\!]
\end{array}$$

$$\begin{array}{rcl@{\qquad}rcll}
\mathtt{and}(\mathtt{true}, x) & \to & x & \mathtt{chk}_{\underline{\mathtt{a}}}(\underline{\mathtt{a}}(xs), \rhd) & \to & \mathtt{true} & [\![\underline{\mathtt{a}} \in \{\mathtt{0}, \mathtt{1}\}]\!] \\
\mathtt{and}(\mathtt{false}, x) & \to & \mathtt{false} & \mathtt{chk}_{\underline{\mathtt{a}}}(\underline{\mathtt{b}}(xs), \rhd) & \to & \mathtt{false} & [\![\underline{\mathtt{a}}, \underline{\mathtt{b}} \in \{\mathtt{0}, \mathtt{1}\} \wedge \underline{\mathtt{a}} \neq \underline{\mathtt{b}}]\!] \\
 & & & \mathtt{chk}_{\underline{\mathtt{a}}}(\underline{\mathtt{b}}(xs), \underline{\mathtt{c}}(ys)) & \to & \mathtt{chk}_{\underline{\mathtt{a}}}(xs, ys) & [\![\underline{\mathtt{a}}, \underline{\mathtt{b}}, \underline{\mathtt{c}} \in \{\mathtt{0}, \mathtt{1}\}]\!]
\end{array}$$

(The signature extends $\mathcal{F}_{\mathtt{string}}$, but is otherwise omitted as types can easily be derived.)

Through cons-freeness, we obtain another useful property: we do not have to consider constructors which take functional arguments.

▶ **Lemma 12.** *Given a cons-free AFS* $(\mathcal{F}, \mathcal{R})$ *with* $\mathcal{F} = \mathcal{D} \cup \mathcal{C}$*, let* $Y = \{c : [\sigma_1 \times \dots \times \sigma_n] \Rightarrow \iota \in \mathcal{C}$ *some* σ_i *is not a sort*$\}$*. Define* $\mathcal{F}' := \mathcal{F} \setminus Y$*, and let* $\mathcal{R}'$ *consist of those rules in* $\mathcal{R}$ *not using any element of* Y *in either left- or right-hand side. Then (a) all data and* $\mathcal{B}$*-safe terms are in* $\mathcal{T}(\mathcal{F}', \emptyset)$*, and (b) if* s *is a basic term and* $s \to_{\mathcal{R}}^* t$*, then* $t \in \mathcal{T}(\mathcal{F}', \mathcal{V})$ *and* $s \to_{\mathcal{R}'}^* t$*.*

Proof. Since data terms have base type, and the subterms of data terms are data terms, we have (a). Then, $\mathcal{B}$-safe terms can only be matched by rules in $\mathcal{R}'$, so Lemma 10 gives (b). ◀

Therefore we may safely assume that all elements of $\mathcal{C}$ are at most first-order.

3.2 A larger example

None of our examples so far have taken advantage of the native non-determinism of term rewriting. To demonstrate the possibilities, we consider a first-order cons-free AFS that solves the Boolean satisfiability problem (SAT). This is striking because, in Jones' language in [14], first-order programs cannot solve this problem unless P = NP, even if a non-deterministic $\mathtt{choose}$ operator is added [10]. The crucial difference is that we, unlike Jones, do not employ a call-by-value evaluation strategy.

Given n boolean variables $x_1, \dots, x_n$ and a boolean formula $\psi ::= \varphi_1 \wedge \dots \wedge \varphi_n$, the satisfiability problem considers whether there is an assignment of each x_i to $\top$ or $\bot$ such that ψ evaluates to $\top$. Here, each clause φ_i has the form $a_{i_1} \vee \dots \vee a_{i_{k_i}}$, where each literal a_{i_j} is either some x_p or $\neg x_p$. We represent this problem as a string over $I := \{0, 1, \#, ?\}$: the formula ψ is represented by $L ::= b_{1,1} \dots b_{1,n} \# b_{2,1} \dots \# b_{m,1} \dots b_{m,n} \#$, where each $b_{i,j}$ is 1 if x_j is a literal in φ_i, is 0 if $\neg x_j$ is a literal in φ_i, and is ? otherwise.

▶ **Example 13.** The satisfiability problem for $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3)$ is encoded as $\mathtt{10?\#?10\#}$.

Letting $\mathtt{0}, \mathtt{1}, \#, ? : [\mathtt{string}] \Rightarrow \mathtt{string}$, and assuming other declarations clear from context, we claim that the AFS in Figure 1 can reduce $\mathtt{decide}(\overline{\mathtt{L}})$ to $\mathtt{true}$ iff ψ is satisfiable.

In this AFS, we follow some of the same ideas as in Example 11. In particular, any string of the form $b_i \dots b_n \# \dots$ with each $b_j \in \{0, 1, ?\}$ is considered to represent the number i. The rules for $\mathtt{eq}$ are defined so that $\mathtt{eq}(s, t)$ tests equality of these *numbers*, not the full strings.

The key idea new to this example is that we use terms not in normal form to represent a *set* of numbers. If we are interested in numbers in $\{1, \dots, n\}$, then a set $X \subseteq \{1, \dots, n\}$ is

$$\left.\begin{array}{rcl@{\qquad}rcl}
\mathtt{eq}(\#(xs),\#(ys)) & \to & \mathtt{true} & \mathtt{eq}(\#(xs),\underline{\mathtt{a}}(ys)) & \to & \mathtt{false} \\
\mathtt{eq}(\underline{\mathtt{a}}(xs),\underline{\mathtt{b}}(ys)) & \to & \mathtt{eq}(xs,ys) & \mathtt{eq}(\underline{\mathtt{a}}(xs),\#(ys)) & \to & \mathtt{false}
\end{array}\right\} [\![\text{for } \underline{\mathtt{a}},\underline{\mathtt{b}} \in \{\mathtt{0},\mathtt{1},?\}]\!]$$

$$\begin{array}{rcl}
\mathtt{decide}(cs) & \to & \mathtt{assign}(cs,\rhd,\rhd,cs) \\
\mathtt{assign}(\#(xs),s,t,cs) & \to & \mathtt{main}(s,t,cs)
\end{array}$$

$$\left.\begin{array}{rcl}
\mathtt{assign}(\underline{\mathtt{a}}(xs),s,t,cs) & \to & \mathtt{assign}(xs,\mathtt{either}(\underline{\mathtt{a}}(xs),s),t,cs) \\
\mathtt{assign}(\underline{\mathtt{a}}(xs),s,t,cs) & \to & \mathtt{assign}(xs,s,\mathtt{either}(\underline{\mathtt{a}}(xs),t),cs)
\end{array}\right\} [\![\text{for } \underline{\mathtt{a}} \in \{\mathtt{0},\mathtt{1},?\}]\!]$$

$$\mathtt{either}(xs,q) \;\to\; xs \qquad \mathtt{either}(xs,q) \;\to\; q$$

$$\begin{array}{rcl}
\mathtt{main}(s,t,?(xs)) & \to & \mathtt{main}(s,t,xs) \\
\mathtt{main}(s,t,\mathtt{0}(xs)) & \to & \mathtt{test}(s,t,xs,\mathtt{eq}(t,\mathtt{0}(xs)),\mathtt{eq}(s,\mathtt{0}(xs))) \\
\mathtt{main}(s,t,\mathtt{1}(xs)) & \to & \mathtt{test}(s,t,xs,\mathtt{eq}(s,\mathtt{1}(xs)),\mathtt{eq}(t,\mathtt{1}(xs)))
\end{array}$$

$$\begin{array}{rcl@{\qquad}rcl}
\mathtt{main}(s,t,\rhd) & \to & \mathtt{true} & \mathtt{test}(s,t,xs,\mathtt{true},z) & \to & \mathtt{main}(s,t,\mathtt{skip}(xs)) \\
\mathtt{main}(s,t,\#(xs)) & \to & \mathtt{false} & \mathtt{test}(s,t,xs,z,\mathtt{true}) & \to & \mathtt{main}(s,t,xs)
\end{array}$$

$$\begin{array}{rcl}
\mathtt{skip}(\#(xs)) & \to & xs \\
\mathtt{skip}(\underline{\mathtt{a}}(xs)) & \to & \mathtt{skip}(xs)\ [\![\text{for } \underline{\mathtt{a}} \in \{\mathtt{0},\mathtt{1},?\}]\!]
\end{array}$$

Figure 1 A cons-free first-order AFS solving the satisfiability problem.

encoded as a pair (s,t) of terms such that, for $i \in \{1,\dots,n\}$: $s \to_{\mathcal{R}}^* q$ for some representation q of i if and only if $i \in X$, and $t \to_{\mathcal{R}}^* q$ for some representation q of i if and only if $i \notin X$.

This is possible because we do not use a call-by-value or similar reduction strategy: an evaluation of this AFS is allowed to postpone reducing such terms, and we focus on those reductions. The AFS is constructed in such a way that reductions which evaluate these "sets" too eagerly simply end in an irreducible, non-data state.

Now, an evaluation starting in $\mathtt{decide}(\overline{\mathtt{L}})$ first non-deterministically constructs a "set" X containing those boolean variables assigned $\mathtt{true}$: $\mathtt{decide}(\overline{\mathtt{L}}) \to_{\mathcal{R}}^* \mathtt{main}(s,t,\overline{\mathtt{L}})$. Then, the main function goes through $\overline{\mathtt{L}}$, finding for each clause a literal that is satisfied by the assignment. Encountering for instance $b_{i_j} = \mathtt{1}$, we determine if $j \in X$ by comparing both a reduct of s and of t to j. If $s \to_{\mathcal{R}}^*$ "j" then $j \in X$, if $t \to_{\mathcal{R}}^*$ "j" then $j \notin X$; in either case, we continue accordingly. If the evaluation state is incorrect, or if s or t both non-deterministically reduce to some other term, the evaluation gets stuck in a non-data normal form.

▶ **Example 14.** To solve satisfiability of $(x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3)$, we reduce $\mathtt{decide}(\overline{\mathtt{L}})$, where $L = \mathtt{10?\#?10\#}$. First, we build a valuation; the choices made by the $\mathtt{assign}$ rules are non-deterministic, but a possible reduction is $\mathtt{decide}(\overline{\mathtt{L}}) \to_{\mathcal{R}}^* \mathtt{main}(s,t,\overline{\mathtt{L}})$, where $s = \mathtt{either}(\overline{\mathtt{10?\#?10\#}},\rhd)$ and $t = \mathtt{either}(\overline{\mathtt{?\#?10\#}},\mathtt{either}(\overline{\mathtt{0?\#?10\#}},\rhd))$. Recall that, since $n = 3$, $\overline{\mathtt{10?\#?10\#}}$ represents 1 while $\overline{\mathtt{?\#?10\#}}$ and $\overline{\mathtt{0?\#?10\#}}$ represent 3 and 2 respectively. Thus, this corresponds to the valuation $[x_1 := \top, x_2 := \bot, x_3 := \bot]$.

Then, the main loop recurses over the problem. Note that s reduces to a term $\overline{\mathtt{10?\#}\dots}$ and t reduces to both $\overline{\mathtt{?\#}\dots}$ and $\overline{\mathtt{0?\#}\dots}$. Therefore, $\mathtt{main}(s,t,\overline{\mathtt{L}}) = \mathtt{main}(s,t,\overline{\mathtt{11?\#?01\#}}) \to_{\mathcal{R}}^* \mathtt{main}(s,t,\mathtt{skip}(\overline{\mathtt{1?\#?01\#}})) \to_{\mathcal{R}}^* \mathtt{main}(s,t,\overline{\mathtt{?01\#}})$: the first clause is confirmed since x_1 is mapped to $\top$, so the clause is removed and the loop continues with the second clause. Next, the loop passes over those variables whose assignment does not contribute to verifying this clause, until the clause is confirmed by x_3: $\mathtt{main}(s,t,\overline{\mathtt{?01\#}}) \to_{\mathcal{R}} \mathtt{main}(s,t,\overline{\mathtt{01\#}}) \to_{\mathcal{R}}^* \mathtt{main}(s,t,\overline{\mathtt{1\#}}) \to_{\mathcal{R}}^* \mathtt{main}(s,t,\mathtt{skip}(\overline{\#})) \to_{\mathcal{R}} \mathtt{main}(s,t,\rhd) \to_{\mathcal{R}} \mathtt{true}$.

Using non-determinism, the term in Example 14 could easily have been reduced to $\mathtt{false}$ instead, simply by selecting a different valuation. This is not problematic: by definition, the AFS accepts the set of satisfiable formulas if $\mathtt{decide}(\overline{\mathtt{L}}) \to_{\mathcal{R}}^{*} \mathtt{true}$ if and only if L is a satisfiable formula: false negatives or reductions which do not end in a data state are allowed.

A longer example derivation is given in Appendix B of the full version of the paper.

4 Simulating $\mathrm{E}^k\mathrm{TIME}$ Turing machines

In order to see that cons-free term rewriting captures certain classes of decidable sets, we will simulate Turing Machines. For this, we use an approach very similar to that by Jones [14]. We introduce constructor symbols $\mathtt{a} : [\mathtt{string}] \Rightarrow \mathtt{string}$ for all $a \in A$ (including the blank symbol, which we shall refer to as $\mathtt{B}$) along with $\triangleright$ and the booleans, $\mathtt{s} : \mathtt{state}$ for all $s \in S \cup \{\text{fail}\}$, $\mathtt{L}, \mathtt{R} : \mathtt{direction}$ and $\mathtt{action} : [\mathtt{string} \times \mathtt{direction} \times \mathtt{state}] \Rightarrow \mathtt{trans}$, $\mathtt{end} : [\mathtt{state}] \Rightarrow \mathtt{trans}$, $\mathtt{NA} : \mathtt{trans}$. We will introduce defined symbols and rules such that, for any string $c \in (A \setminus \{\llcorner\!\lrcorner\})^*$ – represented as the term $cs := c_1(c_2(\cdots c_n(\triangleright)\cdots))$ – we have:

- $\mathtt{decide}(cs) \to_{\mathcal{R}}^{*} \mathtt{true}$ if and only if $(\llcorner\!\lrcorner c \llcorner\!\lrcorner\llcorner\!\lrcorner \dots, 0, \mathtt{start}) \Rightarrow^* (t, i, \mathtt{accept})$ for some t, i;
- $\mathtt{decide}(cs) \to_{\mathcal{R}}^{*} \mathtt{false}$ if and only if $(\llcorner\!\lrcorner c \llcorner\!\lrcorner\llcorner\!\lrcorner \dots, 0, \mathtt{start}) \Rightarrow^* (t, i, \mathtt{reject})$ for some t, i.

As rules may be overlapping, it is possible that $\mathtt{decide}(cs)$ will have additional normal forms, but only one normal form will be a data term.

The rough idea of the simulation is to represent non-negative integers as terms and let $\mathtt{tape}(n, p)$ reduce to the symbol at position p on the tape at the start of the n^{th} step, while $\mathtt{state}(n, p)$ returns the state of the machine at time n, provided the tape head is at position p. If the tape head of the machine is not at position p at time n, then $\mathtt{state}(n, p)$ should return $\mathtt{fail}$ instead; this makes it possible to test the position of the tape head at any given time. As the machine is deterministic, we can devise rules to compute these terms from earlier configurations.

Finding a suitable representation of integers and corresponding manipulating functions is the most intricate part of this simulation, where we may need both higher-order functions and non-deterministic rules. Therefore, let us first assume that this can be done. Then, for a Turing machine which is given to run in time bounded above by $\lambda x.P(x)$, we define the AFS in Figure 2. Note that, by construction, any occurrence of cs can only be instantiated by the input string during evaluation.

Counting

The goal, then, is to find a representation of numbers and functionality to do four things:

- calculate $[P(|cs|)]$ or an overestimation (as the machine cannot move from its final state);
- test whether a "number" represents 0;
- given $[n]$, calculate $[n-1]$, *provided* $n > 0$ – so it suffices to determine $[\max(n-1, 0)]$;
- given $[n]$, calculate $[n+1]$, *provided* $n+1 \leq P(|cs|)$ as necessarily $\mathtt{transition}(cs, [n], [p]) \to_{\mathcal{R}} \mathtt{NA}$ when $n < p$ and n never increases – so it suffices to determine $[\min(n+1, P(|cs|))]$.

Moreover, these calculations all occur in the right-hand side of a rule containing the initial input list cs on the left, which they can therefore use (for instance to recompute $P(|cs|)$).

Rather than representing a number by a single term, we will use *tuples* of terms (which are not terms themselves, as a pairing constructor would conflict with cons-freeness). To illustrate this, suppose we represent each number n by a pair (n_1, n_2). Then the predecessor and successor function must also be split, e.g. $\mathtt{pred}^1(cs, n_1, n_2) \to_{\mathcal{R}}^{*} n_1'$ and $\mathtt{pred}^2(cs, n_1, n_2) \to_{\mathcal{R}}^{*} n_2'$ for (n_1', n_2') some tuple representing $n - 1$. Thus, for instance the first $\mathtt{test}$ rule becomes:

$$\mathtt{test}(\mathtt{fail}, cs, n_1, n_2, p_1, p_2) \to \mathtt{findanswer}(cs, n_1, n_2, \mathtt{pred}^1(cs, p_1, p_2), \mathtt{pred}^2(cs, p_1, p_2))$$

$$\left.\begin{array}{rcl} \mathtt{ifelse}_{\iota}(\mathtt{true},y,z) & \to & y \\ \mathtt{ifelse}_{\iota}(\mathtt{false},y,z) & \to & z \end{array}\right\} \;[\![\text{for } \iota \in \{\mathtt{string},\mathtt{state}\}]\!]$$

$$\begin{array}{rcl} \mathtt{get}(\rhd,[i],q) & \to & q \\ \mathtt{get}(\underline{\mathtt{a}}(xs),[i],q) & \to & \mathtt{ifelse}_{\mathtt{string}}([i=0],\underline{\mathtt{a}}(\rhd),\mathtt{get}(xs,[i-1],q)) \;\; [\![\text{for all } \underline{\mathtt{a}} \in I]\!] \\ \mathtt{inputtape}(cs,[p]) & \to & \mathtt{ifelse}_{\mathtt{string}}([p=0],\mathtt{B}(\rhd),\mathtt{get}(cs,[p-1],\mathtt{B}(\rhd))) \end{array}$$

$$\begin{array}{rcl} \mathtt{tape}(cs,[n],[p]) & \to & \mathtt{ifelse}_{\mathtt{string}}([n=0],\mathtt{inputtape}(cs,[p]),\mathtt{tapex}(cs,[n-1],[p])) \\ \mathtt{tapex}(cs,[n],[p]) & \to & \mathtt{tapey}(cs,[n],[p],\mathtt{transition}(cs,[n],[p])) \end{array}$$

$$\begin{array}{rclrcl} \mathtt{tapey}(cs,[n],[p],\mathtt{action}(q,d,s)) & \to & q & \mathtt{tapey}(cs,[n],[p],\mathtt{NA}) & \to & \mathtt{tape}(cs,[n],[p]) \\ & & & \mathtt{tapey}(cs,[n],[p],\mathtt{end}(s)) & \to & \mathtt{tape}(cs,[n],[p]) \end{array}$$

$$\begin{array}{rcl} \mathtt{state}(cs,[n],[p]) & \to & \mathtt{ifelse}_{\mathtt{state}}([n=0],\mathtt{state0}(cs,[p]),\mathtt{statex}(cs,[n-1],[p])) \\ \mathtt{state0}(cs,[p]) & \to & \mathtt{ifelse}_{\mathtt{state}}([p=0],\mathtt{start},\mathtt{fail}) \\ \mathtt{statex}(cs,[n],[p]) & \to & \mathtt{statey}(\mathtt{transition}(cs,[n],[p-1]),\mathtt{transition}(cs,[n],[p]), \\ & & \quad \mathtt{transition}(cs,[n],[p+1])) \end{array}$$

$$\begin{array}{rclrcl} \mathtt{statey}(\mathtt{action}(q,\mathtt{R},s),y,z) & \to & s & \mathtt{statey}(\mathtt{NA},\mathtt{action}(q,d,s),z) & \to & \mathtt{fail} \\ \mathtt{statey}(\mathtt{action}(q,\mathtt{L},s),y,z) & \to & \mathtt{fail} & \mathtt{statey}(\mathtt{NA},\mathtt{NA},\mathtt{action}(q,\mathtt{L},s)) & \to & s \\ \mathtt{statey}(\mathtt{end}(s),y,z) & \to & \mathtt{fail} & \mathtt{statey}(\mathtt{NA},\mathtt{NA},\mathtt{action}(q,\mathtt{R},s)) & \to & \mathtt{fail} \\ \mathtt{statey}(\mathtt{NA},\mathtt{end}(s),z) & \to & s & \mathtt{statey}(\mathtt{NA},\mathtt{NA},\mathtt{end}(s)) & \to & \mathtt{fail} \end{array}$$

$$\begin{array}{rcl} \mathtt{transition}(cs,[n],[p]) & \to & \mathtt{transitionhelp}(\mathtt{state}(cs,[n],[p]),\mathtt{tape}(cs,[n],[p])) \\ \mathtt{transitionhelp}(\mathtt{fail},q) & \to & \mathtt{NA} \\ \mathtt{transitionhelp}(\underline{\mathtt{s}},\underline{\mathtt{r}}(\rhd)) & \to & \mathtt{action}(\underline{\mathtt{w}}(\rhd),\underline{\mathtt{d}},\underline{\mathtt{t}}) \qquad [\![\text{for all } \underline{\mathtt{s}} \overset{\underline{\mathtt{r}}/\underline{\mathtt{w}}\ \underline{\mathtt{d}}}{\Longrightarrow} \underline{\mathtt{t}} \in T]\!] \\ \mathtt{transitionhelp}(\underline{\mathtt{s}},q) & \to & \mathtt{end}(\underline{\mathtt{s}}) \qquad [\![\text{for } \underline{\mathtt{s}} \in \{\mathtt{accept},\mathtt{reject}\}]\!] \end{array}$$

$$\begin{array}{rcl} \mathtt{decide}(cs) & \to & \mathtt{findanswer}(cs,[P(|cs|)],[P(|cs|)]) \\ \mathtt{findanswer}(cs,[n],[p]) & \to & \mathtt{test}(\mathtt{state}(cs,[n],[p]),cs,[n],[p]) \\ \mathtt{test}(\mathtt{fail},cs,[n],[p]) & \to & \mathtt{findanswer}(cs,[n],[p-1]) \\ \mathtt{test}(\mathtt{accept},cs,[n],[p]) & \to & \mathtt{true} \\ \mathtt{test}(\mathtt{reject},cs,[n],[p]) & \to & \mathtt{false} \end{array}$$

Figure 2 Simulating a deterministic Turing Machine running in $\lambda x.P(x)$ time.

Following Jones [14], we use the notion of a *counting module* which provides an AFS with a representation of a counting function and a means of computing. Counting modules can be composed, making it possible to count to greater numbers. Due to the laxity of term rewriting, our constructions are technically quite different from those of [14].

▶ **Definition 15** (Counting Module). Write $\mathcal{F} = \mathcal{C} \cup \mathcal{D}$ for the signature in Figure 2. For P a function from $\mathbb{N}$ to $\mathbb{N}$, a P-counting module of *order* k is a tuple $C_\pi ::= (\vec{\sigma}, \Sigma, R, A, \langle\cdot\rangle)$ s.t.:

- $\vec{\sigma}$ is a sequence of types $\sigma_1 \times \cdots \times \sigma_a$ where each σ_i has order at most $k-1$;
- Σ is a k^{th}-order signature disjoint from $\mathcal{F}$, with designated symbols $\mathtt{zero}_\pi : [\mathtt{string} \times \vec{\sigma}] \Rightarrow \mathtt{bool}$ and, for $1 \le i \le a$ with $\sigma_i = \tau_1 \Rightarrow \dots \Rightarrow \tau_m \Rightarrow \iota$ symbols $\mathtt{pred}^i_\pi, \mathtt{suc}^i_\pi, \mathtt{inv}^i_\pi : [\mathtt{string} \times \vec{\sigma} \times \vec{\tau}] \Rightarrow \iota$ and $\mathtt{seed}^i_\pi : [\mathtt{string} \times \vec{\tau}] \Rightarrow \kappa$;
- R is a set of cons-free rules $f(\vec{\ell}) \to r$ with $f \in \Sigma$, each $\ell_i \in \mathcal{T}(\mathcal{C},\mathcal{V})$ and $r \in \mathcal{T}(\mathcal{C} \cup \Sigma, \mathcal{V})$;
- for every string $cs \subseteq I^+$, the set $A_{cs} \subseteq \{(s_1,\dots,s_a) \in \mathcal{T}(\mathcal{C} \cup \Sigma)^a \Vdash s_j : \sigma_j \text{ for } 1 \le j \le a\}$;
- for every string cs, $\langle\cdot\rangle_{cs}$ is a surjective mapping from A_{cs} to $\{0,\dots,P(|cs|)-1\}$;
- writing e.g. $\mathtt{pred}^i_\pi[\vec{s}] : \sigma_i$ for the term $\lambda\vec{y}.\mathtt{pred}^i_\pi(\vec{s},\vec{y})$, the following properties are satisfied:
  - $(\mathtt{seed}^1_\pi[cs],\dots,\mathtt{seed}^a_\pi[cs]) \in A_{cs}$ and $\langle(\mathtt{seed}^1_\pi[cs],\dots,\mathtt{seed}^a_\pi[cs])\rangle_{cs} = P(|cs|)-1$

  and for all $(s_1,\dots,s_a) \in A_{cs}$ with $\langle(s_1,\dots,s_a)\rangle_{cs} = m$:

- $(\mathtt{pred}^1_\pi[cs,\vec{s}],\dots,\mathtt{pred}^a_\pi[cs,\vec{s}])$ and $(\mathtt{suc}^1_\pi[cs,\vec{s}],\dots,\mathtt{suc}^a_\pi[cs,\vec{s}])$ and $(\mathtt{inv}^1_\pi[cs,\vec{s}],\dots,\mathtt{inv}^a_\pi[cs,\vec{s}])$ are all in A_{cs}
- $\langle(\mathtt{pred}^1_\pi[cs,\vec{s}],\dots,\mathtt{pred}^a_\pi[cs,\vec{s}])\rangle_{cs} = \max(m-1,0)$
- $\langle(\mathtt{suc}^1_\pi[cs,\vec{s}],\dots,\mathtt{suc}^a_\pi[cs,\vec{s}])\rangle_{cs} = \min(m+1,P(|cs|)-1)$
- $\langle(\mathtt{inv}^1_\pi[cs,\vec{s}],\dots,\mathtt{inv}^a_\pi[cs,\vec{s}])\rangle_{cs} = P(|cs|)-1-m$
- $\mathtt{zero}_\pi(cs,\vec{s}) \to^*_R \mathtt{true}$ iff $m=0$ and $\mathtt{zero}_\pi(cs,\vec{s}) \to^*_R \mathtt{false}$ iff $m>0$
- if each $s_i \to^*_R t_i$ and $(t_1,\dots,t_a) \in A_{cs}$, then also $\langle(t_1,\dots,t_a)\rangle_{cs} = m$.

It is not hard to see how we would use a P-counting module in the AFS of Figure 2; this results in a k^{th}-order AFS for a k^{th}-order module. Note that this works even if some number representations $(s_1,\dots,s_a)$ are not in normal form: even if we reduce $\vec{s}$ to some tuple $\vec{t}$, the result of the $\mathtt{zero}$ test cannot change from $\mathtt{true}$ to $\mathtt{false}$ or vice versa. Since the algorithm relies heavily on these tests, we may safely assume that terms representing numbers are reduced in a lazy way – as we did in Section 3.2 for the arguments s and t of $\mathtt{main}$.

▶ **Lemma 16.** *There is a first-order ($\lambda n.2^{n+1}$)-counting module.*

Proof. Like in Section 3.2, we will represent a set of numbers – or rather, its encoding as a bit-sequence – by a pair of terms. We let $C_{\mathtt{e}} := (\mathtt{string}\times\mathtt{string},\Sigma,R,A,\langle\cdot\rangle)$, where:

- A_{cs} contains all pairs (s,t) such that (a) all data terms q such that $s \to^*_R q$ or $t \to^*_R q$ are subterms of cs, and (b) for each $q \trianglelefteq cs$ either $s \to^*_R q$ or $t \to^*_R q$, but not both.
- Writing $cs = c_N(\dots(c_1(\rhd))\dots)$, we let $cs_0 = \rhd$, $cs_1 = c_1(\rhd)$ and so on. We let $\langle(s,t)\rangle_{cs} = \sum_{i=0}^{N}\{2^{N-i} \mid s \to^*_R cs_i\}$. That is, $\langle(s,t)\rangle_{cs}$ is the number represented by the bit-sequence $b_0\dots b_N$ where $b_i = 1$ iff $s \to^*_R cs_i$, iff not $t \to^*_R cs_i$ (with b_N the least significant digit).
- Σ consists of the defined symbols introduced in R, which we construct below.

As in Section 3.2, we use non-deterministic selection functions to construct (s,t):

$$\mathtt{either}(x,y) \;\to\; x \qquad \mathtt{either}(x,y) \;\to\; y \qquad \bot \;\to\; \bot$$

The symbol $\bot$ will be used for terms which do not reduce to any data (the $\bot \to \bot$ rule is used to force $\bot \in \mathcal{D}$). For the remaining functions, we consider bitvector arithmetic. First, $2^{N+1}-1$ corresponds to the bit-sequence where each $b_i = 1$:

$$\begin{array}{rcl@{\qquad}rcl}
\mathtt{seed}^1_{\mathtt{e}}(cs) & \to & \mathtt{all}(cs,\bot) & \mathtt{all}(\rhd,q) & \to & \mathtt{either}(\rhd,q) \\
\mathtt{seed}^2_{\mathtt{e}}(cs) & \to & \bot & \mathtt{all}(\underline{\mathtt{a}}(xs),q) & \to & \mathtt{all}(xs,\mathtt{either}(\underline{\mathtt{a}}(xs),q))\ \llbracket\text{for } \underline{\mathtt{a}} \in \mathtt{I}\rrbracket
\end{array}$$

Here, $\mathtt{I} = \{\mathtt{a} \mid a \in I\}$. The inverse function is obtained by flipping the sequence's bits:

$$\mathtt{inv}^1_{\mathtt{e}}(cs,s,t) \;\to\; t \qquad \mathtt{inv}^1_{\mathtt{e}}(cs,s,t) \;\to\; s$$

In order to define $\mathtt{zero}_{\mathtt{e}}$, we must test the value of all bits in the sequence. This is done by forcing an evaluation from s or t to some data term. This test is constructed in such a way that both $\mathtt{true}$ and $\mathtt{false}$ results necessarily reflect the state of s and t; any undesirable non-deterministic choices lead to the evaluation getting stuck.

$$\begin{array}{rcl@{\qquad}rcl}
\mathtt{eqLen}(\rhd,\rhd) & \to & \mathtt{true} & \mathtt{eqLen}(\rhd,\underline{\mathtt{a}}(ys)) & \to & \mathtt{false} \\
\mathtt{eqLen}(\underline{\mathtt{a}}(xs),\underline{\mathtt{b}}(ys)) & \to & \mathtt{eqLen}(xs,ys) & \mathtt{eqLen}(\underline{\mathtt{a}}(xs),\rhd) & \to & \mathtt{false}
\end{array}\left.\vphantom{\begin{array}{c}a\\b\end{array}}\right\}\ \llbracket\text{for } \underline{\mathtt{a}},\underline{\mathtt{b}} \in I\rrbracket$$

$$\begin{array}{rcl@{\qquad}rcl}
\mathtt{bitset}(xs,s,t) & \to & \mathtt{test}(\mathtt{eqLen}(xs,s),\mathtt{eqLen}(xs,t)) & \mathtt{test}(\mathtt{true},x) & \to & \mathtt{true} \\
 & & & \mathtt{test}(x,\mathtt{true}) & \to & \mathtt{false}
\end{array}$$

Then $\mathtt{zero}_{\mathtt{e}}$ simply tests whether the bit is unset for each sublist.

$$\begin{array}{rcl@{\qquad}rcl}
\mathtt{zero}_{\mathtt{e}}(xs,s,t) & \to & \mathtt{zo}(xs,s,t,\mathtt{bitset}(xs,s,t)) & \mathtt{zo}(xs,s,t,\mathtt{true}) & \to & \mathtt{false} \\
\mathtt{zo}(\underline{\mathtt{a}}(xs),s,t,\mathtt{false}) & \to & \mathtt{zero}_{\mathtt{e}}(xs,s,t)\ \llbracket\text{for } \underline{\mathtt{a}} \in I\rrbracket & \mathtt{zo}(\rhd,s,t,\mathtt{false}) & \to & \mathtt{true}
\end{array}$$

For the predecessor function, note that the predecessor of a bit-sequence $b_0 \dots b_{i-1} b 10 \dots 0$ is $b_0 \dots b_{i-1} 01 \dots 1$. We first define a helper function `copy` to copy $b_0 \dots b_{i-1}$:

$$\begin{array}{rcl}
\mathtt{copy}(xs, s, t, \mathtt{false}) & \to & \mathtt{maybeadd}(xs, \mathtt{bitset}(xs, s, t), \mathtt{copy}(\mathtt{tl}(xs), s, t, \mathtt{empty}(xs))) \\
\mathtt{copy}(xs, s, t, \mathtt{true}) & \to & \bot \qquad \mathtt{maybeadd}(xs, \mathtt{true}, q) \to \mathtt{either}(xs, q) \\
 & & \qquad\quad\; \mathtt{maybeadd}(xs, \mathtt{false}, q) \to q \\
\mathtt{empty}(\rhd) & \to & \mathtt{true} \qquad\qquad\qquad \mathtt{tl}(\rhd) \to \rhd \\
\mathtt{empty}(\underline{\mathtt{a}}(x)) & \to & \mathtt{false}\ [\![\text{for } \underline{\mathtt{a}} \in \mathtt{I}]\!] \quad \mathtt{tl}(\underline{\mathtt{a}}(x)) \to x\ [\![\text{for } \underline{\mathtt{a}} \in \mathtt{I}]\!]
\end{array}$$

Then $\mathtt{copy}(xs_{\max(i-1,0)}, s, t, [i = 0])$ reduces to those xs_j with $0 \leq j < i$ where $b_j = 1$, and $\mathtt{copy}(xs_{\max(i-1,0)}, t, s, [i = 0])$ to those with $b_j = 0$. This works because s and t are each other's complement. To define `pred`, we first handle the zero case:

$$\begin{array}{l}
\mathtt{pred}_{\mathtt{e}}^{\underline{\mathtt{i}}}(cs, s, t) \;\to\; \mathtt{pz}^{\underline{\mathtt{i}}}(cs, s, t, \mathtt{zero}_{\mathtt{e}}(cs, s, t)) \quad [\![\text{for } \underline{\mathtt{i}} \in \{1, 2\}]\!] \\
\mathtt{pz}^1(cs, s, t, \mathtt{true}) \;\to\; s \qquad \mathtt{pz}^1(cs, s, t, \mathtt{false}) \;\to\; \mathtt{pmain}^1(cs, s, t, \mathtt{bitset}(cs, s, t)) \\
\mathtt{pz}^2(cs, s, t, \mathtt{true}) \;\to\; t \qquad \mathtt{pz}^2(cs, s, t, \mathtt{false}) \;\to\; \mathtt{pmain}^2(cs, s, t, \mathtt{bitset}(cs, s, t))
\end{array}$$

Then, $\mathtt{pmain}(xs_N, s, t, [b_N = 1])$ flips the bits $b_N, b_{N-1}, \dots$ until an index is encountered where $b_i = 1$; this last bit is flipped, and the remaining bits copied. Formally:

$$\begin{array}{rcl}
\mathtt{pmain}^1(xs, s, t, \mathtt{true}) & \to & \mathtt{copy}(\mathtt{tl}(xs), s, t, \mathtt{empty}(xs)) \\
\mathtt{pmain}^2(xs, s, t, \mathtt{true}) & \to & \mathtt{either}(xs, \mathtt{copy}(\mathtt{tl}(xs), t, s, \mathtt{empty}(xs))) \\
\mathtt{pmain}^1(xs, s, t, \mathtt{false}) & \to & \mathtt{either}(xs, \mathtt{pmain}^1(\mathtt{tl}(xs), s, t, \mathtt{bitset}(\mathtt{tl}(xs), s, t))) \\
\mathtt{pmain}^2(xs, s, t, \mathtt{false}) & \to & \mathtt{pmain}^2(\mathtt{tl}(xs), s, t, \mathtt{bitset}(\mathtt{tl}(xs), s, t))
\end{array}$$

Finally, we observe that $x + 1 = N - ((N - x) - 1)$ and for $x = N$ also $\min(x + 1, N) = N - (\max((N - x) - 1, 0))$. Thus, we may define $\mathtt{suc}(b)$ as $\mathtt{inv}(\mathtt{pred}(\mathtt{inv}(x)))$. Taking pairing into account and writing out the definition, this simplifies to:

$$\mathtt{suc}^1(cs, s, t) \;\to\; \mathtt{pred}^2(cs, t, s) \qquad \mathtt{suc}^2(cs, s, t) \;\to\; \mathtt{pred}^1(cs, t, s) \qquad ◀$$

Having Lemma 16 as a basis, we can define composite modules. Here, we give fewer details than for Lemma 16 as the constructions use many of the same ideas.

▶ **Lemma 17.** *If there exist a P-counting module C_π and a Q-counting module C_ρ, both of order at most k, then there is a $(\lambda n.P(n) \cdot Q(n))$-counting module $C_{\pi \cdot \rho}$ of order at most k.*

Proof Sketch. Let $C_\pi ::= ([\sigma_1 \times \dots \times \sigma_a], \Sigma^\pi, R^\pi, A^\pi, \langle\cdot\rangle^\pi)$ and $C_\rho ::= ([\tau_1 \times \dots \times \tau_b], \Sigma^\rho, R^\rho, A^\rho, \langle\cdot\rangle^\rho)$. We will, essentially, represent the numbers $i \in \{0, \dots, P(|cs|) \cdot Q(|cs|) - 1\}$ by a pair (i_1, i_2) with $0 \leq i_1 < P(|cs|)$ and $0 \leq i_2 < Q(|cs|)$, such that $i = i_1 \cdot Q(|cs|) + i_2$. This is done by defining $A^{\pi \cdot \rho}_{cs} = \{(u_1, \dots, u_a, v_1, \dots, v_b) \mid (u_1, \dots, u_a) \in A^\pi_{cs} \wedge (v_1, \dots, v_b) \in A^\rho_{cs}\}$, and $\langle(\vec{u}, \vec{v})\rangle^{\pi \cdot \rho}_{cs} = \langle(\vec{u})\rangle^\pi_{cs} \cdot Q(|cs|) + \langle(\vec{v})\rangle^\rho_{cs}$. The signature of defined symbols and rules of $C_{\pi \cdot \rho}$ are straightforwardly defined as well, extending those in C_π and C_ρ; for instance:

$$\mathtt{zero}_{\pi \cdot \rho}(cs, u_1, \dots, u_a, v_1, \dots, v_b) \;\to\; \mathtt{and}(\mathtt{zero}_\pi(cs, u_1, \dots, u_a), \mathtt{zero}_\rho(cs, v_1, \dots, v_b))$$

$$\mathtt{and}(\mathtt{true}, x) \;\to\; x \qquad \mathtt{and}(\mathtt{false}, y) \;\to\; \mathtt{false} \qquad ◀$$

▶ **Lemma 18.** *If there is a P-counting module C_π of order k, then there is a $(\lambda n.2^{P(n)})$-counting module $C_{\mathtt{p}[\pi]}$ of order $k + 1$.*

Proof Sketch. We represent every bitstring $b_{P(|cs|)-1 \cdots b_0}$ as a function of type $\sigma_1 \Rightarrow \dots \Rightarrow \sigma_a \Rightarrow \mathtt{bool}$. The various functions are defined as bitvector operations. For example:

$$\mathtt{seed}_{\mathtt{p}[\pi]}(cs, k_1, \dots, k_a) \;\to\; \mathtt{true} \qquad \mathtt{inv}_{\mathtt{p}[\pi]}(cs, F, k_1, \dots, k_a) \;\to\; \mathtt{not}(F \cdot k_1 \cdots k_a)$$

$$\begin{array}{rcl}
\mathtt{zero}_{\mathtt{p}[\pi]}(cs, F) & \to & \mathtt{zero'}_{\mathtt{p}[\pi]}(cs, \mathtt{seed}^1_\pi[cs], \dots, \mathtt{seed}^a_\pi[cs], F) \\
\mathtt{zero'}_{\mathtt{p}[\pi]}(cs, k_1, \dots, k_a, F) & \to & \mathtt{ztest}_{\mathtt{p}[\pi]}(F \cdot k_1 \cdots k_a, \mathtt{zero}_\pi(cs, k_1, \dots, k_a), cs, \\
 & & \quad k_1, \dots, k_a, F) \\
\mathtt{ztest}_{\mathtt{p}[\pi]}(\mathtt{true}, z, cs, \vec{k}, F) & \to & \mathtt{false} \\
\mathtt{ztest}_{\mathtt{p}[\pi]}(\mathtt{false}, \mathtt{true}, cs, \vec{k}, F) & \to & \mathtt{true} \\
\mathtt{ztest}_{\mathtt{p}[\pi]}(\mathtt{false}, \mathtt{false}, cs, \vec{k}, F) & \to & \mathtt{zero'}_{\mathtt{p}[\pi]}(cs, \mathtt{pred}^1_\pi[cs, \vec{k}], \dots, \mathtt{pred}^a_\pi[cs, \vec{k}], F)
\end{array} \blacktriangleleft$$

Note that, for instance, $\mathtt{seed}_{\mathtt{p}[\pi]}[cs]$ is $\lambda k_1 \dots k_a.\mathtt{seed}_{\mathtt{p}[\pi]}(cs, k_1, \dots, k_a)$: the additional parameters k_i should be seen as indexing the result of the function.

We obtain:

▶ **Theorem 19.** *Any decision problem in* E^k*TIME can be accepted by a* k^{th}*-order AFS.*

Proof. Following the construction in this section, it suffices if we can find a k^{th}-order counting module counting up to $\exp_2^k(a \cdot n)$ where n is the size of the input and a a fixed positive integer. Lemma 16 gives a first-order $\lambda n.2^{n+1}$-counting module, and by iteratively using Lemma 17 we obtain $\lambda n.(2^{n+1})^a = \lambda n.2^{a(n+1)}$ for any a. Iteratively applying Lemma 18 on the result gives a k^{th}-order $\lambda n.\exp_2^k(a \cdot (n+1))$-counting module. ◀

5 Finding normal forms

In the previous section we have seen that every function in E^kTIME can be implemented by a cons-free k^{th}-order AFS. Towards a characterization result, we must therefore show the converse: that every function implemented by a cons-free k^{th}-order AFS is in E^kTIME.

To achieve this goal, we will now give an algorithm that, on input any basic term in an AFS of order k, will output its set of data normal forms in E^kTIME in the size of the term.

A key idea is to associate terms of higher-order type to functions. We define:

$$\begin{array}{rcl}
[\![\iota]\!] & = & \mathbb{P}(\{s \mid s \in \mathcal{B} \wedge \vdash s : \iota\}) \quad \text{for } \iota \in \mathcal{S} \quad (\text{so a set of subsets of } \mathcal{B}) \\
[\![\sigma \Rightarrow \tau]\!] & = & [\![\tau]\!]^{[\![\sigma]\!]} \quad (\text{so the set of functions from } [\![\sigma]\!] \text{ to } [\![\tau]\!])
\end{array}$$

Intuitively, an element of $[\![\iota]\!]$ represents a set of possible reducts of a term $s : \iota$, while an element of $[\![\sigma \Rightarrow \tau]\!]$ represents the function defined by some $\lambda x.s : \sigma \Rightarrow \tau$. Since – as induction on the structure of σ shows – each $[\![\sigma]\!]$ is *finite*, we can define the following algorithm to find all normal forms of a given basic term. In the algorithm, we build functions $\mathsf{Confirmed}^0, \mathsf{Confirmed}^1, \dots$, each mapping statements $f(A_1, \dots, A_n) \approx t$ to a value in $\{\top, \bot\}$. Intuitively, $\mathsf{Confirmed}^i[f(\vec{A}) \approx t]$ denotes whether, in step i in the algorithm, we have confirmed that $f(s_1, \dots, s_n) \to^*_{\mathcal{R}} t$, where each A_i represents the corresponding s_i.

▶ **Algorithm 20.**

Input: A basic term $s = g(t_1, \dots, t_m)$.

Output: The set of data normal forms of s. Note that this set may be empty.

Set $\mathcal{B} := \mathcal{B}_s$. For all $f : [\sigma_1 \times \dots \times \sigma_n] \Rightarrow \iota \in \mathcal{D}$, all $A_1 \in [\![\sigma_1]\!], \dots, A_n \in [\![\sigma_n]\!]$, all $t \in [\![\iota]\!]$, we let $\mathsf{Confirmed}^0[f(A_1, \dots, A_n) \approx t] := \bot$. Now, for all such $f, \vec{A}, t$ and all $i \in \mathbb{N}$:

- if $\mathsf{Confirmed}^i[f(\vec{A}) \approx t] = \top$, then $\mathsf{Confirmed}^{i+1}[f(\vec{A}) \approx t] := \top$;
- otherwise, for all rules $f(\ell_1, \dots, \ell_n) \to r \in \mathcal{R}$, for all substitutions γ on domain $FV(f(\vec{\ell})) \setminus \{\vec{\ell}\}$ (so on those variables occurring below constructors) such that $\ell_j\gamma \in A_j$ for all j with ℓ_j not a variable (A_j is a set of terms since ℓ_j, a non-variable proper constructor term, must have base type), let η be the function such that for each $\ell_j \in \mathcal{V}$, $\eta(\ell_j) = A_j$, and test whether $t \in \mathcal{NF}^i(r\gamma, \eta)$. If there are a rule and substitution where this test succeeds, let $\mathsf{Confirmed}^{i+1}[f(\vec{A}) \approx t] := \top$, otherwise let $\mathsf{Confirmed}^{i+1}[f(\vec{A}) \approx t] := \bot$.

Here, $\mathcal{NF}^i(s,\eta)$ is defined recursively for $\mathcal{B}$-safe terms s and functions η mapping all variables $x : \sigma$ in $FV(s)$ to an element of $[\![\sigma]\!]$, as follows:

- if s is a data term, then $\mathcal{NF}^i(s,\eta) := \{s\}$;
- if s is a variable, then $\mathcal{NF}^i(s,\eta) := \eta(s)$;
- if $s = f(s_1,\dots,s_n)$ with $f \in \mathcal{D}$, then $\mathcal{NF}^i(s,\eta)$ is the set of all $t \in \mathcal{B}$ such that $\mathsf{Confirmed}^i[f(\mathcal{NF}^i(s_1,\eta),\dots,\mathcal{NF}^i(s_n,\eta)) \approx t] = \top$;
- if $s = u \cdot v$, then $\mathcal{NF}^i(s,\eta) = \mathcal{NF}^i(u,\eta)(\mathcal{NF}^i(v,\eta))$;
- if $s =_\alpha \lambda x.t : \sigma \Rightarrow \tau$ where $x \notin \mathtt{domain}(\eta)$, then $\mathcal{NF}^i(s,\eta) :=$ the function mapping $A \in [\![\sigma]\!]$ to $\mathcal{NF}^i(t,\eta \cup [x := A])$.

When $\mathsf{Confirmed}^{i+1}[f(\vec{A}) \approx t] = \mathsf{Confirmed}^i[f(\vec{A}) \approx t]$ for all statements, the algorithm ends; we let $I := i+1$ and return $\{t \in \mathcal{B} \mid \mathsf{Confirmed}^I[g(\{t_1\},\dots,\{t_m\}) \approx t] = \top\}$.

As $\mathcal{D}$, $\mathcal{B}$ and all $[\![\sigma_i]\!]$ are all finite, and the number of positions at which $\mathsf{Confirmed}^i$ is $\top$ increases in every step, the algorithm always terminates. The intention is that $\mathsf{Confirmed}^I$ reflects rewriting for basic terms. This result is stated formally in Theorem 22.

▶ **Example 21.** Consider the palindrome AFS in Example 11, with starting term $s = \mathtt{1}(\mathtt{0}(\rhd))$. Then $\mathcal{B}_s = \{\mathtt{1}(\mathtt{0}(\rhd)), \mathtt{0}(\rhd), \rhd, \mathtt{true}, \mathtt{false}\}$. Then we have $[\![\mathtt{bool}]\!] = \{\emptyset, \{\mathtt{true}\}, \{\mathtt{false}\}, \{\mathtt{true}, \mathtt{false}\}\}$ and $[\![\mathtt{string}]\!]$ is the set containing all eight subsets of $\{\mathtt{1}(\mathtt{0}(\rhd)), \mathtt{0}(\rhd), \rhd\}$. Thus, there are $8 \cdot 8 \cdot 2$ statements of the form $\mathtt{palindrome}(A,B) \approx t$, $4 \cdot 4 \cdot 2$ of the form $\mathtt{and}(A,B) \approx t$ and so on, totalling 432 statements to be considered in every step.

We consider one step, determining $\mathsf{Confirmed}^1[\mathtt{chk_1}(\{\mathtt{1}(\mathtt{0}(\rhd))\}, \{\mathtt{0}(\rhd), \rhd\}) \approx \mathtt{true}]$. There are two viable combinations of a rule and a substitution: $\mathtt{chk_1}(\mathtt{1}(xs), \mathtt{0}(ys)) \to \mathtt{chk_1}(xs, ys)$ with substitution $\gamma = [xs := \mathtt{0}(\rhd), ys := \rhd]$ and $\mathtt{chk_1}(\mathtt{1}(xs), \rhd) \to \mathtt{true}$ with $\gamma = [xs := \mathtt{0}(\rhd)]$. Consider the first. As there are no functional variables, η is empty and we need to determine whether $\mathtt{true} \in \mathcal{NF}^1(\mathtt{chk_1}(\mathtt{0}(\rhd), \rhd), \emptyset)$. This fails, because $\mathsf{Confirmed}^0[\xi] = \bot$ for all statements ξ. However, the check for the second rule, $\mathtt{true} \in \mathcal{NF}^1(\mathtt{true}, \emptyset)$, succeeds. Thus, we mark $\mathsf{Confirmed}^1[\mathtt{chk_1}(\{\mathtt{1}(\mathtt{0}(\rhd))\}, \{\mathtt{0}(\rhd), \rhd\}) \approx \mathtt{true}] = \top$.

▶ **Theorem 22.** *Let $f : [\iota_1 \times \dots \times \iota_n] \Rightarrow \kappa \in \mathcal{D}$ and $s_1 : \iota_1, \dots, s_n : \iota_n, t : \kappa$ be data terms. Then $\mathsf{Confirmed}^I[f(\{s_1\},\dots,\{s_n\}) \approx t] = \top$ if and only if $f(\vec{s}) \to_{\mathcal{R}}^* t$.*

Proof Sketch. Define a labeled variation of $\mathcal{R}$:

$$\mathcal{R}_{\mathtt{lab}} = \{f_{i+1}(\vec{\ell}) \to \mathsf{label}_i(r) \mid f(\vec{\ell}) \to r \in \mathcal{R} \wedge i \in \mathbb{N}\} \cup \{f_{i+1}(\vec{x}) \to f_i(\vec{x}) \mid f \in \mathcal{D} \wedge i \in \mathbb{N}\}$$

Here label_i replaces each defined symbol f by a symbol f_i. Then $\mathcal{R}_{\mathtt{lab}}$ is infinite, and $f(\vec{s}) \to_{\mathcal{R}}^* t$ iff some $f_i(\vec{s}) \to_{\mathcal{R}_{\mathtt{lab}}}^* t$. Furthermore, $\to_{\mathcal{R}_{\mathtt{lab}}}$ is terminating (even if $\to_{\mathcal{R}}$ is not!) as is provable using, e.g., the *Computability Path Ordering* [9]. Thus, $\to_{\mathcal{R}_{\mathtt{lab}}}$ is a well-founded binary relation on the set of labeled terms, and we can hence perform induction.

Consider the arguments passed to $\mathsf{Confirmed}^i$ in the recursive process: $\mathcal{NF}^i$ is defined using tests of the form $\mathsf{Confirmed}^i[f(\mathcal{NF}^i(s_1,\eta),\dots,\mathcal{NF}^i(s_n,\eta))] = \top$, where each $\eta(x)$ itself has the form $\mathcal{NF}^j(t,\eta')$. To formally describe this, let an *$\mathcal{NF}$-substitution* be recursively defined as a mapping from some (possibly empty) set $V \subseteq \mathcal{V}$ such that for each $x : \sigma \in V$ there are an $\mathcal{NF}$-substitution δ and a term s with $\vdash s : \sigma$ such that $\eta(x) = \mathcal{NF}^j(s,\delta)$ for some j. For an $\mathcal{NF}$-substitution η on domain V, we define $\overline{\eta}(x) = x$ for $x \notin V$, and $\overline{\eta}(x) = \mathsf{label}_j(s)\overline{\zeta}$ for $x \in V$ with $\eta(x) = \mathcal{NF}^j(s,\zeta)$. Then the following two claims can be derived by mutual induction on q ordered with $\to_{\mathcal{R}_{\mathtt{lab}}} \cup \rhd$ (all η_j and ζ are $\mathcal{NF}$-substitutions):

- $\mathsf{Confirmed}^i[f(\mathcal{NF}^{j_1}(s_1,\eta_1),\dots,\mathcal{NF}^{j_n}(s_n,\eta_n)) \approx t] = \top$ *if and only if* $q := f_i(\mathsf{label}_{j_1}(s_1)\overline{\eta_1},\dots,\mathsf{label}_{j_n}(s_n)\overline{\eta_n}) \to_{\mathcal{R}_{\mathtt{lab}}}^* t$;

- $t \in \mathcal{NF}^i(u,\zeta)(\mathcal{NF}^{j_1}(s_1,\eta_1),\dots,\mathcal{NF}^{j_n}(s_n,\eta_n))$ *if and only if* $q := (\mathsf{label}_i(u)\overline{\zeta}) \cdot \mathsf{label}_{j_1}(s_1)\overline{\eta_1} \cdots \mathsf{label}_{j_n}(s_n)\overline{\eta_n} \to^*_{\mathcal{R}_{\mathtt{lab}}} t$.

Since, if we refrain from stopping the process in step I, we have $\mathsf{Confirmed}^I = \mathsf{Confirmed}^{I+1} = \mathsf{Confirmed}^{I+2} = \dots$, the theorem follows because $f(\vec{s}) \to^*_{\mathcal{R}} t$ iff some $f_i(\vec{s}) \to^*_{\mathcal{R}_{\mathtt{lab}}} t$. ◀

It remains to prove that Algorithm 20 runs sufficiently fast.

▶ **Theorem 23.** *If $(\mathcal{F},\mathcal{R})$ has order k, then Algorithm 20 runs in time $O(\exp_2^k(m \cdot n))$ for some m.*

Proof. Write $N := |\mathcal{B}|$. As $\mathcal{R}$ and $\mathcal{F}$ are fixed, N is linear in the size of the only input, s. We claim that if $k, i \in \mathbb{N}$ are such that σ has at most *order* k, and the *longest sequence* $\sigma_1 \Rightarrow \dots \Rightarrow \sigma_n \Rightarrow \iota$ occurring in σ has length $n+1 \le i$, then $\mathtt{card}([\![\sigma]\!]) \le \exp_2^{k+1}(i^k \cdot N)$.

(Proof of claim.) Observe first that $\mathbb{P}(\mathcal{B})$ has cardinality 2^N. Proceed by induction on the form of σ. Note that we can write σ in the form $\sigma_1 \Rightarrow \dots \Rightarrow \sigma_n \Rightarrow \iota$ with $n < i$ and each σ_j having order at most $k-1$ (as $n = 0$ when given a 0^{th}-order type). We have:

$$\begin{aligned}
\mathtt{card}([\![\sigma_1 \Rightarrow \dots \Rightarrow \sigma_n \Rightarrow \iota]\!]) &= \mathtt{card}((\cdots([\![\iota]\!]^{[\![\sigma_n]\!]})^{[\![\sigma_{n-1}]\!]}\cdots)^{[\![\sigma_1]\!]}) = \mathtt{card}([\![\iota]\!])^{\mathtt{card}([\![\sigma_n]\!])\cdots\mathtt{card}([\![\sigma_1]\!])} \\
&\le 2^{N\cdot\mathtt{card}([\![\sigma_n]\!])\cdots\mathtt{card}([\![\sigma_1]\!])} \le 2^{N\cdot\exp_2^k(i^k\cdot N)\cdots\exp_2^k(i^k\cdot N)} \text{(by IH)} \\
&= 2^{N\cdot\exp_2^k(i^k\cdot N)^n} \le 2^{\exp_2^k(i^k\cdot N\cdot n+N)} \text{(by induction on } k\text{)} \\
&= \exp_2^{k+1}(n\cdot i^k\cdot N+N) \le \exp_2^{k+1}(i\cdot i^k\cdot N) = \exp_2^{k+1}(i^{k+1}\cdot N) \\
&\quad \text{(because } n\cdot i^k+1 \le (n+1)\cdot i^k \le i\cdot i^k\text{)}
\end{aligned}$$

(End of proof of claim.)

Since, in a k^{th}-order AFS, all types occurring in type declarations have order at most $k-1$, there is some i (depending solely on $\mathcal{F}$) such that all sets $[\![\sigma]\!]$ in the algorithm have cardinality $\le \exp_2^k(i^{k-1}\cdot N)$. Writing a for the maximal arity in $\mathcal{F}$, there are at most $|\mathcal{D}|\cdot\exp_2^k(i^{k-1}\cdot N)^a\cdot N \le |\mathcal{D}|\cdot\exp_2^k((i^{k-1}\cdot a+1)\cdot N)$ distinct statements $f(\vec{A}) \approx t$.

Writing $m := i^{k-1}\cdot a+1$ and $X := |\mathcal{D}|\cdot\exp_2^k(m\cdot N)$, we thus find: the algorithm has at most $I \le X+2$ steps, and in each step we consider at most X statements φ where $\mathsf{Confirmed}^i[\varphi] = \bot$. For every applicable rule, there are at most $(2^N)^a$ different substitutions γ, so we have to test a statement $t \in \mathcal{NF}^i(r\gamma,\eta)$ at most $X\cdot(X+2)\cdot|\mathcal{R}|\cdot 2^{aN}$ times. The exact cost of calculating $\mathcal{NF}^i(r\gamma,\eta)$ is implementation-specific, but is certainly bounded by some polynomial $P(X)$ (which depends on the form of r). This leaves the total time cost of the algorithm at $O(X\cdot(X+1)\cdot 2^{aN}\cdot P(X)) = O(P'(\exp_2^k(m\cdot N)))$ for some polynomial P' and constant m. As E^kTIME is robust under taking polynomials, the result follows. ◀

▶ **Theorem 24.** *Let $k \ge 1$. A set $S \subseteq \{0,1\}^+$ is in E^kTIME iff there is an AFS of order k that accepts S.*

Proof. If $S \in \text{E}^k\text{TIME}$, Theorem 19 shows that it is accepted by an AFS of order k. Conversely, if there is an AFS of order k that accepts S, Theorem 23 shows that we can find whether any basic term reduces to $\mathtt{true}$ in time $O(\exp_2^k(m\cdot n))$ for some m, and thus $S \in \text{E}^k\text{TIME}$. ◀

▶ Remark. Observe that Theorem 24 concerns *extensional* rather than *intensional* behavior of cons-free AFSs: a cons-free AFS may take arbitrarily many steps to reduce its input to normal form, even if it accepts a set that a Turing machine may decide in a bounded number of steps. However, Algorithm 20 can often find the possible results of an AFS faster than evaluating the AFS would take, by avoiding duplicate calculations.

$$
\begin{array}{lclclcl}
\bot{::}t & \to & t & \mathtt{rnd} & \to & \mathtt{I} & \\
\mathtt{rnd} & \to & \mathtt{0} & \mathtt{rnd} & \to & \mathtt{B} &
\end{array}
\qquad
\begin{array}{rcl}
\mathtt{translate}(\mathtt{0}(xs)) & \to & \mathtt{0} :: \mathtt{translate}(xs) \\
\mathtt{translate}(\mathtt{1}(xs)) & \to & \mathtt{I} :: \mathtt{translate}(xs) \\
\mathtt{translate}(\rhd) & \to & \mathtt{B} :: \mathtt{translate}(\rhd) \\
\mathtt{translate}(\rhd) & \to & \rhd \\
\mathtt{equal}(xl, xl) & \to & \mathtt{true}
\end{array}
$$

$$
\begin{array}{l}
\mathtt{rndtape}(x) \to \rhd \\
\mathtt{rndtape}(x) \to \mathtt{rnd} :: \mathtt{rndtape}(x)
\end{array}
$$

$$
\begin{array}{rcl}
\mathtt{start}(cs) & \to & \mathtt{run}(\mathtt{startstate}, \rhd, \mathtt{B}, \mathtt{translate}(cs)) \\
\mathtt{run}(\underline{\mathtt{s}}, xl, \underline{\mathtt{r}}, yl) & \to & \mathtt{shift}(\underline{\mathtt{t}}, xl, \underline{\mathtt{w}}, yl, \underline{\mathtt{d}}) \quad [\![\text{for every transition } \underline{\mathtt{s}} \overset{\underline{\mathtt{r}}/\underline{\mathtt{w}}\ \underline{\mathtt{d}}}{\Longrightarrow} \underline{\mathtt{t}}]\!] \\
\mathtt{shift}(s, xl, c, yl, d) & \to & \mathtt{shift}_1(s, xl, c, yl, d, \mathtt{rnd}, \mathtt{rndtape}(\mathtt{0}), \mathtt{rndtape}(\mathtt{I})) \\
\mathtt{shift}_1(s, xl, c, yl, d, \underline{\mathtt{b}}, t, t) & \to & \mathtt{shift}_2(s, xl, c, yl, d, \underline{\mathtt{b}}, t) \quad [\![\text{for every } \underline{\mathtt{b}} \in \{\mathtt{0}, \mathtt{I}, \mathtt{B}\}]\!] \\
\mathtt{shift}_2(s, xl, c, yl, \mathtt{R}, z, t) & \to & \mathtt{shift}_3(s, c :: xl, z, t, \mathtt{equal}(yl, z :: t)) \\
\mathtt{shift}_2(s, xl, c, yl, \mathtt{L}, z, t) & \to & \mathtt{shift}_3(s, t, z, c :: yl, \mathtt{equal}(xl, z :: t)) \\
\mathtt{shift}_3(s, xl, c, yl, \mathtt{true}) & \to & \mathtt{run}(s, xl, c, yl)
\end{array}
$$

Figure 3 A first-order non-left-linear AFS that simulates a Turing machine.

6 Changing the restrictions

In the presence of non-determinism, minor syntactical changes can make a large difference in expressivity. We briefly consider two natural changes here.

6.1 Non-left-linearity

Recall that we imposed three restrictions: the rules in $\mathcal{R}$ must be *constructor rules*, *left-linear* and *cons-free*. Dramatically, dropping the restriction on left-linearity allows us to decide every Turing-decidable set using first-order systems. This is demonstrated by the first-order AFS in Figure 3 which simulates an arbitrary Turing Machine on input alphabet $I = \{0, 1\}$. Here, a tape $x_0 \dots x_n \sqcup \sqcup \dots$ with the tape head at position i is represented by a triple $(x_{i-1}{::} \cdots {::}x_0,\ x_i,\ x_{i+1}{::} \cdots {::}x_n)$, where the "list constructor" :: is a *defined symbol*, ensured by a rule which never fires. To split such a list into a head and tail, the AFS non-deterministically generates a *new* head and tail, makes sure they are fully evaluated, and uses a non-left-linear rule to test whether their combination corresponds to the original list.

6.2 Product Types

Unlike AFSs, Jones' minimal language in [14] employs a *pairing constructor*, essentially admitting terms $(s, t) : \iota \times \kappa$ if $\vdash s : \iota$ and $\vdash t : \kappa$ are data terms or themselves pairs. This is not in conflict with the cons-freeness requirement due to type restrictions: it does not allow the construction of an arbitrarily large structure of fixed type. In our (non-deterministic) setting, however, pairing is significantly more powerful. Following the ideas of Section 4, one can count up to arbitrarily large numbers: for an input string $x_n(\dots(x_1(\rhd)))$ of length n,

- the counting module C_0 represents $i \in \{0, \dots, n\}$ by a substring $x_i(\dots(x_1(\rhd))) : \mathtt{string}$;
- given a $(\lambda n.\exp_2^k(n+1))$-counting module C_k, we let C_{k+1} represent a number b with bit representation $b_0 \dots b_N$ (for $N < \exp_2^k(n+1)$) as the pair (s, t) – a term! – where s reduces to representations of those bits set to 1, and t to representations of bits set to 0.

Then for instance a number in $\{0, \dots, 2^{2^{n+1}} - 1\}$ is represented by a pair $(s, t) : (\mathtt{string} \times \mathtt{string}) \times (\mathtt{string} \times \mathtt{string})$, where s and t themselves are *not* pairs; rather, they are both terms reducing to a variety of different pairs. A membership test would take the form

$$\texttt{elem}_2(k,(s,t)) \to \texttt{elemtest}(\texttt{equal}_1(k,s),\texttt{equal}_1(k,t))$$
$$\texttt{elemtest}(\texttt{true},x) \to \texttt{true} \qquad\qquad \texttt{elemtest}(x,\texttt{true}) \to \texttt{false}$$

with the rule for $\texttt{equal}_1$ having the form $\texttt{equal}_1((s_1,t_1),(s_2,t_2)) \to r$. That is, the rule *forces a partial evaluation*. This is possible because a "false constructor" (i.e., a syntactic structure that rules can match) is allowed to occur above non-data terms.

7 Future work

In this paper, we have considered the expressive power of cons-free term rewriting, and seen that restricting data order results in characterizations of different classes. A natural direction for future work is to consider further restrictions, either on rule formation, reduction strategy, or both. Following Jones [14], we suspect that restricting to innermost evaluation will give the hierarchy $\text{P} \subseteq \text{EXPTIME} \subseteq \text{EXP}^2\text{TIME} \subsetneq \cdots$. Furthermore, we conjecture that a combination of higher-order rewriting and restrictions on rule formation, possibly together with additions such as product types, may yield characterizations of a wide range of classes, including non-deterministic classes like NP or very small classes like LOGTIME.

References

1 M. Avanzini, N. Eguchi, and G. Moser. A new order-theoretic characterisation of the polytime computable functions. In *APLAS*, volume 7705 of *LNCS*, pages 280–295, 2012. doi:10.1007/978-3-642-35182-2_20.

2 M. Avanzini and G. Moser. Closing the gap between runtime complexity and polytime computability. In *RTA*, volume 6 of *LIPIcs*, pages 33–48, 2010. doi:10.4230/LIPIcs.RTA.2010.33.

3 M. Avanzini and G. Moser. Polynomial path orders. *LMCS*, 9(4), 2013. doi:10.2168/LMCS-9(4:9)2013.

4 P. Baillot. From proof-nets to linear logic type systems for polynomial time computing. In *TLCA*, volume 4583 of *LNCS*, pages 2–7, 2007. doi:10.1007/978-3-540-73228-0_2.

5 P. Baillot, M. Gaboardi, and V. Mogbil. A polytime functional language from light linear logic. In *ESOP*, volume 6012 of *LNCS*, pages 104–124, 2010. doi:10.1007/978-3-642-11957-6_7.

6 P. Baillot and U. Dal Lago. Higher-Order Interpretations and Program Complexity. In *CSL*, volume 16 of *LIPIcs*, pages 62–76, 2012. doi:10.4230/LIPIcs.CSL.2012.62.

7 S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992. doi:10.1007/BF01201998.

8 S. Bellantoni, K. Niggl, and H. Schwichtenberg. Higher type recursion, ramification and polynomial time. *Annals of Pure and Applied Logic*, 104(1–3):17–30, 2000. doi:10.1016/S0168-0072(00)00006-3.

9 F. Blanqui, J. Jouannaud, and A. Rubio. The computability path ordering: The end of a quest. In *CSL*, volume 5213 of *LNCS*, pages 1–14, 2008.

10 G. Bonfante. Some programming languages for logspace and ptime. In *AMAST*, volume 4019 of *LNCS*, pages 66–80, 2006. doi:10.1007/11784180_8.

11 D. de Carvalho and J. Simonsen. An implicit characterization of the polynomial-time decidable sets by cons-free rewriting. In *RTA-TLCA*, volume 8560 of *LNCS*, pages 179–193, 2014.

12 M. Hofmann. Type systems for polynomial-time computation, 1999. Habilitationsschrift.

13 N. Jones. *Computability and Complexity from a Programming Perspective.* MIT Press, 1997.

14 N. Jones. The expressive power of higher-order types or, life without CONS. *JFP*, 11(1):55–94, 2001.

15 J. Jouannaud and A. Rubio. The higher-order recursive path ordering. In *LICS*, pages 402–411, 1999.

16 C. Kop and J. Simonsen. Complexity hierarchies and higher-order cons-free rewriting (extended version). Technical report, University of Copenhagen, 2016. Available online at the authors' homepages.

17 L. Kristiansen and K. Niggl. On the computational complexity of imperative programming languages. *TCS*, 318(1–2):139–161, 2004. `doi:10.1016/j.tcs.2003.10.016`.

18 R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *TCS*, 192(1):3–29, 1998.

19 C. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

20 M. Sipser. *Introduction to the Theory of Computation.* Thomson Course Technology, 2006.

21 F. van Raamsdonk. Higher-order rewriting. In *Term Rewriting Systems*, Chapter 11. Cambridge University Press, 2003.

Weighted Relational Models for Mobility

James Laird[*]

Department of Computer Science, University of Bath, UK

Abstract

We investigate operational and denotational semantics for computational and concurrent systems with mobile names which capture their computational properties. For example, various properties of fixed networks, such as shortest or longest path, transition probabilities, and secure data flows, correspond to the "sum" in a semiring of the weights of paths through the network: we aim to model networks with a dynamic topology in a similar way. Alongside rich computational formalisms such as the λ-calculus, these can be represented as terms in a calculus of solos with weights from a complete semiring R, so that reduction associates a weight in R to each reduction path.

Taking inspiration from differential nets, we develop a denotational semantics for this calculus in the category of sets and R-weighted relations, based on its differential and compact-closed structure, but giving a simple, syntax-independent representation of terms as matrices over R. We show that this corresponds to the sum in R of the values associated to its independent reduction paths, and that our semantics is fully abstract with respect to the observational equivalence induced by sum-of-paths evaluation.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages, F.4.1 Mathematical Logic

Keywords and phrases Mobility, Concurrency, Solos

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.24

1 Introduction

Calculi based on name mobility [20, 23, 11] are well established as an elegant and expressive formalism for describing computation and communication in a broad range of concurrent systems. Semantics for these calculi, such as labelled transition systems, typically focus on local properties of processes – in particular, bisimulation equivalence. In this article we introduce resource-sensitive operational and denotational semantics for mobility which can capture quantitative properties of the whole system being modelled for a variety of potential resources (cost, security level, probability, . . .). Potentially, this will allow algorithmic reasoning principles developed for models such as weighted graphs to be extended to more dynamic systems.

1.1 Related Work

We will describe operational and denotational interpretations of the *solos calculus* [17] – that is, the fusion calculus [23] without any sequentialization in the form of input or output prefixing. The solos calculus presents name mobility in a particularly pure form, without any explicit notion of causal or temporal dependency, with an elegant graphical representation via *solo diagrams* [22]. However, Laneve and Victor [17] have shown that sequentialization

[*] Research supported by EPSRC grant EP/K037633/1.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 24; pp. 24:1–24:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

protocols may be written in the calculus using name-passing, recovering the expressive power of the π-calculus, for example, and establishing the solos calculus as an elegant and economical syntax for describing mobility in highly distributed systems.

We take inspiration from work by Ehrhard and Laurent [7], who have developed an interpretation of the solos calculus in the formal graphical language of *differential nets* [10], establishing a striking connections between name mobility and the differential structure [4] which underlies our model. There are some significant differences: our semantics includes replication (unlike the differential net semantics) but also the *acyclic* terms [7], which have some pathological behaviours.

We develop and extend work by Manzonetto, McCusker, Pagani and the author [16], which introduced operational and denotational semantics for nondeterministic functional programs with weights from a continuous semiring R. Each terminating reduction path in the operational semantics may be associated with a value in R by multiplying the weights ecountered, giving an interpretation of programs as a sum in R of the weights of their reduction paths. The corresponding denotational model in the category of free R-modules has differential structure, although this is not reflected in the syntax, leading to a failure of full abstraction. By moving to the solos calculus – with its close connection to differential structure – this paper develops an analogous, but fully abstract interpretation for a broader class of programs. (We show that there is a sound interpretation of R-weighted nondeterministic λ-terms in the solos calculus over R.)

Our semantics of a concurrent process calculus with mobility which represents terms as sums of their independent reduction paths is adumbrated by the work of Beffara [2], which captures directly this notion of independent path in the (finitary) πI-calculus, and uses it to give a trace semantics which is observed to possess many of the algebraic properties which we use to define our model (e.g. the processes over a given set of free names form a semiring). Our compositional construction of a semantics of this kind is therefore complementary, and opens up the question of defining a formal relationship with the trace semantics. Similarly, the representation of replication as a formal power series is foreshadowed by Boreale and Gadducci [5].

1.2 Contribution

In this paper we develop a new semantic account of the solos calculus, by weighting terms and reduction paths with values from a complete semiring R. Our main results are operational and denotational semantics for a "unidirectional" fragment of the calculus in which each closed term is interpreted as the sum in R of the weights of its reduction paths. We show that the unidirectional fragment is sufficiently expressive to capture reduction behaviour in the full calculus, and to evaluate sums-of-paths for an R-weighted non-deterministic λ-calculus.

Our denotational semantics interprets terms in the category of free R-modules and their homomorphisms, which correspond simply to matrices with entries in R. We formalize the differential structure required to interpret R-weighted unidirectional terms – a reflexive differential bialgebra – and use it to establish soundness of the model.

2 Preliminaries: Complete Semirings

Both operational and denotational semantics will use notions of complete monoid, semiring and semimodule, which we define here. A *complete monoid* [12] is a commutative monoid with infinite sums – a pair (S, Σ) of a set S with an operation Σ taking indexed[1] sets over S to elements of S, satisfying the following axioms:

- For any indexed family $\{a_i\}_{i \in I}$, and partitioning function $f : I \to J$, $\Sigma_{i \in I} a_i = \Sigma_{j \in J}(\Sigma_{i \in f^{-1}(j)} a_i)$.
- $\Sigma_{i \in \{j\}} a_i = a_j$.

A complete (commutative[2]) semiring R is a tuple $(|R|, \Sigma, \cdot, 1)$ such that $(|R|, \Sigma)$ is a complete monoid and $(|R|, \cdot, 1)$ is a commutative monoid which distributes over Σ – i.e. $a \cdot \Sigma_{i \in I} b_i = \Sigma_{i \in I} a \cdot b_i$.

If R is a complete semiring, then $(R, +, 0, \cdot, 1)$ is a commutative semiring in the usual sense (where 0 is the sum of the empty family, and $a_1 + a_2 = \Sigma_{i \in \{1,2\}} a_i$). R is *idempotent* if $a_i = b$ for all $i \in I$ implies $\Sigma_{i \in I} a_i = b$.

If the sub-semiring of R, of elements generated from the unit 1 is a semifield, then we may define an *exponential* function on R (a homomorphism from its additive to its multiplicative structure) as follows:

▶ **Definition 1.** Let R be a complete semiring. For each natural number n, let n_R denote the sum $\Sigma_{1 \leq i \leq n} 1$. If n_R has a multiplicative inverse $\frac{1}{n_R}$ for each $n > 0$, we may define the *Taylor exponential* $! : (R, +, 0) \to (R, \cdot, 1)$ as the sum of the formal power series $!a = \Sigma_{n \geq 0} \frac{1}{!n}.a^n$.

Note that we may define the Taylor exponential on any idempotent complete semiring (as $n_R = \frac{1}{n_R} = 1$ for all n).

2.1 Semiring-Weighted Networks

We shall represent concurrent systems as (possibly infinite) *matrices* over a complete semiring R (an $A \times B$ matrix over R is a function from $A \times B$ to R). To motivate the rest of the paper, we note that such matrices provide a general setting for defining and studying shortest-path and related problems [21], which are a classical application of semirings in quantitative analysis of static systems. An $A \times A$ matrix $G \in R^{A \times A}$ corresponds to a network, or weighted digraph on the set of nodes A, with $G(a, a')$ being the weight of the edge from a to a'. For any *path* (sequence of length at least 2) in A^* we may compute a weight by multiplication in R – i.e. $w(a_0, \ldots, a_{n+1}) = G(a_0, a_1) \cdot \ldots \cdot G(a_n, a_{n+1})$ – and so define the sum of weights of all paths between a and a': $\Sigma_G(a, a') = \Sigma_{s \in A^*} w(asa')$. The significance of this value depends on the choice of R. For example:

- If R is the *Boolean semiring* $\mathbb{B} = (\{\top, \bot\}, \bigvee, \wedge, \top)$ (i.e. G represents an unlabelled digraph) then $\Sigma_G(a, a') = \top$ iff there is a path from a to a'.
- In general, if R is a *lattice*, (e.g. $G(a, a')$ is the security level of information that can pass from a to a') then $\Sigma_G(a', a)$ is the least upper bound of all information that may flow from a to a'
- If R is the *tropical semiring* $\mathbb{T}^\infty = (\mathbb{R}_+ \cup \{\infty\}, \bigwedge, +, 0)$ (i.e. $G(a, a')$ is the length or cost of travelling from a to a') then $w(s)$ represents the length of the path s and $\Sigma_G(a, a')$ is the length of the shortest path from a to a'.
- If R is the *probability semiring* $(\mathbb{R}_+ \cup \{\infty\}, \Sigma, \times, 1)$, and the sum of weights entering (or leaving) each node is less than 1 (i.e G is a *stochastic matrix*) then $\Sigma_G(a, a')$ is the probability of reaching a' from a.

[1] Our semantics restrict straightforwardly to countably complete monoids/semirings.
[2] Only commutative monoids and semirings will be considered throughout.

3 A Calculus of Solos With Resources

In this section we describe a "resource sensitive" version of the *solos calculus* [17]. This is the fragment of the fusion calculus [23] without prefixing – that is, we have primitives (solos) which can emit or receive channel names, but no primitives for expressing sequentialization (although these may be expressed). We include *weights* from a commutative monoid, which can quantify the resources used (the monoid operation shows how to combine weights across parallel composition).

We work in the *dyadic*[3] solos calculus, omitting matching of channel names, but including *explicit fusions* of names, which simplify presentation of the semantics (these are studied in detail by Wischik and Gardner [25]). Let $M = (|M|, \cdot, 1)$ be a commmutative monoid. Terms of the solos calculus over M are formed according to the grammar:

$$p, q ::= \underline{a} \mid x(y,z) \mid \overline{x}(y,z) \mid x = y \mid p|q \mid p + q \mid !p \mid \nu x.p$$

where variables x, y, z represent communication channel names, and:

- Each constant $\underline{a}$ is a *weight* representing the value a in $|M|$.
- $x(y,z)$ and $\overline{x}(y,z)$ are input and output *solos* – representing the receiving and sending of the pair of names y and z on the channel x, respectively.
- $x = y$ is an *explicit fusion* asserting the identity of the names x and y.
- $p|q$ is *parallel composition*, with unit $\underline{1}$.
- $\nu x.p$ is *hiding*, binding the name x in p.
- $!p$ is the *exponential* of p, offering arbitrarily many copies of p in parallel.
- $p + q$ is an (external) *choice* of the processes p and q.

3.1 Reduction Semantics

Our operational semantics for the solos calculus is non-standard – the primary justification for this is that it reflects an elegant denotational, *algebraic* model. The close correspondence between the calculus and *differential nets* [10, 8] suggests that a term of the solos calculus can represent a collection of resources, so that reduction determines whether these resources are successfully consumed or not (as in the differential λ-calculus [9]). In practical terms, this means that our reduction rules for the solos calculus are linear, rather than affine – a resource which cannot be consumed (e.g. $\nu x.x(y,z)$) is not equivalent to the unit for parallel composition – and the sum is an external choice corresponding to the sum in a semiring. Note that the latter may be macro-expressed (e.g. as $p + q = \nu x.\overline{x}(-,-)|!(x(-,-)|p)|!(x(-,-)|q)$ where x is not free in p or q). Linearity allows us to be more precise about how resources are used, but we can express the affine behaviour of the original solos calculus (e.g. by representing an affine solo as a choice $x(y,z) + 1$), as for the π-calculus [2].

We work up to *structural congruence*, which is the smallest congruence on terms containing α-equivalence with respect to bound variables and the following axioms:

$$\begin{array}{llll} p|(q|r) \equiv (p|q)|r & p|q \equiv q|p & p|\underline{1} \equiv p & \underline{a}|\underline{b} \equiv \underline{a \cdot b} \\ \nu x.\nu y.p \equiv \nu y.\nu x.p & \nu x.\underline{1} \equiv \underline{1} & (\nu x.p)|q \equiv \nu x.(p|q)\ (x \notin FV(p)) & \end{array}$$

In other words, terms are identified up to associativity and commutativity of parallel composition (which acts as multiplication in M on weights) and scope extrusion of variables. The basic reduction rules are as follows:

[3] This is a minimal, expressive version of the calculus – Laneve and Victor [17] show that monadic solos cannot express polyadic solos.

$$\overline{x}(u,v)|x(y,z) \rightarrow u = y|z = v$$

$$!p \rightarrow \underline{1} \qquad \nu x.x = y|p \rightarrow p[y/x] \qquad p + q \rightarrow p$$

$$!p \rightarrow p|!p \qquad \nu y.x = y|p \rightarrow p[x/y] \qquad p + q \rightarrow q$$

In other words, communicating solos reduce by fusing their arguments, $!p$ may replicate or discard p,[4] explicit fusion of a bound variable reduces by substitution with the variable to which it is fused and non-deterministic choice reduces to one of its branches.

These reductions may be applied inside hiding and parallel composition, to terms identified up to structural congruence. We define the compatible reduction $\longrightarrow$ to be the least relation on terms such that:

$$\frac{p \rightarrow q}{p \longrightarrow q} \qquad \frac{p \equiv p' \quad p \longrightarrow q \quad q \equiv q'}{p' \longrightarrow q'} \qquad \frac{p \longrightarrow q}{\nu x.p \longrightarrow \nu x.q} \qquad \frac{p \longrightarrow q}{p|r \longrightarrow q|r}$$

Every terminating reduction path ends either in a weight $\underline{a}$ or an irreducible term p which contains solos which cannot communicate – i.e. resources which cannot be consumed or demands for resources which cannot be satisfied (cf. the differential λ-calculus [9]). In the former case, we say that a is a weight for the path, in the latter, that p is a *failure* (written $p \not\downarrow$).

For instance, we might represent a finite directed graph with weights from M as a term of the solos calculus, such that reduction paths correspond to paths through G. Assuming for the sake of simplicity that G is acyclic, let $\widetilde{G} = |_{i,j \in N}!([x_i \mapsto x_j]|a_{ij})$,

▶ **Proposition 2.** $\nu x_1 \ldots x_n.\overline{x_1}(-,-)|\widetilde{G}|x_k(-,-) \downarrow a$ *if and only if there is a path from node* 1 *to node* k *of weight* a.

More importantly, using the solos calculus allows us to describe networks which do not have a fixed topology – for example by passing names *through* the network to create new (weighted) connections.

3.2 The Unidirectional Solos Calculus

Our aim is to give a semantics of the solos calculus which accounts for all reduction paths, by summing their weights in a complete semiring. In general, to compute a sum of path weights for a term it is necessary to take account of the *multiplicity* of distinct paths to the same value, where paths are distinguished according to the different choices made during reduction, but not the order in which they are made. To make this notion of "sum of independent paths" precise, we restrict attention to an expressive fragment of the solos calculus, closer to differential nets, for which we are able to give operational and denotational semantics which give a consistent interpretation of path sum. This "unidirectional" fragment is defined by a derivation system which separates input and output capabilities and enforces constraints on mobility of names related to those in the private π-calculus [24].

In a unidirectional term, the solo $\overline{x}(y,z)$ is assumed to send (on x) the capability to receive on y and send on z. Accordingly, we say that x and z occur as output names, and y as an input name in $\overline{x}(y,z)$. Dually $x(y,z)$ receives on x the capability to send on y, and receive on z – i.e. x and z occur as input names and y as an output name. The fusion $x = y$ joins an input name (x) to an output name (y).

We shall say that an occurrence of a variable is *mobile* if it is the argument to an input or output solo, or in an explicit fusion – i.e. y, z occur as mobile names in both $x(y,z)$,

[4] Note that $!p$ is *not* structurally congruent to $p|!p$, reflecting the linear nature of our rules.

$$\frac{}{x,\underline{z}\vdash x(y,z);\underline{y}} \qquad \frac{}{\underline{y}\vdash \overline{x}(y,z);x,\underline{z}} \qquad \frac{}{\underline{x}\vdash x=y;\underline{y}}$$

$$\frac{\Gamma\vdash p;\Delta}{\Gamma-\{x\}\vdash \nu x.p;\Delta-\{x\}} \qquad \frac{\Gamma\vdash p;\Delta \qquad \Gamma\vdash q;\Delta}{\Gamma\vdash p+q;\Delta} \qquad \frac{}{\_\vdash a;\_}\, a\in |M|$$

$$\frac{\Gamma\vdash p;\Delta \qquad \Gamma'\vdash q;\Delta'}{\Gamma\cup\Gamma'\vdash p|q;\Delta\cup\Delta'}\,\Gamma\lightning\Gamma',\Delta\lightning\Delta' \qquad \frac{\Gamma\vdash p;\Delta}{\Gamma\vdash !p;\Delta}\,\underline{\Gamma}=\underline{\Delta}=\varnothing$$

Figure 1 Derivation Rules for Unidirectional Solos.

$\overline{x}(y,z)$ (and $y = z$), whereas x does not. A unidirectional context Γ is a set of names with a specified subset $\underline{\Gamma} \subseteq \Gamma$ of mobile names. Figure 1 gives derivation rules for unidirectional terms-in-context of the form $\Gamma \vdash p; \Delta$, where Γ and Δ are unidirectional contexts of input and output names occurring in p. These rules may be seen as enforcing a simple *linear* typing discipline on terms of the solos calculus: mobile names must be used linearly, whereas static names may be used freely, with respect to the input and output modalities separately.

We write $\Gamma \lightning \Gamma'$ if $\underline{\Gamma} \cap \Gamma' = \varnothing$ and $\underline{\Gamma'} \cap \Gamma = \varnothing$. Sharing of mobile names is constrained by requiring that in the parallel composition $p|q$ the input and output contexts of p and q must be non-interfering in this sense. Similarly, the exponential $!p$ may contain no (free) mobile names. A name is static in $\Gamma \vdash p; \Delta$ if it does not occur in $\underline{\Gamma} \cup \underline{\Delta}$.

▶ **Proposition 3.** *If* $\Gamma \vdash p; \Delta$ *and* $p \longrightarrow q$ *then there exist* Γ', Δ' *such that* $\Gamma' \vdash p; \Delta'$.

Proof. This is evident for the basic reductions of communicating solos, choice and replication. The key case is reduction of explicit fusion by substitution. We show that if $\Gamma, y \vdash p; \Delta, x$, where $x \notin \Gamma$ and $y \notin \Delta$, then $\Gamma, y \vdash p[y/x]; \Delta, y$ by induction on p. Hence if $\Gamma, y \vdash \nu x.x = y|p; \Delta, \underline{y}$, so that $\Gamma, y \vdash p; \Delta, x$ then $\Gamma, y \vdash p[y/x]; \Delta, y$ as required. (Note, however, that in this case, y is now output-static.)

It is straightforward to check that unidirectionality is preserved under structural congruence – i.e. if $p \equiv p'$ and $\Gamma \vdash p; \Delta$ then $\Gamma \vdash p'; \Delta$. So subject reduction extends to the compatible reduction relation. ◀

We now assume that our monoid of resources is the multiplication in a complete semiring R with a Taylor exponential.

3.3 Expressiveness of Unidirectional Terms

Passing of bound names, as in the private π-calculus [24], is naturally expressed in the unidirectional fragment: we write $x(\underline{y}, \underline{z})p$ and $\overline{x}(\underline{y}, \underline{z})p$ for $\nu y.\nu z.x(y, z)|p$ and $\nu y.\nu z.\overline{x}(y, z)|p$ respectively. These are essentially bound input and output operations for the "synchronous π-calculus" [3].

To show the expressiveness of the unidirectional solos calculus , we may define unidirectional terms which correspond to bidirectional solos – i.e. they can pass the capability to send and receive on static names. Hence we may give a translation of the full solo calculus into the unidirectional fragment which is sound with respect to reduction.

Using the *forwarder* $[x \mapsto y] =_{df} \nu u.\nu v.x(u, v)|\overline{y}(u, v)$, we may define macros for explicit fusions of static variables (the *equators* of [14]) – let $\widehat{x = y} =_{df} ![x \mapsto y]|![y \mapsto x]$ – and for monadic bidirectional solos $\widehat{x(u)}$ (input) and $\widehat{\overline{x}(u)}$ (output) which pass both input and output capabilities on the static name u:

$\widehat{x(u)} =_{df} x(\underline{v}, \underline{w})![v \mapsto u]|![u \mapsto w]$ $\qquad$ $\widehat{\overline{x}(u)} =_{df} \overline{x}(\underline{v}, \underline{w})![u \mapsto v]|![w \mapsto u]$ We define a unidirectional term representing the dyadic bidirectional solo $x(y, z)$ by passing private names v, w on x, and communicating send and receive capacity for y on v, and for z on w (as in the encoding of polyadic communication in the monadic π-calculus [24]) – i.e.

- $\widehat{x(y,z)} = x(\underline{v}, \underline{w})\widehat{v(y)}|\widehat{\overline{w}(z)}$.
- $\widehat{\overline{x}(y,z)} = \overline{x}(\underline{v}, \underline{w})\widehat{\overline{v}(y)}|\widehat{w(z)}$.

This yields a compositional translation $\widehat{\_}$ of the (dyadic) solos calculus into its unidirectional fragment by replacing each solo and explicit fusion with the corresponding macro.

▶ **Proposition 4.** *For every term p, $p \longrightarrow^* \underline{a}$ if and only if $\widehat{p} \longrightarrow^* \underline{a}$.*

Proof Outline. We show the following by induction on reduction:

- for any unidirectional term p, $\nu x.\widehat{x = y}|p \longrightarrow^* \underline{a}$ if and only if $\nu x.p[x/y] \longrightarrow^* \underline{a}$. (We prove from left to right by showing that if $\nu x.([x \mapsto y]|[y \mapsto x])^n|\widehat{x = y}|p \longrightarrow^* \underline{a}$ for some n then $\nu x.p[x/y] \longrightarrow^* \underline{a}$.)
- $\nu x.\widehat{v(y)}|\widehat{\overline{v}(y')}|\widehat{p} \longrightarrow^* \underline{a}$ if and only if $\widehat{y = y'}|\widehat{p} \longrightarrow^* \underline{a}$.
- $\widehat{x(y,z)}|\widehat{\overline{x}(y',z')}|\widehat{p} \longrightarrow^* \underline{a}$ if and only if $\widehat{y = y'}|\widehat{z' = z}|\widehat{p} \longrightarrow^* \underline{a}$.

In other words, reduction of $\widehat{p}$ precisely tracks reduction of p, and hence $p \longrightarrow^* \underline{a}$ if and only if $\widehat{p} \longrightarrow^* \underline{a}$. ◀

3.4 The Quantitative λ-Calculus

As a demonstration of the expressiveness of quantitative unidirectional solos, we adapt Milner's translation of the λ-calculus into the π-calculus [19]. [16] introduced an applied λ-calculus (PCF) with non-deterministic choice and scalar multiplication by weights in a continuous semiring R, describing an operational semantics evaluating programs to elements of R and a corresponding denotational semantics in the category of weighted relations (i.e. matrices) over R (which will also furnish models of the solos calculus over R). We show that these results may be recast as an interpretation of the untyped λ-calculus with choice in the unidirectional solo calculus (where they may be extended to any *complete* semiring with a Taylor exponential).

For any complete semiring R, let Λ^+_{R} be the (lazy) λ-calculus extended with a binary choice operator $+$, and weighting with values from R – i.e. terms are given by the grammar:

$$M, N ::= x \mid \lambda x.M \mid M\ N \mid M + N \mid \underline{a}(M)$$

where a ranges over elements of R.

We may interpret Λ^+_{R} by translation into the unidirectional solos calculus over R . A term M of Λ^+_{R} over the free variables $x_1, \ldots, x_n$ is interpreted as a R-term $x_1, \ldots, x_n \vdash (\![M]\!)(u); u$ with free static input names $x_1, \ldots, x_n$ and output name u.

- $(\![x]\!)(u) = x(u, -)$
- $(\![\lambda x.M]\!)(u) =!\overline{u}(\underline{v}, \underline{x})(\![M]\!)(v)$
- $(\![M\ N]\!)(u) = \nu v.v(u, \underline{w})|(\![M]\!)(v)|!\overline{w}(\underline{y}, -)(\![N]\!)(y)$.
- $(\![M + N]\!)(u) = (\![M]\!)(u) + (\![N]\!)(u)$.
- $(\![\underline{a}(M)]\!)(u) = \underline{a}|(\![M]\!)(u)$.

We show that this translation is sound with respect to an operational semantics of Λ^+_{R} which evaluates each term to the sum of its reduction-path weights (based on [16]).

4 Denotational Semantics

We now describe, for each complete semiring R, a fully abstract interpretation of the unidirectional solos calculus in the symmetric monoidal category $\mathsf{Mat}_{\mathrm{R}}$ of sets and matrices over R. The objects of $\mathsf{Mat}_{\mathrm{R}}$ are sets, and morphisms from X to Y are $X \times Y$ matrices (a.k.a. "R-weighted relations") over R, composed by matrix multiplication – given $f : X \to Y$ and $g : Y \to Z$, $(f;g)(x,z) = \Sigma_{y \in Y} f(x,y) \cdot g(y,z)$.

The tensor product $X \otimes Y$ is the cartesian product of X and Y as sets (with unit I being the singleton set), sending $f : X \to X'$ and $g : Y \to Y'$ to the matrix $f \otimes g$ with $(f \otimes g)(x,y,x',y') = f(x,x') \cdot g(y,y')$. $\mathsf{Mat}_{\mathrm{R}}$ is *compact closed*: every object Y is dual to itself – i.e. there are evident natural isomorphisms $\mathsf{Mat}_{\mathrm{R}}(X \otimes Y, Z) \cong \mathsf{Mat}_{\mathrm{R}}(X, Y \otimes Z)$.

For each X, Y, the $X \times Y$ matrices over R form a *R-module* – that is, a complete monoid with an operation of scalr multiplication by elements of R, which satisfies:

$$(\Sigma_{i \in I} a_i).u = \Sigma_{i \in I}(a_i.u) \qquad a.\Sigma_{i \in I} u_i = \Sigma_{i \in I} a.u_i \qquad (a \cdot b).u = a.(b.u) \qquad (1.u) = u$$

▶ **Proposition 5.** Mat_{R} *is enriched over the category of R-modules and their homomorphisms.*[5]

Proof. Concretely, this means that there are indexed sum and scalar multiplication operations on each hom-set (i.e. pointwise addition and multiplication of matrix entries) which satisfy the axioms for a R-module and distribute over composition and the tensor product – i.e.

- $(\Sigma_{i \in I} f_i); g = \Sigma_{i \in I} f_i; g$, $f; \Sigma_{j \in J} g_j = \Sigma_{j \in J}(f; g_j)$ and $(a.f); g = a.(f;g) = f;(a.g)$.
- $(\Sigma_{i \in I} f_i) \otimes g = \Sigma_{i \in I}(f_i \otimes g)$ and $(a.f \otimes g) = a.(f \otimes g)$.

By elementary linear algebra, the embedding of $\mathsf{Mat}_{\mathrm{R}}$ into the category of R-modules which sends each set X to the free R-module R^X, and each $X \times Y$ matrix to the corresponding linear function from R^X to R^Y is fully faithful. ◀

4.1 Differential Structure

We interpret sharing of channels (contraction and weakening) using simple *differential structure* in our category of matrices. The properties we require may be presented as follows:

▶ **Definition 6.** A *differential bialgebra* on a pair of objects (A, B) in a commutative-monoid-enriched symmetric monoidal category is given by morphisms $(\mu : B \otimes B \to B, \eta : I \to B, \delta : B \to B \otimes B, \epsilon : B \to I, \zeta : A \to B, \xi : B \to A)$ such that $(B, \mu, \eta, \delta, \eta)$ is a *commutative bialgebra*, and the following equations hold:

(i) $\zeta; \xi : A \to A = \mathsf{id}_A$

(ii) $\eta; \xi : I \to A = 0$ and $\zeta; \epsilon : A \to I = 0$

(iii) $\mu; \xi : B \otimes B \to A = (\epsilon \otimes \xi) + (\xi \otimes \epsilon)$ and $\zeta; \delta : A \to B \otimes B = (\eta \otimes \zeta) + (\zeta \otimes \eta)$

These equations are implicit in the definition of differential nets [10], and included explicitly (alongside further structure) in the notion of a *model of the differential calculus* [4], which is proven equivalent to a *differential category with a storage modality* – in any such category there is a comonad $! : A \to A$ with a differential bialgebra on $(A, !A)$ for each A. In $\mathsf{Mat}_{\mathrm{R}}$ we may define a differential bialgebra on $(A, \mathcal{M}_*(A))$ for any set A, where $\mathcal{M}_*(A)$ is the set of *finite multisets* over A, by following the construction of the *cofree commutative comonoid* on [18, 16] (essentially, a generalization of the finite multiset exponential on the relational model of linear logic). Specifically, we may define the matrices:

[5] The category of R-modules and their homomorphisms is symmetric monoidal closed, with construction of the tensor product of R-modules following [13] – see e.g. [1] for extension with infinite sums).

- $\eta(*, X) = \epsilon(X, *) = 1$ if $X = \{\}$, 0 otherwise.
- $\mu((X, Y), Z) = \delta(X, (Y, Z)) = 1$ if $X = Y \uplus Z$, 0 otherwise.
- $\xi(X, x) = \zeta(x, X) = 1$ if $X = \{x\}$, 0 otherwise.

We interpret (the "type" of) channels as a differential bialgebra which satisfies a basic recursive equation: on a channel we may send (finitely but unboundedly many) pairs of an output and input name.

▶ **Definition 7.** A *reflexive* differential bialgebra is an object B (in a commutative monoid-enriched SMC) with a *dual* B^* and a differential bialgebra on $(B \otimes B^*, B)$.

To define a reflexive differential bialgebra in $\mathsf{Mat}_{\mathrm{R}}$, we take the least fixed point of the $\subseteq$-continuous operation sending the set X to the set $\mathcal{M}_*(X \times X)$ of finite multisets of pairs of elements of X. Let B be the (countable) set $\bigcup_{i \in \omega} B_i$, where $B_0 = \varnothing$, and $B_{i+1} = \mathcal{M}_*(B_i \times B_i)$, so that $B = \mathcal{M}_*(B \times B)$. Since B is self-dual, and the tensor product in $\mathsf{Mat}_{\mathrm{R}}$ is cartesian product of sets, $B = \mathcal{M}_*(B \otimes B^*)$.

4.2 Denotational Interpretation

Let R be a complete semiring with a Taylor exponential, and let B be a reflexive differential bialgebra in a R-module-enriched symmetric monoidal category $\mathcal{C}$, yielding a commutative bialgebra $(B^{\otimes n}, \mu_n, \eta_n, \delta_n, \epsilon_n)$ for each n.

For each m, n, $\mathcal{C}(B^{\otimes m}, B^{\otimes n})$ is a complete semiring – we may define a product operation on $\mathcal{C}(B^{\otimes m}, B^{\otimes n})$: $f \cdot g = \delta_m; (f \otimes g); \mu_n$, with neutral element $1 = \epsilon_m; \eta_n : B^{\otimes m} \to B^{\otimes n}$. This is associative and commutative, and distributes over the indexed sum on $\mathcal{C}(B^{\otimes m}, B^{\otimes n})$, yielding a complete semiring $\mathcal{C}_{\mathrm{R}}(B^{\otimes m}, B^{\otimes n})$, with a homomorphism of semirings from R into $\mathcal{C}_{\mathrm{R}}(B^{\otimes m}, B^{\otimes n})$ sending $a \in R$ to $a.1$. Hence, in particular, $\mathcal{C}_{\mathrm{R}}(B^{\otimes m}, B^{\otimes n})$ has a Taylor exponential. The full subcategory of $\mathcal{C}$ generated from I, B'B^* is compact closed, and therefore has a canonical *trace operator* [15], with which we interpret hiding.

Ordering input and output contexts, we interpret terms-in-context $x_1, \ldots, x_m \vdash p; x_1, \ldots, x_n$ in $\mathcal{C}_{\mathrm{R}}(B^{\otimes m}, B^{\otimes n})$, as follows:

- *Constants* denote scalar multiples of the unit: $[\![\_ \vdash \underline{a}; \_]\!] = k.\mathsf{id}_I$.
- *Solos* denote ξ and ζ: $[\![x, z \vdash x(y, z); y]\!] = \Lambda^{-1}(\xi)$, $[\![y \vdash \overline{x}(y, z); x, z]\!] = \Lambda(\zeta)$.
- *Explicit Fusions* denote the identity: $[\![x \vdash x = y; y]\!] = \mathsf{id}_B$.
- *Hiding* denotes the trace operation: $[\![\Gamma \vdash \nu x.p; \Delta]\!] = \mathrm{tr}([\![\Gamma, x \vdash p; \Delta, x]\!])$.
- *Composition* denotes the product: $[\![\Gamma \vdash p|q; \Delta]\!] = [\![\Gamma \vdash p; \Delta]\!] \cdot [\![\Gamma \vdash q; \Delta]\!]$.
- *Choice* denotes the sum: $[\![\Gamma \vdash p + q; \Delta]\!] = [\![\Gamma \vdash p; \Delta]\!] + [\![\Gamma \vdash q; \Delta]\!]$.
- *Replication* denotes the Taylor exponential: $[\![\Gamma \vdash !p; \Delta]\!] = ![\![\Gamma \vdash p; \Delta]\!]$.

Permutation of contexts corresponds to composition with the corresponding isomorphisms on $B^{\otimes m}$ and $B^{\otimes n}$, and weakening of contexts to composition with $B^{\otimes m} \otimes \epsilon : B^{\otimes m+1} \to B^{\otimes m}$ and $B^{\otimes n} \otimes \eta : B^{\otimes n} \to B^{\otimes n+1}$

It may be noted that this interpretation does not mention unidirectionality. However, it plays a critical role in the proof of its *soundness* in the following section.

5 Sum-of-Paths Evaluation

We now aim to show that our denotational semantics for a (closed) unidirectional term p over the semiring R corresponds to an operational semantics which computes a sum in R of the residues of the reduction paths of p.

$$\frac{}{\underline{a}\Downarrow_{\mathrm{R}}a} \qquad \frac{p\Downarrow_{\mathrm{R}}a \quad p\equiv q}{q\Downarrow_{\mathrm{R}}a} \qquad \frac{p\not\Downarrow}{p\Downarrow_{\mathrm{R}}0}$$

$$\frac{p\Downarrow_{\mathrm{R}}a}{\nu x.p\Downarrow_{\mathrm{R}}a} \qquad \frac{p|r\Downarrow_{\mathrm{R}}a \qquad q|r\Downarrow_{\mathrm{R}}b}{(p+q)|r\Downarrow_{\mathrm{R}}a+b} \qquad \frac{p^n|q\Downarrow_{\mathrm{R}}a_n}{!p|q\Downarrow_{\mathrm{R}}\Sigma_{n\geq 0}\frac{a_n}{n!}}$$

$$\frac{p\Downarrow_{\mathrm{R}}a}{x{=}x|p\Downarrow_{\mathrm{R}}\infty.a} \qquad \frac{x(y_1,z_1)|\ldots|u{=}y_i|z_i{=}v|\ldots|x(y_n,z_n)|p\Downarrow_{\mathrm{R}}a_i}{\overline{x}(u,v)|x(y_1,z_1)|\ldots|x(y_n,z_n)|p\Downarrow_{\mathrm{R}}\Sigma_{i\leq n}a_i}\; x\notin FV^-(p) \qquad \frac{p[y/x]\Downarrow_{\mathrm{R}}a}{x{=}y|p\Downarrow_{\mathrm{R}}a} x\not\equiv y$$

Figure 2 Evaluation Rules for Unidirectional Terms.

If R is idempotent, then we may define this to be $\bigvee\{a \mid p \downarrow a\}$, but in the general case, we require a notion of (syntax) *independent reduction path*. On the one hand, it is necessary to take account of the *multiplicity* of distinct paths to the same value. For example, there should be two reduction paths from $\underline{a}+\underline{a}$ to $\underline{a}$. On the other hand some distinct reduction paths in the rewriting system are different syntactic representations of the same events and so do not represent independent paths in the required sense. For example, there is a single path from $\underline{a}+\underline{b}|\underline{c}+\underline{d}$ to $\underline{a}|\underline{c}$. Moreover, in calculi with mobility such as the solos calculus the order in which reduction choices are made changes the communications available – as in the term $\nu x\nu y.\nu z.x(y,z)|\overline{x}(z,y)|y(x,z)|\overline{y}(z,x)|z(x,y)|\overline{z}(y,x)$, for example.

Beffara makes this notion of independent reduction path *explicit* in giving a trace semantics of the πI-calculus [2], which is quotiented by an equivalence relation between reduction paths. However, if p is a term in our unidirectional solos calculus we can always find a channel on which *all possible interactions are simultaneously available*, and so it remains implicit in the operational semantics given here.

▶ Remark. Unidirectionality, and our denotational semantics, provide a perspective on the notion of *acyclicity* introduced by Ehrhard and Laurent [7]. Cycles arise when a channel name becomes fused to itself – a term is acyclic if it never attempts to fuse a channel to itself in this way.

In our semantics, cyclic terms denote infinite sums of paths. For example, consider the solo term $\nu x.x = x$, which denotes the composition of the unit and counit $\nu_B;\epsilon_B : I \Rightarrow I$. Letting $\infty = \Sigma_{i\in\mathbb{N}}1$ (note that if R is idempotent, then $\infty = 1$) then $\Sigma_{b\in B}\mathsf{id}_I = \infty.\mathsf{id}_I$, whereas (e.g.) $\nu x.\nu y.x = y$ denotes id_I. Semantically, this corresponds to identifying an explicit fusion such as $x = x$ with the forwarder $![x \mapsto x]$, which reduces to $\nu yz.x(y,z)|\overline{x}(y,z)|![x \mapsto x] \longrightarrow \nu yz.y = y|z = z|![x \mapsto x]$, generating infinitely many reduction paths. Note that this does not arise for forwarders and equators in the π-calculus, which must receive an input before they can send an output – more generally (as noted by Ehrhard and Laurent), terms of the π-calculus and λ-calculus can be represented as acyclic solos.

5.1 Evaluation Semantics

The rules in Figure 2 define a relation $\Downarrow_{\mathrm{R}}$ between unidirectional R-terms and values in R, such that if $p \Downarrow_{\mathrm{R}} a$ then a is the sum in R of the weights of the reduction paths of p. We write p^n for the composition of n copies of p – i.e. $p^0 = 1$, $p^{n+1} = p|p^n$, and $FV^-(p)$ for the set of input names of p.

▶ **Proposition 8.** *For any term p, there exists a such that $p \Downarrow_R a$.*

Proof. By induction on the (well-founded [6]) multiset ordering on the measure $\ell(p)$, defined as follows:

$$\begin{array}{cc} \ell(\underline{a}) = \ell(x = y) = \varnothing & \ell(x(y,z)) = \ell(\overline{x}(y,z)) = \{\{\}\} \\ \ell(\nu x.p) = \ell(p) & \ell(p|q) = \ell(p) \cup \ell(q) \\ \ell(p+q) = \ell(p) \cup \ell(q) \cup \{\{\}\} & \ell(!p) = \{\ell(p)\} \end{array}$$

Note that ℓ is invariant with respect to structural congruence, and if $q \Downarrow_{\mathrm{R}} b$ is a premise for a rule (other than structural congruence) with conclusion $p \Downarrow_{\mathrm{R}} a$, then $\ell(q) \ll \ell(p)$. If p contains occurrences of $!$, $=$ or $+$, then one of the corresponding rules is applicable. Otherwise p is equivalent to $\nu x_1 \ldots \nu x_m.p'$, where p' is a parallel composition of solos and constants. If p' consists only of constants, then $p \equiv \underline{a}$ for some a. If p' contains a pair of complementary solos $x_i(x_j, x_k)$ and $\overline{x_i}(x_{j'}, x_{k'})$ then by unidirectionality x_i cannot occur as a mobile name in p', and so the communication rule applies. Otherwise p is a failure. ◀

Moreover, it is a consequence of the soundness of the denotational model (Proposition 17) that the result of evaluation is unique and therefore $\Downarrow_{\mathrm{R}}$ defines a function from closed R-terms to elements of R. For idempotent complete semirings, this agrees with small-step reduction in the following sense.

▶ **Proposition 9.** *If R is idempotent, $p \Downarrow_{R} \bigvee\{k \mid p \longrightarrow^{*} \underline{a}\}$*

Proof. By induction over the nested multiset ordering on $\ell(p)$. ◀

Each complete semiring R induces a notion of contextual equivalence (R-equivalence) on unidirectional solo terms, by testing closed terms with $\Downarrow_{\mathrm{R}}$. Say that a context $C[\_]$ is a closing context for Γ, Δ if for any unidirectional term $\Gamma \vdash p; \Delta$, $C[p]$ is a closed unidirectional term.

▶ **Definition 10.** Given terms $\Gamma \vdash p, q; \Delta$, $p \sim_{\mathrm{R}}^{\Gamma,\Delta} q$ if for all closing contexts $C[\_]$ for Γ, Δ, $C[p] \Downarrow_{\mathrm{R}} a$ if and only if $C[q] \Downarrow_{\mathrm{R}} a$.[6]

The properties of $\sim_{\mathrm{R}}$ depend on R, but in general it is neither coarser nor finer than the bisimulation equivalence for the solos calculus [17]. We give some illustrative examples of equivalences and inequivalences (for non-trivial R), which follow from full abstraction of the denotational semantics. We leave the input and output contexts implicit.

- *Units* $\underline{1} \not\sim_{\mathrm{R}} \underline{0}$ – as noted, our interpretation of solo terms is not *affine*.
  $p|\underline{0} \sim_{\mathrm{R}} \underline{0}$ – $\underline{0}$ is an absorbing element for parallel composition.
- *Choice* $p|(q+r) \sim_{\mathrm{R}} p|q + p|r$ (distributivity).
  $p + p \sim_{\mathrm{R}} p$ if and only if the finite sum in R is idempotent .
- *Exponential* $!(p+q) \sim_{\mathrm{R}} !p|!q$ and $!0 \sim_{\mathrm{R}} 1$ – $!$ is a homomorphism from $+$ to $|$
  $!p \not\sim_{\mathrm{R}} p|!p$ in general (since $!0 \sim_{\mathrm{R}} 0|!0$ implies $0 \sim_{\mathrm{R}} !0 \sim_{\mathrm{R}} 1$).
  For idempotent R, $!p \sim_{\mathrm{R}} 1 + (p|!p)$.

5.2 Sum-of-paths for λ-terms

We illustrate our sum-of-paths interpretation of unidirectional processes by relating to an evaluation semantics of Λ_{R}^{+} terms based on that given in loc. cit. [16] – i.e. we prove *soundness* of the translation given in Section 3.4. We evaluate closed terms of Λ_{R}^{+} to elements of R using a *CEK machine* equipped with an oracle determining which branch is taken at each choice encountered in evaluation. A *state* of the machine is a triple $(C; E; K; w)$ of

[6] It will follow from our full abstraction result that any terms which are not $\sim_{\mathrm{R}}$-equivalent can be separated by pure contexts – i.e. each semiring induces a single notion of equivalence, regardless of which elements are denoted as constants.

$$\frac{}{(\lambda x.M;E;\varepsilon)\Downarrow_{\mathrm{R}}1} \qquad \frac{(M_i;E;K;w)\Downarrow_{\mathrm{R}}a}{(M_0+M_1;E;K;iw)\Downarrow_{\mathrm{R}}a}i\in\{0,1\} \qquad \frac{(M;E,(x,N);K;w)\Downarrow_{\mathrm{R}}a}{(\lambda x.M;E;N::K;w)\Downarrow_{\mathrm{R}}a}$$

$$\frac{(M;E;N::S;w)\Downarrow_{\mathrm{R}}a}{(M\ N;E;K;w)\Downarrow_{\mathrm{R}}a} \qquad \frac{(M;E;K;w)\Downarrow_{\mathrm{R}}b}{(\underline{a}(M);E;K;w)\Downarrow_{\mathrm{R}}a\cdot b} \qquad \frac{(M;E,(x,M);K;w)\Downarrow_{\mathrm{R}}a}{(x;E,(x,M);K;w)\Downarrow_{\mathrm{R}}a}$$

Figure 3 Evaluation rules for the Λ_{R}^{+} CEK machine.

$$\frac{}{(\lambda x.M;\varnothing;\varepsilon;\varepsilon)\Downarrow_{\mathrm{R}}1} \qquad \frac{(M_i;\mathsf{E};K;w)\Downarrow_{\mathrm{R}}a}{(M_0+M_1;\mathsf{E};K;iw)\Downarrow_{\mathrm{R}}a}i\in\{0,1\} \qquad \frac{(M;\mathsf{E},(x,N)^j;K;w)\Downarrow_{\mathrm{R}}a}{(\lambda x.M;\mathsf{E};N::K;w)\Downarrow_{\mathrm{R}}a}$$

$$\frac{(M;\mathsf{E};N::S;w)\Downarrow_{\mathrm{R}}a}{(M\ N;\mathsf{E};K;w)\Downarrow_{\mathrm{R}}a} \qquad \frac{(M;\mathsf{E};K;w)\Downarrow_{\mathrm{R}}b}{(\underline{a}(M);\mathsf{E};K;w)\Downarrow_{\mathrm{R}}a\cdot b} \qquad \frac{(M;\mathsf{E},(x,M)^j;K;w)\Downarrow_{\mathrm{R}}a}{(x;\mathsf{E},(x,M)^{j+1};K;w)\Downarrow_{\mathrm{R}}a}$$

Figure 4 Multiset CEK Machine for Λ_{R}^{+}.

a term C, an *environment* E (a finite set of pairs (x,M) defining a partial function from variables to terms) and a continuation K (a finite list S of terms), and an *oracle* w (an element of the set $\{0,1\}^*$ of binary words). The rules in Figure 3 define a "big-step" reduction relation from states to elements of R.

By induction on derivation, we prove that:

▶ **Lemma 11.** *If $(M;E;K,w)\Downarrow a$ and $(M;E;K;w)\Downarrow a'$ then $a=a'$.*

So for any closed term M we may define the function $\mathsf{ev}_M:\{0,1\}^*\to R$:
$\mathsf{ev}_M(w)=a$ if $(M;\varnothing;\varepsilon;w)\Downarrow_{\mathrm{R}} a$, and $\mathsf{ev}_M(w)=0$, otherwise.
Hence we may evaluate the sum of paths for M by taking the sum of $\mathsf{ev}_M(w)$ over $\{0,1\}^*$.

To prove soundness of the translation with respect to this operational semantics, we define an equivalent version of the latter in which the environment E is a *finite multiset* of variable bindings (rather than a set), such that application creates a finite (non-deterministically chosen) number of copies of the binding of x to N, invocation of x consumes a single instance and convergence requires that the environment is empty (See Figure 4.) We prove the following lemma by a straightforward induction on derivation ($\mathsf{sup}(\mathsf{E})$ is the *support* of the multiset E):

▶ **Lemma 12.** *$(M;\mathcal{E};K;w)\Downarrow_R a$ if and only if there exists a multiset E such that $\mathsf{sup}(\mathsf{E})\subseteq\mathcal{E}$ and $(M;\mathsf{E};K;w)\Downarrow_R a$.*

▶ **Proposition 13.** *For any closed term M of Λ_R^{+}, $\llparenthesis M\rrparenthesis\Downarrow_R\Sigma_{w\in\{0,1\}^*}\mathsf{ev}_M(w)$.*

Proof. By Lemma 12, it is sufficient to show $\llparenthesis M;\mathsf{E};K\rrparenthesis\Downarrow_{\mathrm{R}}\Sigma_{w\in\{0,1\}^*}\mathsf{ev}(M;\mathsf{E};K;w)$, where:

- $\mathsf{ev}(M;\mathsf{E};K;w)$ is the evaluation function for states of the multiset CEK machine – i.e. $\mathsf{ev}(M;\mathsf{E};K;w)=a$ if $(M;\mathsf{E};K;w)\Downarrow_{\mathrm{R}} a$, and $\mathsf{ev}(M,\mathsf{E},K,w)=0$, otherwise.
- $\llparenthesis M;\mathsf{E};K\rrparenthesis$ is defined by extending the translation of Λ_{R}^{+}-terms to states of the multiset CEK machine:

$$\llparenthesis M;\{(x_1,N_1)^{j_1},\ldots,(x_k,N_k)^{j_k}\};P_1:\ldots:P_l\rrparenthesis=$$
$$\llparenthesis M\rrparenthesis(v_1)|\tfrac{1}{!j_1}|(\overline{x_1}(\underline{y},-)\llparenthesis N_1\rrparenthesis(y))^{j_1}|\ldots|\tfrac{1}{!j_k}|(\overline{x_k}(\underline{y},-)\llparenthesis N_k\rrparenthesis(y))^{j_k}|v_1(v_2,\underline{w_1})|\llparenthesis P_1\rrparenthesis(w_1)|\ldots$$
$$\ldots|v_l(u,\underline{w_l})|\llparenthesis P\rrparenthesis_l(w_l).$$

This is shown by nested multiset induction on $\ell\llparenthesis M;\mathsf{E};K;w\rrparenthesis$. ◀

6 Soundness and Full Abstraction

We now prove soundness and completeness results relating the operational and denotational semantics of unidirectional terms. We first show that structurally congruent terms have the same denotation, using the symmetric monoidal structure and the properties of the trace operator

▶ **Lemma 14.** *For processes* $\Gamma \vdash; p, q; \Delta$*, if* $p \equiv q$ *then* $[\![\Gamma \vdash p; \Delta]\!]_{R} = [\![\Gamma \vdash q; \Delta]\!]_{R}$.

Soundness of the communication rule is established using the differential structure. Given a differential bialgebra (A, B) we may derive a morphism $\varepsilon : A \otimes B \to B = (\xi \otimes B); \mu$ (this corresponds to the *deriving transform* for the differential operation). For $n \geq 0$ let $\delta^n_A : B \to B^{\otimes n}$ be n-fold comultiplication derived from the comonoid structure on B, and for each $i \leq n$, let $\theta_{i,n} : A^{\otimes n} \to A^{\otimes n}$ be the permutation isomorphism swapping the first and ith copies of A. The equations for a differential bialgebra yield: $\varepsilon; \delta_{n+1}; \zeta^{\otimes n+1} : (A \otimes B) \to A^{\otimes n+1} = \Sigma_{i \leq n}(A \otimes (\delta_n; \zeta^{\otimes n})); \theta_{i,n+1}$.

▶ **Lemma 15.** *If* $x \notin FV^{-}(p)$ *then* $[\![\nu x.\overline{x}(u, v)|x(y_1, z_1)| \ldots |x(y_{n+1}, z_{n+1})|p]\!]$ $= \Sigma_{i \leq n}[\![\nu x.\overline{x}(u, v)|x(y_1, z_1)| \ldots |u = y_i|z_i = v| \ldots |x(y_{n+1}, z_{n+1})|p]\!]$.

Proof. Suppose $\Gamma \vdash p; \Delta, x$, with $x \notin \Gamma$ – so p denotes a morphism $[\![p]\!] : [\![\Gamma]\!] \to [\![\Delta]\!] \otimes B$. Then $\overline{x}(u, v)|p$ denotes (a currying of) $([\![p]\!] \otimes (B \otimes B^*)); ([\![\Delta]\!] \otimes \varepsilon) : [\![\Gamma]\!] \otimes (B \otimes B^*) \to [\![\Delta]\!] \otimes B$, and $x(y_1, z_1)| \ldots |x(y_{n+1}, z_{n+1})$ denotes (an uncurrying of) $\delta_n; \zeta^{\otimes n} : B \to (B \otimes B^*)^n$.

By dinaturality of the trace, $\nu x.\overline{x}(u, v)|x(y_1, z_1)| \ldots |x(y_{n+1}, z_{n+1})|p$ denotes the composition of these morphisms – i.e. $([\![p]\!] \otimes (B \otimes B^*)); ([\![\Delta]\!] \otimes (\varepsilon; \delta_n; \zeta^{\otimes n})) : [\![\Gamma]\!] \otimes (B \otimes B^*) \to \Delta \otimes (B \otimes B^*)^n)$.

By the differential rule above this is equal to $[\![p]\!] \otimes (\Sigma_{i \leq n}(B \otimes B^*) \otimes (\delta_n; \zeta^{\otimes n}); \theta_{i,n+1})$, and hence to $\Sigma_{i \leq n}[\![\nu x.x(y_1, z_1)| \ldots |u = y_i|z_i = v| \ldots |x(y_{n+1}, z_{n+1})|p]\!]$. ◀

Soundness of the evaluation rules for choice and replication follow directly from their definition, and for (non-acyclic) explicit fusion, from the yanking rule for the trace operator. However, the categorical structure is not sufficient to establish that every failure denotes the zero map – this requires a global argument to show that every failure corresponds to "deadlocked" matrix with a trace of zero.

▶ **Lemma 16.** *If* $p \not\Downarrow$ *then* $[\![p]\!] = 0$.

Proof. Suppose $p \not\Downarrow$. Then $p \equiv \nu x_1 \ldots x_m.q$ where q is a parallel composition of solos and constants. By definition, $[\![p]\!]$ is the trace of the $B^m \times B^m$ matrix $[\![x_1, \ldots, x_m \vdash q; x_1, \ldots, x_m]\!]$ – i.e. the sum of the entries on the diagonal of $[\![q]\!]$. Suppose (for a contradiction) that this is non-zero – then there is a non-zero entry on this diagonal – i.e. there exist $e_1, \ldots, e_m \in B$ such that $[\![q]\!](e_1, \ldots, e_m, e_1, \ldots, e_m) \neq 0$.

Choose the smallest $n \in \mathbb{N}$ such that $e_1, \ldots, e_m \in B_{n+1}$. Then (without loss of generality) there exists j such that $q \equiv x_j(y, z)|q'$ and $e_j \notin B_n$. By assumption that p is a failure, x_j must appear as a mobile output name in q' (otherwise, the communication rule may be applied to reduce p to 0). Suppose (w.l.o.g.) $q' \equiv \overline{x_k}(u, x_j)|q''$ for some q''. Then there exists d with $(d, e_j) \in e_k$. But $e_k \in B_{n+1}$, and therefore $e_j \in B_n$, a contradiction. ◀

▶ **Proposition 17.** *For any closed term* p*,* $p \Downarrow_{R} a$ *if and only if* $[\![p]\!]_{R}(*, *) = a$.

Proof. From left-to-right, this follows from an induction on $\ell(p)$, using Lemmas 14 15 and 16. For the converse, suppose $[\![p]\!]_{\mathrm{R}}(*, *) = a$. By Proposition 8, $p \Downarrow_{\mathrm{R}} b$ for some b, and by the left-to-right implication just established, $a = b$. ◀

6.1 Full Abstraction

We will now establish a full abstraction result, showing that contextually equivalent R-weighted solo terms denote the same matrix over R – i.e. if $\Gamma \vdash p, q; \Delta$ where $\Gamma \cap \Delta = \varnothing$, $p \sim_{\mathrm{R}} q$ if and only if $[\![p]\!]_{\mathrm{R}} = [\![q]\!]_{\mathrm{R}}$[7]. This establishes the closeness of the syntax and semantics, and also that we can define a testing equivalence for weighted processes which obeys the algebraic laws of R-modules, and differential nets. Note that the R-weighted model of PCF in [16] is not fully abstract, essentially because it contains finite elements which are not denoted by any term. By contrast, we will show thatit is straightforward to define a basis of definable elements for each R-module in our model.

For an element $b \in B$, define $\chi_b^- : B \to I$ and (its transpose) $\chi_B^+ : I \to B$:
$\chi_b^-(a, *) = \chi_b^+(*, a) = 1$ if $a = b$: $\chi_b^-(a, *) = \chi_b^+(*, a) = 0$, otherwise.
$\{\chi_b^- \mid b \in B\}$ and $\{\chi_b^+ \mid b \in B\}$ are *bases* for the R-modules $\mathsf{Mat}_{\mathrm{R}}(B, I)$ and $\mathsf{Mat}_{\mathrm{R}}(I, B)$.

▶ **Lemma 18.** *For all $b \in B$, there exist terms $x \vdash p_b^-$ and $\vdash p_b^+; y$ which denote χ_b^- and χ_b^+.*

Proof. We prove that if $b \in B_k$ then χ_b^- and χ_b^+ are definable, by induction on k. $B_0 = \varnothing$, so suppose $b \in B_{k+1}$. Then b is a finite multiset $\{(b_1^-, b_1^+), \ldots, (b_m^-, b_m^+)\}$, where each $b_i^-, b_i^+ \in B_k$. So by hypothesis, for each i, $\chi_{b_i^+}^-$ is definable as a term $u \vdash p_i^-$ and each $\chi_{b_i^-}^+$ as $\vdash p_i^+; v$. Hence χ_b^- is definable as $p^-(x) = x(\underline{v}, \underline{u})p_1^- | p_1^+ | \ldots | x(\underline{v}, \underline{u})p_m^- | p_m^+$. By symmetry, χ_b^+ is definable as $\_ \vdash p^+(y); y$. ◀

▶ **Theorem 19.** *If $\Gamma \vdash q, q'; \Delta$, where $\Gamma \cap \Delta = \varnothing$ then $[\![q]\!]_R = [\![q']\!]_R \iff q \sim_R q'$.*

Proof. Equational soundness follows from Proposition 17.

To prove completeness, suppose $x_1, \ldots, x_m \vdash q, q'; y_1, \ldots, y_n$, and $[\![q]\!]_{\mathrm{R}} \neq [\![q']\!]_{\mathrm{R}}$. There exist $b_1^-, \ldots, b_m^-, b_1^+, \ldots, b_n^+ \in B$ such that $[\![q]\!](b_1^-, \ldots, b_m, b_1^+, \ldots, b_n^+) \neq [\![q']\!](b_1^-, \ldots, b_m, b_1^+, \ldots, b_n^+)$. By Lemma 18, each $\chi_{b_i^-}^+$ and $\chi_{b_j^+}^-$ are definable as terms $\_ \vdash p_i^+; x_i$ and $y_j \vdash p_j^-$ for $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$.

Let $C[\_] = \nu x_1 \ldots x_m.\nu y_1 \ldots y_n.[\_] | p_1^+ | \ldots | p_m^+ | p_1^- | \ldots | p_n^-$. By Proposition 17 $C[q] \Downarrow_{\mathrm{R}} [\![q]\!](b_1^-, \ldots, b_m, b_1^+, \ldots, b_n^+)$ and $C[q'] \Downarrow_{\mathrm{R}} [\![q]\!](b_1^-, \ldots, b_m, b_1^+, \ldots, b_n^+)$ – i.e. $q \not\sim_{\mathrm{R}} q'$ as required. ◀

7 Conclusions and Further Directions

We have defined a semantic basis for name mobility which focusses on quantitative testing. Areas in which it might be extended, refined or applied, include:

- Describing systems: which classes of processes can be expressed in the (acyclic part of) the calculus of R-solos? In particular, can we establish a precise relationship with Beffara's quantitative trace semantics of the πI-calculus [2].
- Expressing properties: which quantitative and qualitative properties of systems can be naturally expressed using quantitative solos?
- Computing sums of paths: For which terms can we give algorithms for computing the evaluation function?
- Constructing new models: Are there instances of reflexive differential bialgebras with richer structure, for example in categories of games or event structures?

[7] Our result is restricted to terms with disjoint input and output channels, essentially because input and output capabilities are modelled separately. Without this restriction, full abstraction may fail – e.g. in an idempotent semiring, the term $\nu y.\nu z.x(y, z) | \overline{x}(y, z)$ (which forwards to itself) is observationally equivalent to the unit, 1.

References

1 A. Bahamonde. Tensor product of partially-additive monoids. *Semigroup Forum*, 32(1):31–53, 1985.

2 Emmanuel Beffara. Quantitative testing semantics for non-interleaving. *CoRR*, abs/0906.3994, 2009.

3 G. Bellin and P. J. Scott. On the pi-calculus and linear logic. *Theoretical Computer Science*, 135(1):11–65, 1994.

4 R. F. Blute, J. R. B. Cockett, and R. A. G. Seely. Differential categories. *Mathematical. Structures in Comp. Sci.*, 16, 2006.

5 M. Boreale and F. Gaducci. Processes as formal power series: A coinductive approach to denotational semantics. *Theoretical Computer Science*, 360(1–3):440–458, 2006.

6 N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22:465–476, 1979.

7 T. Ehrhard and O. Laurent. Acyclic solos and differential interaction nets. *Logical Methods in Computer Science*, 6(3), 2010.

8 T. Ehrhard and O. Laurent. Interpreting a finitary pi-calculus in differential nets. *Information and Computation*, 208(6):606–633, 2010.

9 T. Ehrhard and L. Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309, 2003.

10 T. Ehrhard and L. Regnier. Differential interaction nets. *Theoretical Computer Science*, 364(2):166–195, 2006.

11 C. Fornet and G. Gonthier. The reflexive chemical abstract machine and the join-calculus. In *23rd ACM Symposium on Principles of Programming Languages (POPL'96)*, 1996.

12 J. S. Golan. *The theory of semirings with applications in mathematics and theoretical computer science.* Addison-Wesley, 1992.

13 R. Guitart. Tenseurs et machines. *Cahiers de Topologie et Geometrie Differentielle*, XXI(1):5–62, 1980.

14 K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152:437–686, 1995.

15 A. Joyal, R. Street, and D. Verity. Traced monoidal categories. *Math. Proc. Camb. Phil. Soc.*, 119:447–468, 1996.

16 J. Laird, G. Manzonetto, G. McCusker, and M. Pagani. Weighted relational models of typed lambda-calculi. In *Proceedings of LICS'13*, 2013.

17 C. Laneve and B. Victor. Solos in concert. *Mathematical Structures in Computer Science*, 13(5), 2003.

18 P. Melliès, N. Tabareau, and C. Tasson. An explicit formula for the free exponential modality of linear logic. In *Proc. ICALP'09*, number 5556 in LNCS, pages 247–260, 2009.

19 R. Milner. Functions as processes. *Math. Struct. in Computer Science*, 2(2):119–141, 1992.

20 Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part i. *I AND II. INFORMATION AND COMPUTATION*, 100, 1989.

21 M. Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.

22 C. Laneve. J. Parrow and B. Victor. Solo diagrams. In *Proceedings of TACS 2001*, LNCS. Springer, 2001.

23 J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proceedings of LICS'98*, pages 176–185. IEEE press, 1998.

24 D. Sangiorgi and D. Walker. *The Pi-Calculus: A theory of mobile processes.* Cambridge University Press, 2001.

25 L. Wischik and P. Gardner. Explicit fusions. *Theoretical Computer Science*, 340(3):606–630, 2005.

Focusing in Orthologic[*]

Olivier Laurent[†]

Université de Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France

Abstract

We propose new sequent calculus systems for *orthologic* (also known as *minimal quantum logic*) which satisfy the cut elimination property. The first one is a very simple system relying on the involutive status of negation. The second one incorporates the notion of *focusing* (coming from linear logic) to add constraints on proofs and thus to facilitate proof search. We demonstrate how to take benefits from the new systems in automatic proof search for orthologic.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases orthologic, focusing, minimal quantum logic, linear logic, automatic proof search, cut elimination

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.25

1 Introduction

Classical (propositional) logic can be used to reason about facts in classical mechanics and is related with the lattice structure of Boolean algebras. On its side, quantum (propositional) logic has been introduced to represent observable facts in quantum mechanics. It is provided as an axiomatization of the lattice structure of the closed subspaces of Hilbert spaces. This corresponds to the structure of so-called orthomodular lattices. Among the properties of these lattices, and thus of quantum logic, one finds the orthomodularity law ($a \leq b \Longrightarrow b \leq a \vee (\neg a \wedge b)$) which is a very weak form of distributivity. Removing this law gives the notion of *ortholattice* and leads to the associated *orthologic* (also called minimal quantum logic, as it can be defined as quantum logic without orthomodularity). In the description and reasoning about quantum properties, quantum logic is more accurate than orthologic. Nevertheless a formula valid in orthologic is also valid in quantum logic, and thus provides a valid quantum property. In the current state of the art, orthologic benefits of much better logical properties than plain quantum logic (in proof theory in particular) and, since it also corresponds to a nice class of lattices, many authors focus on it [9, 14, 10, 13, 4]. From the point of view of lattice theory, ortholattices are bounded lattices with an involutive negation such that $p \vee \neg p = \top$. As a consequence they can be understood as Boolean lattices without distributivity, and indeed distributive ortholattices are exactly Boolean lattices.

The main topic of the present work is the study of the proof theory of orthologic, from the sequent calculus point of view. Sound and complete sequent calculi satisfying the cut-elimination property already occur in the literature (see for example [14, 13, 6]). Our first result is another such calculus which is particularly simple: each sequent has exactly two formulas and only seven rules are required. It relies on ideas of J.-Y. Girard in linear logic [7]

[*] This work was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007), and by projects Récré (ANR-11-BS02-0010) and Elica (ANR-14-CE25-0005), all operated by the French National Research Agency (ANR).

[†] `http://perso.ens-lyon.fr/olivier.laurent/`

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 25; pp. 25:1–25:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for the representation of systems with an involutive negation, and shows how orthologic can be seen as an extension of the additive fragment of linear logic with one new contraction-weakening rule. The second and main contribution of this paper lies in the development of a “second-level proof-theory” for orthologic by investigating the notion of focusing in this setting.

Focusing, introduced in linear logic by J.-M. Andreoli [1], is a constraint on the structure of proofs which requires connectives sharing some structural properties (like reversibility) to be grouped together. The key point is that this restriction is sound and complete: focused proofs are proofs and any provable sequent admits a focused proof. Together with cut elimination, focusing can be used as a strong tool in proof search and proof study since it reduces the search space to focused proofs. Focusing has also been used to define new logical systems [8].

In the case of orthologic, we show that focusing can be defined and interacts particularly well with the 2-formulas sequents. In particular, not only logical rules associated with connectives are constrained but also structural rules can be organised. The exchange rule can be hidden easily in the specific focusing rules and the contraction-weakening rule becomes precisely constrained. As a consequence, we obtain a bound on the height of all focused proofs of a given sequent, which is rarely the case in the presence of a contraction rule. Starting from this remark, we experiment proof search strategies for orthologic based on our focused system.

In Section 2, we recall the definition of ortholattice and orthologic with the main results from the literature on sequent calculus and cut-elimination for orthologic. In Section 3, we introduce the sequent calculus OL (inspired by additive linear logic) with a few properties. Section 4 gives the two-steps construction of the focused system OL_f. We explain how focusing is applied to orthologic and we prove soundness, completeness and cut-elimination. The last Section 5 is dedicated to the application of OL_f in (backward and forward) proof search for orthologic. This is based on upper bounds on the height of proofs and on additional structural properties of focused cut-free proofs.

2 Ortholattices and Orthologic

Orthologic or *minimal quantum logic* is the logic associated with the order relation of *ortholattices* (for some results about ortholattices, see for example [2]).

▶ **Definition 1** (Ortholattice)**.** An *ortholattice* $\mathcal{O}$ is a bounded lattice (a lattice with smallest and biggest elements $\bot$ and $\top$) with an order-reversing involution $p \mapsto \neg p$ (also often denoted $p^\perp$ in the literature), called *orthocomplement*, satisfying $p \vee \neg p = \top$ (for all p in $\mathcal{O}$).

In particular the following properties hold for any two elements p and q of any ortholattice: $p \leq q \Longrightarrow \neg q \leq \neg p$, $\quad \neg\neg p = p$, $\quad \neg\bot = \top$, $\quad \neg(p \vee q) = \neg p \wedge \neg q$, $\quad p \wedge \neg p = \bot$, as well as the other De Morgan’s laws, but there is no distributivity law between $\wedge$ and $\vee$.

Orthologic is the logic associated with the class of ortholattices, or conversely ortholattices are the algebras associated with orthologic. Formulas in orthologic are built using connectives corresponding to the basic operations of ortholattices:

$$A ::= X \mid A \wedge A \mid A \vee A \mid \top \mid \bot \mid \neg A$$

where X ranges over elements of a given countable set $\mathcal{X}$ of *variables*.

We want then $A \vdash B$ to be derivable in orthologic if and only if $A \leq B$ is true in any ortholattice $\mathcal{O}$ (for every interpretation of variables as elements of $\mathcal{O}$, and with connectives in A and B interpreted through the corresponding operations of $\mathcal{O}$). In particular, the

Lindenbaum algebra associated with orthologic over the set $\mathcal{X}$ is the free ortholattice over $\mathcal{X}$ (which is infinite as soon as $\mathcal{X}$ contains at least two elements [3]).

If we adopt a sequent calculus style presentation, an (sound and complete) axiomatization of orthologic can be given by the following axioms and rules (in the spirit of [9]):

$$\begin{array}{llll}
\frac{}{A \vdash A} & \frac{A \vdash B \qquad B \vdash C}{A \vdash C} & & \\[2ex]
\frac{}{A \wedge B \vdash A} & \frac{}{A \wedge B \vdash B} & \frac{C \vdash A \qquad C \vdash B}{C \vdash A \wedge B} & \frac{}{C \vdash \top} \\[2ex]
\frac{}{A \vdash A \vee B} & \frac{}{B \vdash A \vee B} & \frac{A \vdash C \qquad B \vdash C}{A \vee B \vdash C} & \frac{}{\bot \vdash C} \\[2ex]
\frac{A \vdash B}{\neg B \vdash \neg A} & \frac{}{A \vdash \neg\neg A} & \frac{}{\neg\neg A \vdash A} & \frac{}{\top \vdash A \vee \neg A}
\end{array}$$

The first line corresponds to an (pre) order relation. The second and third lines correspond to a bounded inf semi-lattice and bounded sup semi-lattice (thus together they provide us the structure of a bounded lattice). The fourth line adds the missing ortholattice ingredients related with the orthocomplement $\neg A$.

▶ **Example 2.** If one wants to prove that for any p and q in an ortholattice, we have: $\top \leq ((p \wedge q) \vee \neg p) \vee \neg q$. We can either use algebraic properties of ortholattices (which have to be proved as well): $((p \wedge q) \vee \neg p) \vee \neg q = (p \wedge q) \vee (\neg p \vee \neg q) = (p \wedge q) \vee \neg(p \wedge q) = \top$ or we can use, on the logic side, a derivation with conclusion the corresponding sequent $\top \leq ((X \wedge Y) \vee \neg X) \vee \neg Y$. This requires us to use most of the rules above.

The axiomatization proposed above is a direct translation of the order-theoretic definition of ortholattices. From a proof-theoretic point of view, it has strong defects such has the impossibility of eliminating the cut rule:

$$\frac{A \vdash B \qquad B \vdash C}{A \vdash C}\ \mathit{cut}$$

(which encodes the transitivity of the order relation). Example 2 could not be derived without this rule for example. A reason for trying to avoid the cut rule is that when studying a property like $A \vdash C$, the cut rule tells us that we may need to invent some arbitrary B (unrelated with A and C). This may lead us to difficulties, undecidability, etc. In the opposite, cut-free systems usually satisfy the *sub-formula property* stating that every formula appearing in a proof of a given sequent is a sub-formula of a formula of this sequent. The idea of finding presentations of the logic associated with lattices in such a way that cut (or transitivity) could be eliminated goes back to Whitman [15] with applications to the theory of lattices. In the case of ortholattices, one can find such an axiomatization in [14] under the name OCL+ (also called GOL in [5]):

OCL+

$$\begin{array}{lll}
\frac{}{A \vdash A}\ ax & \frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta}\ wL & \frac{\Gamma \vdash \Delta}{\Gamma \vdash A, \Delta}\ wR \\[2ex]
\frac{\Gamma, A \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ \wedge_1 L & \frac{\Gamma, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ \wedge_2 L & \frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}\ \wedge R \\[2ex]
\frac{\Gamma \vdash A, \Delta}{\Gamma \vdash A \vee B, \Delta}\ \vee_1 R & \frac{\Gamma \vdash B, \Delta}{\Gamma \vdash A \vee B, \Delta}\ \vee_2 R & \frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\ \vee L
\end{array}$$

$$\frac{}{\Gamma \vdash \top, \Delta}\ \top R \qquad \frac{}{\Gamma, \bot \vdash \Delta}\ \bot L \qquad \frac{\Gamma, A \vdash \Delta}{\Gamma \vdash \neg A, \Delta}\ \neg R \qquad \frac{\Gamma \vdash A, \Delta}{\Gamma, \neg A \vdash \Delta}\ \neg L$$

where sequents $\Gamma \vdash \Delta$ are given from two finite *sets* Γ and Δ of formulas such that the size of Γ plus the size of Δ is *at most* 2 (and the comma denotes set union).

▶ **Example 3.** We can prove in OCL+ the sequent of Example 2:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{X \vdash X}\,ax}{\vdash X, \neg X}\,\neg R}{\vdash X, (X \wedge Y) \vee \neg X}\,\vee_2 R}{\vdash X, ((X \wedge Y) \vee \neg X) \vee \neg Y}\,\vee_1 R \qquad \dfrac{\dfrac{\dfrac{}{Y \vdash Y}\,ax}{\vdash Y, \neg Y}\,\neg R}{\vdash Y, ((X \wedge Y) \vee \neg X) \vee \neg Y}\,\vee_2 R}{\vdash X \wedge Y, ((X \wedge Y) \vee \neg X) \vee \neg Y}\,\wedge R}{\vdash (X \wedge Y) \vee \neg X, ((X \wedge Y) \vee \neg X) \vee \neg Y}\,\vee_1 R}{\vdash ((X \wedge Y) \vee \neg X) \vee \neg Y}\,\vee_1 R}{\top \vdash ((X \wedge Y) \vee \neg X) \vee \neg Y}\,wL$$

The following key properties of OCL+ are proved in [14]:

▶ **Theorem 4** (Cut Elimination in OCL+). *The cut rule* $\dfrac{\Gamma_1 \vdash A, \Delta_1 \qquad \Gamma_2, A \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}$ *is admissible in* OCL+.

▶ **Theorem 5** (Soundness and Completeness of OCL+). OCL+ *is sound and complete for orthologic.*

By looking at the structure of the rules, one can see there is an important symmetry between $\vee$ on the left and $\wedge$ on the right, $\wedge$ on the left and $\vee$ on the right, $\bot$ on the left and $\top$ on the right, etc. This is not very surprising in a context where negation is an involution, and this is an incarnation of De Morgan's duality between $\wedge$ and $\vee$ and $\top$ and $\bot$. J.-Y. Girard has shown for linear logic [7] how to simplify sequent calculi in the presence of an involutive negation by restricting negation to variables and by considering one-sided sequents only. This idea has been partly applied in [6] where they define formulas for orthologic as:

$$A ::= X \mid A \wedge A \mid A \vee A \mid \top \mid \bot \mid \neg X$$

and negation is then extended to all formulas by induction (it is not a true connective anymore):

$$\neg(\neg X) := X \quad \neg(\bot) := \top \quad \neg(\top) := \bot \quad \neg(A \vee B) := \neg A \wedge \neg B \quad \neg(A \wedge B) := \neg A \vee \neg B$$

so that we obtain $\neg\neg A = A$ for any A. However the system proposed in [6] does not really take benefits from this encoded involutive negation on formulas, since they use two-sided sequents. One can also note that no remark is given in [6] regarding the number of formulas in sequents. However one can see that, in their system, $\Gamma \vdash \Delta$ is provable if and only if $\bigwedge \Gamma \vdash \bigvee \Delta$ is provable, and that a proof of a sequent $\Gamma \vdash \Delta$ with at most one formula in Γ and at most one formula in Δ contains only sequents satisfying this property.

We propose to go further in this direction of involutive negation to target a simpler sequent calculus system for orthologic.

3 One-Sided Orthologic

In order to clarify the analysis and to be closer to an implementation, we prefer to consider sequents based on lists rather than sets or multi-sets. The main difference with respect to OCL+ is the necessity to use an explicit contraction rule and an explicit exchange rule. We thus consider two kinds of sequents: $\vdash A, B$ and $\vdash A$. As a notation, Π corresponds to 0 or 1 formula so that $\vdash A, \Pi$ is a common notation for both kinds of sequents. Like in [6], formulas are built with negation on variables only:

$$A ::= X \mid A \wedge A \mid A \vee A \mid \top \mid \bot \mid \neg X$$

and, by moving to a one-sided list-based system, we obtain derivation rules like:

$$\frac{}{\vdash \neg A, A}\,ax \qquad \frac{\vdash A, B}{\vdash B, A}\,ex \qquad \frac{\vdash A, A}{\vdash A}\,c \qquad \frac{\vdash A}{\vdash A, B}\,w \qquad \frac{\vdash A, \Pi}{\vdash A \vee B, \Pi}\,\vee_1 \qquad \cdots$$

But we can optimise these rules. First, we can assume Π not to be empty since the case of an empty Π is derivable from the non-empty case. For example, for the $(\vee_1)$ rule:

$$\cfrac{\cfrac{\cfrac{\vdash A}{\vdash A, A \vee B}\,w}{\vdash A \vee B, A \vee B}\,\vee_1}{\vdash A \vee B}\,c$$

Second, once we thus consider only logical rules with two formulas in sequents, the only rule with a premise with only one formula is the (w) rule and the only rule with a conclusion with only one formula is the (c) rule. This means that in a proof of a sequent with two formulas, (c) and (w) rules always come together, one above the other, and we can group them. Finally a sequent $\vdash A$ can always be encoded as $\vdash A, A$ since one is provable if and only if the other is (thanks to the rules (c) and (w)). We thus focus on sequents $\vdash A, B$ only, and on the following rules:

OL

$$\frac{}{\vdash \neg A, A}\,ax \qquad \frac{\vdash A, B}{\vdash B, A}\,ex \qquad \frac{\vdash A, A}{\vdash A, B}\,cw$$

$$\frac{\vdash A, C}{\vdash A \vee B, C}\,\vee_1 \qquad \frac{\vdash B, C}{\vdash A \vee B, C}\,\vee_2 \qquad \frac{\vdash A, C \qquad \vdash B, C}{\vdash A \wedge B, C}\,\wedge \qquad \frac{}{\vdash \top, C}\,\top$$

This sequent calculus with 7 rules (6 rules in its multi-set-based and set-based versions) does not seem to occur in the literature and looks simpler than all the sound and complete calculi for orthologic we have found. We call it OL. Relying on the remarks above, we have:

▶ **Theorem 6** (Soundness and Completeness of OL). *$\vdash \neg A, B$ is provable in* OL *if and only if $A \vdash B$ is provable in* OCL+*, so that* OL *is sound and complete for orthologic.*

Proof. To be completely precise, we have to recall that formulas of OL are all formulas of OCL+. While the converse is not true, there is a canonical mapping of formulas of OCL+ into formulas of OL obtained by unfolding the definition of $\neg$. For soundness, we use Theorem 4. Concerning completeness, we prove simultaneously that $A \vdash B$ in OCL+ entails $\vdash \neg A, B$ in OL, $A \vdash$ in OCL+ entails $\vdash \neg A, \neg A$ in OL and $\vdash B$ in OCL+ entails $\vdash B, B$ in OL. ◀

For readers familiar with linear logic [7], this calculus OL can be seen as one-sided additive linear logic extended with the (cw) rule, if we replace $\vee$ by $\oplus$, $\wedge$ by $\&$ and $\bot$ by 0.

▶ **Example 7.** We can prove in OL the sequent of Example 2 in its one-sided version:

$$\begin{prooftree}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$\vdash \neg X, X$}
\RightLabel{$\vee_2$}
\UnaryInfC{$\vdash (X \wedge Y) \vee \neg X, X$}
\RightLabel{$\vee_1$}
\UnaryInfC{$\vdash ((X \wedge Y) \vee \neg X) \vee \neg Y, X$}
\RightLabel{ex}
\UnaryInfC{$\vdash X, ((X \wedge Y) \vee \neg X) \vee \neg Y$}
\AxiomC{}
\RightLabel{ax}
\UnaryInfC{$\vdash \neg Y, Y$}
\RightLabel{$\vee_2$}
\UnaryInfC{$\vdash ((X \wedge Y) \vee \neg X) \vee \neg Y, Y$}
\RightLabel{ex}
\UnaryInfC{$\vdash Y, ((X \wedge Y) \vee \neg X) \vee \neg Y$}
\RightLabel{$\wedge$}
\BinaryInfC{$\vdash X \wedge Y, ((X \wedge Y) \vee \neg X) \vee \neg Y$}
\RightLabel{$\vee_1$}
\UnaryInfC{$\vdash (X \wedge Y) \vee \neg X, ((X \wedge Y) \vee \neg X) \vee \neg Y$}
\RightLabel{$\vee_1$}
\UnaryInfC{$\vdash ((X \wedge Y) \vee \neg X) \vee \neg Y, ((X \wedge Y) \vee \neg X) \vee \neg Y$}
\RightLabel{cw}
\UnaryInfC{$\vdash ((X \wedge Y) \vee \neg X) \vee \neg Y, \bot$}
\RightLabel{ex}
\UnaryInfC{$\vdash \bot, ((X \wedge Y) \vee \neg X) \vee \neg Y$}
\end{prooftree}$$

We now describe a few properties of OL which will be used later. First, the cut rule $\dfrac{\vdash A, B \qquad \vdash \neg B, C}{\vdash A, C}$ is admissible (see Proposition 19 for an indirect proof). Also:

▶ **Proposition 8** (Axiom expansion for OL). *If we restrict the axiom rule of* OL *to its variable case* $\dfrac{}{\vdash \neg X, X}\,ax_v$ *, the general rule (ax) is derivable.*

▶ **Lemma 9** (Reversibility of $\wedge$). *$\vdash A \wedge B, C$ is provable iff both $\vdash A, C$ and $\vdash B, C$ are.*

▶ **Lemma 10** (Reversing). *If we restrict the (cw) rule to formulas of the shape $A_1 \vee A_2$:*

$$\dfrac{\vdash A_1 \vee A_2, A_1 \vee A_2}{\vdash A_1 \vee A_2, B}\,cw_{\vee}$$

where moreover B is neither $\top$ nor a $\wedge$, the general rule (cw) is admissible.

Proof. This is done in two steps, first by proving the restriction on A (by induction on A for an arbitrary B) and then the restriction on B (by induction on B, with $A = A_1 \vee A_2$). ◀

4 Focused Orthologic

Relying on the strong relation between the sequent calculus OL and linear logic, we import the idea of *focusing* [1]. This constraint on the structure of proofs is based on an analysis of the polarity of connectives, by separating those which are reversible and those which are not. By reducing the space of proofs of each formula, it is a strong tool for accelerating proof search. In orthologic, the connectives $\wedge$ and $\top$ are reversible: the conclusion of their introduction rule implies its premises (see Lemma 9 for example). Such connectives are also called *asynchronous* or negative. Their dual connectives are called *synchronous* or positive. Following this pattern, we separate formulas into synchronous and asynchronous ones according to their main connective: X, $\bot$ and $A \vee B$ are synchronous, and $\neg X$, $\top$ and $A \wedge B$ are asynchronous. So that A is synchronous if and only if $\neg A$ is asynchronous. The choice for variables is in fact arbitrary, as soon as we preserve this dual polarity between X and $\neg X$ for each of them.

4.1 A First Focused System $\mathsf{OL}_{\mathrm{f}}^{\mathbf{0}}$

Dealing with variables in focused systems is delicate, so we recommend the reader not very familiar with focusing to concentrate on the other aspects of the system first.

A key result will be to prove the focused system to be as expressive as OL (and thus sound and complete for orthologic). In order to make this as simple and clear as possible, we will work in two steps. Indeed some optimisations (to be introduced later on in Section 4.4) would make a direct translation more difficult.

Our first focused system $\mathsf{OL}_{\mathsf{f}}^0$ is based on four kinds of sequents. For each of them, we give an *informal explanation* based on how we can find a proof of such a sequent, thus from the point of view of a bottom-up reading of proofs and rules:

- In a sequent $\vdash\ \Uparrow A, B$, all the asynchronous connectives at the roots of A and B (in formulas A and B seen as trees) will be deconstructed and after that, A and B will be synchronous (or negation of a variable) and allowed to move to the left of $\Uparrow$. In fact we first work on A and then we move to a sequent $\vdash A \Uparrow B$.
- In a sequent $\vdash A \Uparrow B$, A is synchronous or is the negation of a variable. The asynchronous connectives at the root of B will be deconstructed and after that, B will be synchronous (or negation of a variable) and allowed to move to the left of $\Uparrow$.
- In a sequent $\vdash A, B \Uparrow$, A and B are synchronous or the negation of a variable. We have to select a synchronous formula and start decomposing its synchronous connectives at the root, in a sequent $\vdash A \Downarrow B$. Before that, we can apply contraction-weakening rules to A and B. This is the main place where choices have to be made during proof search.
- In a sequent $\vdash A \Downarrow B$, A is synchronous or is the negation of a variable. The synchronous connectives at the root of B will be deconstructed and after that, B will be asynchronous (and we will start decomposing its asynchronous connectives at the root in a sequent $\vdash A \Uparrow B$). Choices concerning the decomposition of $\vee$ will have to be made here.

Note, sequents $\vdash\ \Uparrow A, B$ are crucial for the comparison with other systems but play a weak role inside this system. Indeed they occur only in proofs of sequents of the same shape and only at the bottom part of such a proof. As soon as we reach a sequent $\vdash A \Uparrow B$ (in the bottom-up reading of a proof), we will not find any other sequent $\vdash\ \Uparrow A, B$ above.

Let us be more formal now with the explicit list of the rules of the system $\mathsf{OL}_{\mathsf{f}}^0$:

$\mathsf{OL}_{\mathsf{f}}^0$

$$\frac{\vdash\ \Uparrow A, C \qquad \vdash\ \Uparrow B, C}{\vdash\ \Uparrow A \wedge B, C}\,{\Uparrow}\wedge \qquad \frac{}{\vdash\ \Uparrow \top, C}\,{\Uparrow}\top \qquad \frac{\vdash A \Uparrow C}{\vdash\ \Uparrow A, C}\,{\Uparrow}\mathrm{R}$$

$$\frac{\vdash C \Uparrow A \qquad \vdash C \Uparrow B}{\vdash C \Uparrow A \wedge B}\,\wedge{\Uparrow} \qquad [\text{(s) or (n)}]\ \frac{}{\vdash A \Uparrow \top}\,\top{\Uparrow} \qquad \frac{\vdash C, A \Uparrow}{\vdash C \Uparrow A}\,\mathrm{R}{\Uparrow}$$

$$[\text{(s) or (n)}]\ \frac{\vdash C, C \Uparrow}{\vdash C, A \Uparrow}\,cw_1 \qquad [\text{(s) or (n)}]\ \frac{\vdash C, C \Uparrow}{\vdash A, C \Uparrow}\,cw_2$$

$$[\text{(s)}]\ \frac{\vdash C \Downarrow A}{\vdash A, C \Uparrow}\,\mathrm{D}_1 \qquad [\text{(s)}]\ \frac{\vdash C \Downarrow A}{\vdash C, A \Uparrow}\,\mathrm{D}_2$$

$$\frac{}{\vdash \neg X \Downarrow X}\,ax_v \qquad \frac{\vdash C \Downarrow A}{\vdash C \Downarrow A \vee B}\,\vee_1 \qquad \frac{\vdash C \Downarrow B}{\vdash C \Downarrow A \vee B}\,\vee_2 \qquad [\text{(a)}]\ \frac{\vdash C \Uparrow A}{\vdash C \Downarrow A}\,\mathrm{R}{\Downarrow}$$

with the following side conditions written between square brackets [_]:
(a) A is asynchronous (s) A is synchronous (n) A is the negation of a variable.

One could have been more explicit by asking [(s) or (n)] as side condition in the (${\Uparrow}$R) and (R${\Uparrow}$) rules but the following lemma proves these two side conditions to be redundant.

▶ **Lemma 11.** *If $\vdash A \Uparrow C$ or $\vdash A, B \Uparrow$ or $\vdash A \Downarrow C$ is provable then A and B are synchronous or the negation of a variable.*

▶ **Example 12.** The sequent $\vdash (X \vee A) \vee B, (C \vee (D \vee \neg X)) \wedge \top$ has many proofs in the systems of the previous sections, in particular in OL. However the corresponding sequent $\vdash\ \Uparrow (X \vee A) \vee B, (C \vee (D \vee \neg X)) \wedge \top$ has a unique proof in $\mathsf{OL}_{\mathsf{f}}^0$:

$$\begin{prooftree}
\AxiomC{}
\RightLabel{ax_v}
\UnaryInfC{$\vdash \neg X \Downarrow X$}
\RightLabel{$\vee_1$}
\UnaryInfC{$\vdash \neg X \Downarrow X \vee A$}
\RightLabel{$\vee_1$}
\UnaryInfC{$\vdash \neg X \Downarrow (X \vee A) \vee B$}
\RightLabel{D_1}
\UnaryInfC{$\vdash (X \vee A) \vee B, \neg X \Uparrow$}
\RightLabel{$\mathrm{R}\Uparrow$}
\UnaryInfC{$\vdash (X \vee A) \vee B \Uparrow \neg X$}
\RightLabel{$\mathrm{R}\Downarrow$}
\UnaryInfC{$\vdash (X \vee A) \vee B \Downarrow \neg X$}
\RightLabel{$\vee_2$}
\UnaryInfC{$\vdash (X \vee A) \vee B \Downarrow D \vee \neg X$}
\RightLabel{$\vee_2$}
\UnaryInfC{$\vdash (X \vee A) \vee B \Downarrow C \vee (D \vee \neg X)$}
\RightLabel{D_2}
\UnaryInfC{$\vdash (X \vee A) \vee B, C \vee (D \vee \neg X) \Uparrow$}
\RightLabel{$\mathrm{R}\Uparrow$}
\UnaryInfC{$\vdash (X \vee A) \vee B \Uparrow C \vee (D \vee \neg X)$}
\AxiomC{}
\RightLabel{$\top\Uparrow$}
\UnaryInfC{$\vdash (X \vee A) \vee B \Uparrow \top$}
\RightLabel{$\wedge\Uparrow$}
\BinaryInfC{$\vdash (X \vee A) \vee B \Uparrow (C \vee (D \vee \neg X)) \wedge \top$}
\RightLabel{$\Uparrow\mathrm{R}$}
\UnaryInfC{$\vdash \Uparrow (X \vee A) \vee B, (C \vee (D \vee \neg X)) \wedge \top$}
\end{prooftree}$$

One can prove the soundness of $\mathsf{OL}^0_{\mathrm{f}}$ with respect to orthologic by translation into OL.

▶ **Proposition 13** (Soundness of $\mathsf{OL}^0_{\mathrm{f}}$)**.** *If* $\vdash\ \Uparrow A, B$ *or* $\vdash A \Uparrow B$ *or* $\vdash A, B \Uparrow$ *or* $\vdash A \Downarrow B$ *is provable in* $\mathsf{OL}^0_{\mathrm{f}}$ *then* $\vdash A, B$ *is provable in* OL*.*

To conclude this section, here are a few simple facts which will be useful later and which can be obtained by simple induction on proofs:

- $\vdash X, Y \Uparrow$, $\vdash \neg X, \neg Y \Uparrow$ and $\vdash \bot, \bot \Uparrow$ are not provable (both if $X = Y$ or $X \neq Y$);
- if $\vdash A, B \Uparrow$ is provable then $\vdash B, A \Uparrow$ as well (and with a proof of the same size);
- if $\vdash A, A \Uparrow$ is provable then the proof contains a proof of $\vdash A \Downarrow A$.

4.2 Cut Elimination in $\mathsf{OL}^0_{\mathrm{f}}$

Due to the very rigid structure of proofs in focused systems, the possibility of enriching them with admissible cut rules is often used in their study [8, 11] (in particular for expressiveness analysis). It is the tool we are going to use here in order to prove the completeness of $\mathsf{OL}^0_{\mathrm{f}}$ with respect to orthologic.

▶ **Theorem 14** (Cut Elimination in $\mathsf{OL}^0_{\mathrm{f}}$)**.** *The following cut rules are admissible in* $\mathsf{OL}^0_{\mathrm{f}}$*:*

C synchronous

$$\dfrac{\vdash X \Downarrow A \qquad \vdash \neg X \Downarrow C}{\vdash C \Downarrow A}\ v\text{-}cut_2$$

C synchronous

$$\dfrac{\vdash X \Uparrow A \qquad \vdash \neg X \Downarrow C}{\vdash C \Uparrow A}\ v\text{-}cut_3$$

$$\dfrac{\vdash A, X \Uparrow \qquad \vdash C, \neg X \Uparrow}{\vdash A, C \Uparrow}\ v\text{-}cut_1$$

B asynchronous or variable

$$\dfrac{\vdash A \Uparrow B \qquad \vdash C \Uparrow \neg B}{\vdash A, C \Uparrow}\ cut_1$$

$$\dfrac{\vdash A \Uparrow B \qquad \vdash C, \neg B \Uparrow}{\vdash A, C \Uparrow}\ cut_2$$

B asynchronous

$$\dfrac{\vdash A \Uparrow B \qquad \vdash \neg B \Downarrow C}{\vdash A \Downarrow C}\ cut_3$$

B asynchronous or variable

$$\dfrac{\vdash A \Uparrow B \qquad \vdash C \Downarrow \neg B}{\vdash A, C \Uparrow}\ cut_4$$

$$\dfrac{\vdash A \Uparrow B \qquad \vdash \neg B \Uparrow C}{\vdash A \Uparrow C}\ cut_5$$

$$\dfrac{\vdash\ \Uparrow A, B \qquad \vdash\ \Uparrow C, \neg B}{\vdash\ \Uparrow A, C}\ cut_0$$

$$\dfrac{\vdash\ \Uparrow A, B \qquad \vdash C \Uparrow \neg B}{\vdash C \Uparrow A}\ cut_0'$$

Proof. This is a proof involving many cases which require a precise management of the four kinds of sequents. We try to explain the key ingredients which work in successive steps.

- We prove simultaneously the admissibility of (v-cut_2) and (v-cut_3) by induction on the size of the left premise.
- We deduce the admissibility of (v-cut_1) by induction on the size of the left premise. For example:

$$\dfrac{\dfrac{\vdash X \Downarrow A}{\vdash A, X \Uparrow}\,\mathrm{D}_1 \qquad \dfrac{\vdash \neg X \Downarrow C}{\vdash C, \neg X \Uparrow}\,\mathrm{D}_1}{\vdash A, C \Uparrow}\,v\text{-}cut_1 \qquad \rightsquigarrow \qquad \dfrac{\dfrac{\vdash X \Downarrow A \qquad \vdash \neg X \Downarrow C}{\vdash C \Downarrow A}\,v\text{-}cut_2}{\vdash A, C \Uparrow}\,\mathrm{D}_1$$

  since A and C are synchronous.
- Using the previous steps, we prove simultaneously the admissibility of (cut_1), (cut_2), (cut_3), (cut_4) and (cut_5) by induction on the pair (f, p) where f is the size of the cut-formula B and p is the size of the right premise. The crucial cases are the following:
  - Starting from:

$$\dfrac{\dfrac{\vdash A \Uparrow B_1 \qquad \vdash A \Uparrow B_2}{\vdash A \Uparrow B_1 \wedge B_2}\,\wedge\Uparrow \qquad \dfrac{\vdash C \Downarrow \neg B_1}{\vdash C \Downarrow \neg B_1 \vee \neg B_2}\,\vee_1}{\vdash A, C \Uparrow}\,cut_4$$

    we can apply the induction hypothesis by means of (cut_4) with a smaller cut formula.
  - If B is asynchronous, we have:

$$\dfrac{\vdash A \Uparrow B \qquad \dfrac{\vdash \neg B \Downarrow C}{\vdash C, \neg B \Uparrow}\,\mathrm{D}_1}{\vdash A, C \Uparrow}\,cut_2 \qquad \rightsquigarrow \qquad \dfrac{\dfrac{\vdash A \Uparrow B \qquad \vdash \neg B \Downarrow C}{\vdash A \Downarrow C}\,cut_3}{\vdash A, C \Uparrow}\,\mathrm{D}_2$$

    otherwise B is a variable so that $\vdash A \Uparrow X$ must come from ($\mathrm{R}\Uparrow$) and we apply (v-cut_1).
  - The most tricky case is contraction where we need two induction steps:

$$\dfrac{\vdash A \Uparrow B \qquad \dfrac{\dfrac{\dfrac{\vdash \neg B \Downarrow \neg B}{\vdash \neg B, \neg B \Uparrow}\,\mathrm{D}}{\vdash \neg B, \neg B \Uparrow}\,cw}{\vdash C, \neg B \Uparrow}\,cw_2}{\vdash A, C \Uparrow}\,cut_2 \qquad \rightsquigarrow \qquad \dfrac{\dfrac{\vdash A \Uparrow B \qquad \dfrac{\vdash A \Uparrow B \qquad \vdash \neg B \Downarrow \neg B}{\vdash A \Downarrow \neg B}\,cut_3}{\vdash A, A \Uparrow}\,cut_4}{\vdash A, C \Uparrow}\,cw_1$$

    First we apply (cut_3) with a smaller right premise and then, by transforming one more step the (cut_4), we reach a smaller cut formula.
- We deduce the case (cut'_0) and then (cut_0), by induction on the size of the left premise. ◀

Among the 10 cut rules considered in the theorem above, mainly two will be used now (namely cut_0 and cut'_0). The other rules were however necessary as intermediary steps to prove the admissibility of these two rules.

4.3 Completeness of $\mathsf{OL}^0_{\mathrm{f}}$

We are going to translate proofs of OL into proofs of $\mathsf{OL}^0_{\mathrm{f}}$. We start with some preliminary results about sequents $\vdash\ \Uparrow A, B$ in $\mathsf{OL}^0_{\mathrm{f}}$ which will be the target of sequents of OL.

▶ **Lemma 15.** *The following rules are admissible in* $\mathsf{OL}^0_{\mathrm{f}}$*:*

$$\dfrac{}{\vdash\ \Uparrow C, \top} \qquad \dfrac{\vdash\ \Uparrow C, A \qquad \vdash\ \Uparrow C, B}{\vdash\ \Uparrow C, A \wedge B} \qquad \dfrac{\vdash A \Uparrow C}{\vdash\ \Uparrow C, A} \qquad \dfrac{\vdash\ \Uparrow A, C}{\vdash\ \Uparrow C, A}$$

▶ **Lemma 16.** *In OL_f^0, the following rules are admissible (and similarly for $B \lor A$ instead of $A \lor B$):*

$$\begin{array}{c} A \textit{ asynchronous} \\ \dfrac{\vdash C \Uparrow A}{\vdash A \lor B \Uparrow C} \end{array} \qquad \begin{array}{c} A \textit{ synchronous} \\ \dfrac{\vdash A \Uparrow C}{\vdash A \lor B \Uparrow C} \end{array} \qquad \begin{array}{c} A \textit{ synchronous} \\ \dfrac{\vdash A, C \Uparrow}{\vdash A \lor B, C \Uparrow} \end{array} \qquad \begin{array}{c} A \textit{ synchronous} \\ \dfrac{\vdash A \Downarrow C}{\vdash A \lor B \Downarrow C} \end{array}$$

▶ **Proposition 17** (Axiom expansion for OL_f^0)**.** *If A is synchronous or a negation of a variable, $\vdash A \Uparrow \neg A$ is provable.*

This leads us to the completeness of OL_f^0 for orthologic by means of the completeness of OL and the following translation result:

▶ **Theorem 18** (Completeness of OL_f^0)**.** *If $\vdash A, B$ is provable in OL then $\vdash\ \Uparrow A, B$ is provable in OL_f^0.*

Proof. By induction on the proof of $\vdash A, B$ in OL, the main cases are:

- If the last rule is a contraction-weakening rule, we use Lemma 10 to restrict ourselves to the $(cw_\lor)$ case, and by induction hypothesis we have $\vdash\ \Uparrow A_1 \lor A_2, A_1 \lor A_2$. The only way this is provable is by:

$$\dfrac{\dfrac{\vdash A_1 \lor A_2, A_1 \lor A_2 \Uparrow}{\vdash A_1 \lor A_2 \Uparrow A_1 \lor A_2}\,\mathrm{R}\Uparrow}{\vdash\ \Uparrow A_1 \lor A_2, A_1 \lor A_2}\,\Uparrow\mathrm{R}$$

  so that we can build:

$$\dfrac{\dfrac{\dfrac{\vdash A_1 \lor A_2, A_1 \lor A_2 \Uparrow}{\vdash A_1 \lor A_2, B \Uparrow}\,cw_1}{\vdash A_1 \lor A_2 \Uparrow B}\,\mathrm{R}\Uparrow}{\vdash\ \Uparrow A_1 \lor A_2, B}\,\Uparrow\mathrm{R}$$

- If the last rule is a $(\lor_1)$ rule, by induction hypothesis we have $\vdash\ \Uparrow A, C$, thus using Lemmas 15 and 16, Proposition 17 and Theorem 14:

$$\dfrac{\dfrac{\dfrac{\vdash\ \Uparrow A, C}{\vdash\ \Uparrow C, A} \quad \dfrac{\dfrac{A \text{ synchronous}}{\vdash A \Uparrow \neg A}}{\vdash A \lor B \Uparrow \neg A}}{\vdash A \lor B \Uparrow C}\,cut_0'}{\vdash\ \Uparrow A \lor B, C}\,\Uparrow\mathrm{R} \qquad \text{and} \qquad \dfrac{\dfrac{\dfrac{\vdash\ \Uparrow A, C}{\vdash\ \Uparrow C, A} \quad \dfrac{\dfrac{A \text{ asynchronous}}{\vdash \neg A \Uparrow A}}{\vdash A \lor B \Uparrow \neg A}}{\vdash A \lor B \Uparrow C}\,cut_0'}{\vdash\ \Uparrow A \lor B, C}\,\Uparrow\mathrm{R}$$

◀

As promised in Section 3, we can deduce cut elimination for OL.

▶ **Proposition 19** (Cut Elimination for OL)**.** *The cut rule is admissible in OL.*

Proof. By Theorem 18, we have $\vdash\ \Uparrow A, B$ and $\vdash\ \Uparrow \neg B, C$ in OL_f^0. By Lemma 15 we deduce $\vdash\ \Uparrow C, \neg B$. Using cut_0 (Theorem 14) we have $\vdash\ \Uparrow A, C$, and by Proposition 13, $\vdash A, C$ in OL. ◀

4.4 A Second Focused System OL_f

If we try to apply a simple bottom-up proof-search procedure in a sequent calculus system, a first obstacle to the finiteness of the search is given by cut rules. If a cut rule cannot be eliminated then a given conclusion leads us to a possibly infinite set of premises. A second obstacle comes from loops, *i.e.* non trivial derivations leading from a sequent to the same sequent (note however this obstacle can be dealt with by using loop detection during the search, but loops make the proof-search longer). All the systems we have seen so far contain non-trivial loops. Avoiding loops is one of the motivations for looking for a more constrained focused system. Let us analyse loops in $\mathsf{OL}_{\mathsf{f}}^{0}$. They mainly come from rules acting on sequents of the shape $\vdash \_, \_ \Uparrow$. If we look at derivations in a bottom-up way, we reach such a sequent through a (R$\Uparrow$) rule:

$$\frac{\vdash C, A \Uparrow}{\vdash C \Uparrow A}\;\mathrm{R}\Uparrow$$

then we stay with sequents $\vdash \_, \_ \Uparrow$ by using (upwardly):

$$\frac{\vdash C, C \Uparrow}{\vdash C, A \Uparrow}\;cw_1 \qquad \text{and} \qquad \frac{\vdash C, C \Uparrow}{\vdash A, C \Uparrow}\;cw_2$$

until we reach:

$$\frac{\vdash C \Downarrow A}{\vdash A, C \Uparrow}\;\mathrm{D}_1 \qquad \text{or} \qquad \frac{\vdash C \Downarrow A}{\vdash C, A \Uparrow}\;\mathrm{D}_2\;.$$

Globally, this means we start with a sequent $\vdash C \Uparrow A$ and we must end with $\vdash C \Downarrow A$, $\vdash A \Downarrow C$, $\vdash A \Downarrow A$ or $\vdash C \Downarrow C$. This would correspond to four derivable rules:

$$\frac{\vdash C \Downarrow A}{\vdash C \Uparrow A} \qquad \frac{\vdash A \Downarrow C}{\vdash C \Uparrow A} \qquad \frac{\vdash A \Downarrow A}{\vdash C \Uparrow A} \qquad \frac{\vdash C \Downarrow C}{\vdash C \Uparrow A}$$

In the same time we want to try to constrain contraction so that it is applied on $\vee$-formulas only (in the spirit of Lemma 10). Moreover we would like contraction not being applied twice on the same formula. In particular we get read of the fourth rule just above, which would allow C to be contracted (uselessly) many times. All these remarks lead us to the following new focused system called OL_{f}:

OL_{f}

$$\frac{\vdash\ \Uparrow A, C \qquad \vdash\ \Uparrow B, C}{\vdash\ \Uparrow A \wedge B, C}\;\Uparrow\!\wedge \qquad\qquad \frac{}{\vdash\ \Uparrow \top, C}\;\Uparrow\!\top$$

$$\frac{\vdash C \Uparrow A \qquad \vdash C \Uparrow B}{\vdash C \Uparrow A \wedge B}\;\wedge\!\Uparrow \qquad\qquad [(\mathrm{s})\text{ or }(\mathrm{n})]\;\frac{}{\vdash A \Uparrow \top}\;\top\!\Uparrow$$

$$\frac{\vdash B \vee C \Downarrow B \vee C}{\vdash\ \Uparrow B \vee C, A}\;\Uparrow\! cw \qquad\qquad [(\mathrm{s})\text{ or }(\mathrm{n})]\;\frac{\vdash B \vee C \Downarrow B \vee C}{\vdash A \Uparrow B \vee C}\;cw\!\Uparrow$$

$$\frac{}{\vdash \neg X \Downarrow X}\;ax_v \qquad\qquad \frac{\vdash C \Downarrow A}{\vdash C \Downarrow A \vee B}\;\vee_1 \qquad\qquad \frac{\vdash C \Downarrow B}{\vdash C \Downarrow A \vee B}\;\vee_2$$

$$\frac{\vdash A \Uparrow C}{\vdash\ \Uparrow A, C}\;\Uparrow\!\mathrm{R} \qquad [(\mathrm{a})]\;\frac{\vdash C \Uparrow A}{\vdash C \Downarrow A}\;\mathrm{R}\!\Downarrow \qquad [(\mathrm{s})]\;\frac{\vdash C \Downarrow A}{\vdash A \Uparrow C}\;\mathrm{D}_1 \qquad [(\mathrm{s})]\;\frac{\vdash C \Downarrow A}{\vdash C \Uparrow A}\;\mathrm{D}_2$$

(a) A is asynchronous (s) A is synchronous (n) A is the negation of a variable

Note, sequents $\vdash A, B \Uparrow$ disappear in this system which relies on three kinds of sequents only: $\vdash A \Uparrow B$, $\vdash A \Downarrow B$ and $\vdash\ \Uparrow A, B$.

▶ **Example 20.** We can prove in OL_{f} the sequent associated with Example 2:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \neg X \Downarrow X}\,ax_v}{\vdash X \Uparrow \neg X}\,\mathrm{D}_1}{\vdash X \Downarrow \neg X}\,\mathrm{R}{\Downarrow}}{\vdash X \Downarrow (X \wedge Y) \vee \neg X}\,\vee_2}{\vdash X \Downarrow ((X \wedge Y) \vee \neg X) \vee \neg Y}\,\vee_1}{\vdash ((X \wedge Y) \vee \neg X) \vee \neg Y \Uparrow X}\,\mathrm{D}_1 \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \neg Y \Downarrow Y}\,ax_v}{\vdash Y \Uparrow \neg Y}\,\mathrm{D}_1}{\vdash Y \Downarrow \neg Y}\,\mathrm{R}{\Downarrow}}{\vdash Y \Downarrow ((X \wedge Y) \vee \neg X) \vee \neg Y}\,\vee_2}{\vdash ((X \wedge Y) \vee \neg X) \vee \neg Y \Uparrow Y}\,\mathrm{D}_1}{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash ((X \wedge Y) \vee \neg X) \vee \neg Y \Uparrow X \wedge Y}{\vdash ((X \wedge Y) \vee \neg X) \vee \neg Y \Downarrow X \wedge Y}\,\mathrm{R}{\Downarrow}}{\vdash ((X \wedge Y) \vee \neg X) \vee \neg Y \Downarrow (X \wedge Y) \vee \neg X}\,\vee_1}{\vdash ((X \wedge Y) \vee \neg X) \vee \neg Y \Downarrow ((X \wedge Y) \vee \neg X) \vee \neg Y}\,\vee_1}{\vdash \bot \Uparrow ((X \wedge Y) \vee \neg X) \vee \neg Y}\,cw{\Uparrow}}{\vdash\ \Uparrow \bot, ((X \wedge Y) \vee \neg X) \vee \neg Y}\,{\Uparrow}\mathrm{R}}\,\wedge{\Uparrow}$$

The system $\mathsf{OL_f}$ is as expressive as OL^0_f for sequents $\vdash\ \Uparrow A, B$. In particular:

▶ **Proposition 21** (Expressiveness of $\mathsf{OL_f}$). *If $\vdash\ \Uparrow A, B$ is provable in OL^0_f, it is also provable in $\mathsf{OL_f}$.*

Proof. We prove by induction on the proof π in OL^0_f the more general statement:

- If $\vdash\ \Uparrow A, B$ in OL^0_f then $\vdash\ \Uparrow A, B$ in $\mathsf{OL_f}$.
- If $\vdash A \Uparrow B$ in OL^0_f then either $\vdash A \Uparrow B$ in $\mathsf{OL_f}$ or $A = A_1 \vee A_2$ with $\vdash A \Downarrow A$ in $\mathsf{OL_f}$.
- If $\vdash A \Downarrow B$ in OL^0_f then either $\vdash A \Downarrow B$ in $\mathsf{OL_f}$ or $A = A_1 \vee A_2$ with $\vdash A \Downarrow A$ in $\mathsf{OL_f}$.
- If $\vdash A, B \Uparrow$ in OL^0_f then at least one of the following four possibilities holds:
  - B is synchronous and $\vdash A \Downarrow B$ in $\mathsf{OL_f}$;
  - A is synchronous and $\vdash B \Downarrow A$ in $\mathsf{OL_f}$;
  - $A = A_1 \vee A_2$ and $\vdash A \Downarrow A$ in $\mathsf{OL_f}$;
  - $B = B_1 \vee B_2$ and $\vdash B \Downarrow B$ in $\mathsf{OL_f}$.

We consider each possible last rule for π. Interesting cases are:

- For the two contraction rules, we have $\vdash C, C \Uparrow$ in OL^0_f thus, by induction hypothesis, $\vdash C \Downarrow C$ in $\mathsf{OL_f}$ with C synchronous and we are done since $\vdash \bot \Downarrow \bot$ and $\vdash X \Downarrow X$ are not provable thus $C = C_1 \vee C_2$.
- For $(\vee_1)$, by induction hypothesis, we have either $\vdash C \Downarrow A$ or $\vdash C_1 \vee C_2 \Downarrow C_1 \vee C_2$ in $\mathsf{OL_f}$ with $C = C_1 \vee C_2$. In the first case, we apply the corresponding rule. In the second case, we are immediately done.
- For $({\Uparrow}\mathrm{R})$, by induction hypothesis we have $\vdash A \Uparrow C$ or $A = A_1 \vee A_2$ with $\vdash A_1 \vee A_2 \Downarrow A_1 \vee A_2$, we can build: $\dfrac{\vdash A \Uparrow C}{\vdash\ \Uparrow A, C}\,{\Uparrow}\mathrm{R}$ or $\dfrac{\vdash A_1 \vee A_2 \Downarrow A_1 \vee A_2}{\vdash\ \Uparrow A_1 \vee A_2, C}\,{\Uparrow}cw$
- For $(\mathrm{R}{\Uparrow})$, we apply the induction hypothesis and we obtain four possible cases:
  - If $\vdash C \Downarrow A$ in $\mathsf{OL_f}$ with A synchronous, we have: $\dfrac{\vdash C \Downarrow A}{\vdash C \Uparrow A}\,\mathrm{D}_2$
  - If $\vdash A \Downarrow C$ in $\mathsf{OL_f}$ with C synchronous, we have: $\dfrac{\vdash A \Downarrow C}{\vdash C \Uparrow A}\,\mathrm{D}_1$
  - If $\vdash C_1 \vee C_2 \Downarrow C_1 \vee C_2$ $(C = C_1 \vee C_2)$ in $\mathsf{OL_f}$, we are done.
  - If $\vdash A_1 \vee A_2 \Downarrow A_1 \vee A_2$ $(A = A_1 \vee A_2)$ in $\mathsf{OL_f}$, we have: $\dfrac{\vdash A_1 \vee A_2 \Downarrow A_1 \vee A_2}{\vdash C \Uparrow A_1 \vee A_2}\,cw{\Uparrow}$

◀

▶ **Proposition 22** (Soundness of $\mathsf{OL_f}$). *If $\vdash\ \Uparrow A, B$ is provable in $\mathsf{OL_f}$ then $\vdash A, B$ is provable in OL.*

From Propositions 19, 21 and 22, and Theorem 18, we can deduce the admissibility of the following cut rule in $\mathsf{OL_f}$: $\dfrac{\vdash\ \Uparrow A, B \qquad \vdash\ \Uparrow \neg B, C}{\vdash\ \Uparrow A, C}\ \mathit{cut}$

We have thus built yet another sound and complete system for orthologic. This one has very strong constraints on the structure of proofs. A key property of this new system (which holds in none of the previous ones) is the termination of the naive bottom-up proof search strategy (Proposition 23).

5 Proof Search in $\mathsf{OL_f}$

We first develop a few properties of $\mathsf{OL_f}$ on which we will rely for proof search. In a second time, we will compare with other algorithms from the literature.

5.1 Backward Proof Search

The basic idea of backward proof search in a cut-free sequent calculus system is to start from the sequent to be proved, to look in a bottom-up manner at each possible instance of a rule with this sequent as conclusion and to continue recursively with the premises of these instances until axioms are reached. Given a sequent, we are going to bound the length of branches of its proofs in $\mathsf{OL_f}$. Let us first define the following measure on formulas:

$$\varphi(X) = \varphi(\neg X) = \varphi(\bot) = \varphi(\top) = 1 \quad \varphi(A \wedge B) = \varphi(A) + \varphi(B) \quad \varphi(A \vee B) = 2\varphi(A) + 2\varphi(B)$$

As a bound on φ, we have $\varphi(A) < 2^{|A|}$ where $|A|$ is the size (number of symbols) of A.

▶ **Proposition 23** (Finiteness of Branches in $\mathsf{OL_f}$). *Given two formulas A and B, $2\varphi(A) + 2\varphi(B)$ is a bound on the length of the branches of any proof of $\vdash\ \Uparrow A, B$ in $\mathsf{OL_f}$.*

Proof. We define the measure ψ of a sequent, according to its shape:

$$\psi(\vdash\ \Uparrow A, B) = 2\varphi(A) + 2\varphi(B) \qquad\qquad \psi(\vdash A \Uparrow B) = \varphi(A) + 2\varphi(B)$$

$$\psi(\vdash A \Downarrow B) = \begin{cases} \varphi(A) + \varphi(B) & \text{if } B \text{ is synchronous} \\ \varphi(A) + 2\varphi(B) + 1 & \text{if } B \text{ is asynchronous} \end{cases}$$

We now prove for each rule of $\mathsf{OL_f}$: if S_1 is a sequent premise of the rule and S_2 is the sequent conclusion of the rule, then $\psi(S_1) < \psi(S_2)$. For example:

$(\vee_1)$ with A synchronous

$$\begin{aligned} \psi(\vdash C \Downarrow A) &= \varphi(C) + \varphi(A) \\ &< \varphi(C) + 2\varphi(A) + 2\varphi(B) \\ &= \varphi(C) + \varphi(A \vee B) \\ &= \psi(\vdash C \Downarrow A \vee B) \end{aligned}$$

$(\vee_1)$ with A asynchronous

$$\begin{aligned} \psi(\vdash C \Downarrow A) &= \varphi(C) + 2\varphi(A) + 1 \\ &< \varphi(C) + 2\varphi(A) + 2\varphi(B) \\ &= \varphi(C) + \varphi(A \vee B) \\ &= \psi(\vdash C \Downarrow A \vee B) \end{aligned}$$

Thus for any sequent S, $\psi(S)$ is a bound on the height of the branches of the proofs of S. ◀

Since rules of $\mathsf{OL_f}$ are finitely branching, this bound on the length of branches ensures (the absence of loops and) the termination of the backward proof search. Moreover, thanks to the sub-formula property, we know every sequent appearing in a proof of $\vdash\ \Uparrow A, B$ is made of two formulas which are sub-formulas of A or B. Since we have three different kinds of sequents, there are at most $3(|A| + |B|)^2$ such sequents. We have just proved a sequent cannot appear twice in a branch of a proof, so we can deduce a tighter bound than ψ on the height of branches: $3(|A| + |B|)^2$. We thus have an upper bound $2^{3(|A|+|B|)^2+1}$ on the size of proofs since rules have arity at most 2.

5.2 Single Formula Proof Search

As we have seen in Section 3, in systems with exactly two formulas in sequents presented in this paper, the provability of a formula A in orthologic is encoded as the provability of a sequent of the shape $\vdash A, A$ or $\vdash\ \Uparrow A, A$. Since we are often interested in the provability of a single formula, these sequents play a specific role. We can give some optimisation on the bottom structure of proofs of sequents $\vdash\ \Uparrow A, A$.

▶ **Proposition 24** (Diagonal Sequent). *The following properties hold in* $\mathsf{OL_f}$*:*

- $\vdash\ \Uparrow X, X$, $\vdash\ \Uparrow \neg X, \neg X$ *and* $\vdash\ \Uparrow \bot, \bot$ *are not provable.*
- $\vdash\ \Uparrow \top, \top$ *is provable.*
- $\vdash\ \Uparrow B \wedge C, B \wedge C$ *is provable if and only if both* $\vdash\ \Uparrow B, B$ *and* $\vdash\ \Uparrow C, C$ *are provable.*
- $\vdash\ \Uparrow B \vee C, B \vee C$ *is provable if and only if* $\vdash B \vee C \Downarrow B \vee C$ *is provable.*

Proof. We consider the last two statements only. For $\wedge$, we move back and forth to OL thanks to Theorem 18 and Propositions 21 and 22. In OL, we use Lemma 9 and:

$$\dfrac{\dfrac{\vdash B, B}{\vdash B, B \wedge C}\,cw \qquad \dfrac{\vdash C, C}{\vdash C, B \wedge C}\,cw}{\vdash B \wedge C, B \wedge C}\,\wedge$$

For $\vee$, the only possible last rules are:

$$\dfrac{\vdash B \vee C \Downarrow B \vee C}{\vdash\ \Uparrow B \vee C, B \vee C}\,{\Uparrow}cw \qquad\qquad \dfrac{\vdash B \vee C \Uparrow B \vee C}{\vdash\ \Uparrow B \vee C, B \vee C}\,{\Uparrow}\mathrm{R}$$

and for a proof of $\vdash B \vee C \Uparrow B \vee C$, the only possible last rules are:

$$\dfrac{\vdash B \vee C \Downarrow B \vee C}{\vdash B \vee C \Uparrow B \vee C}\,cw{\Uparrow} \qquad \dfrac{\vdash B \vee C \Downarrow B \vee C}{\vdash B \vee C \Uparrow B \vee C}\,\mathrm{D}_1 \qquad \dfrac{\vdash B \vee C \Downarrow B \vee C}{\vdash B \vee C \Uparrow B \vee C}\,\mathrm{D}_2$$

so that $\vdash B \vee C \Downarrow B \vee C$ must be provable for $\vdash\ \Uparrow B \vee C, B \vee C$ to be provable. In the other direction we directly use (${\Uparrow}cw$). ◀

This means in particular that any sequent $\vdash A, A$ is equivalent to a finite family of sequents $\vdash B_1 \vee C_1 \Downarrow B_1 \vee C_1, \dots,$ $\vdash B_n \vee C_n \Downarrow B_n \vee C_n$ (with each $B_i \vee C_i$ sub-formula of A) or clearly not provable.

5.3 Forward Proof Search

Forward proof search consists in building, in a top-down way, proof-trees which are candidates to be sub-proof-trees of proofs of a given sequent. Clearly the sub-formula property can be used to control the sequents to be considered inside the proof-trees. We use here even stronger constraints. Let us fix a formula A. We want to study sub-proof-trees of proofs of $\vdash\ \Uparrow A, A$ in $\mathsf{OL_f}$. We do not consider the more general case $\vdash\ \Uparrow A, B$ here.

▶ **Proposition 25** (Strengthened Sub-Formula Property). *If* $\vdash C \Downarrow B$ *or* $\vdash C \Uparrow B'$ *or* $\vdash\ \Uparrow D, E$ *appears in a proof of* $\vdash\ \Uparrow A, A$ *in* $\mathsf{OL_f}$*, these are sub-formulas of A and moreover:*

- *if B is asynchronous, it appears inside A just below a $\vee$ connective;*
- *if B' is synchronous, it is equal to A or it appears inside A just below a $\wedge$ connective;*
- *if $B' = A$ then $C = A$ or C appears inside A below $\wedge$ connectives only;*
- *if C is synchronous, it is equal to A or it appears inside A just below a $\wedge$ connective;*
- *$E = A$, and $D = A$ or D appears inside A below $\wedge$ connectives only.*

Proof. Since $\vdash\ \Uparrow A, A$ satisfies the conclusion of the statement, we prove for each rule that if the conclusion satisfies it, then all its premises as well. For example, for the $(\mathrm{R}\Downarrow)$ rule, the formula in position B' in the premise must be asynchronous. Moreover we cannot have $B' = A$, since the property for the conclusion gives B' below a $\vee$ connective inside A. ◀

This proposition provides us constraints on the meaningful sequents to be considered during forward proof search. This means we can restrict the application of rules in the algorithm for forward proof search to the case where they generate sequents satisfying the properties given by Proposition 25.

5.4 Benchmark

We want to compare our proof-search procedures with procedures from the literature. We consider some formulas from [12] and [5] as well as some random formulas in the language of orthologic:

$$\begin{aligned}
\mathtt{E}_1 &= ((\neg X \vee Y) \wedge X) \ \vee\ ((X \wedge \neg Y) \ \vee\ ((\neg X \wedge ((X \vee \neg Y) \wedge (X \vee Y))) \\
&\qquad\qquad\qquad \vee\ (\neg X \wedge ((\neg X \wedge Y) \vee (\neg X \wedge \neg Y)))))\\
\mathtt{E}_2 &= X \ \vee\ ((\neg X \wedge ((X \vee \neg Y) \wedge (X \vee Y))) \ \vee\ (\neg X \wedge ((\neg X \wedge Y) \vee (\neg X \wedge \neg Y))))\\
\mathtt{E}_3 &= (((X \vee \neg Y) \wedge (X \vee Y)) \wedge (\neg X \vee (X \wedge \neg Y))) \ \vee\ (\neg X \vee Y)\\
\Phi_0 &= X \vee \neg X \qquad \Phi_{n+1} = ((X_n \wedge Y_n) \wedge (X_n \wedge Z_n)) \vee (((\neg X_n \wedge \Phi_n) \vee \neg Y_n) \vee \neg Z_n)\\
\Psi_0^1 &= \top \qquad \Psi_0^2 = \bot \qquad \Psi_{n+1}^1 = \Psi_n^1 \wedge X_n \qquad \Psi_{n+1}^2 = \Psi_n^2 \vee Y_n\\
\Psi_n^3 &= (X \vee (Y \wedge \Psi_n^2)) \wedge \Psi_n^1 \qquad \Psi_n^4 = (Y \wedge (X \vee \Psi_n^1)) \vee \Psi_n^2 \qquad \Psi_n = \neg\Psi_n^3 \vee \Psi_n^4
\end{aligned}$$

The formulas $\mathtt{E}_2$, $\mathtt{E}_3$ and Φ_n are provable, while $\mathtt{E}_1$ and Ψ_n are not.

We compare four algorithms: cf is prove-cf from [5], fw is the forward algorithm from [5], bw_f is the backward algorithm based on OL_f, and fw_f is the forward algorithm based on OL_f. The implementations are done in `OCaml` in the most naive way (except that we use some memoization), so that running time (*time*, in seconds) should not be taken too seriously. As an alternative measure which depends less on the particular implementation, we also count the number of rule occurrences (*rules*) applied during search.

time	cf	bw_f	fw	fw_f
$\mathtt{E}_1$	0.00	0.00	0.04	0.03
$\mathtt{E}_2$	0.00	0.00	0.02	0.01
$\mathtt{E}_3$	0.00	0.00	0.02	0.02
Φ_5	0.07	0.00	15.00	3.56
Φ_{10}	0.34	0.00	368.86	88.60
Ψ_5	0.22	0.00	1.43	0.13
Ψ_{20}	_	0.00	161.84	4.92

rules	cf	bw_f	fw	fw_f
$\mathtt{E}_1$	2 305	132	47	64
$\mathtt{E}_2$	210	104	33	49
$\mathtt{E}_3$	42	144	47	49
Φ_5	6 094	384	724	338
Φ_{10}	12 344	774	2 639	1 023
Ψ_5	244 055	308	343	123
Ψ_{20}	_	2 603	2 083	723

rules	cf	bw_f	fw	fw_f
Average on some random formulas of size 20	971	28	163	29
Average on some random formulas of size 100	129 428	264	2 753	734

This is really a minimalist benchmarking. Deeper experiments must be done to obtain more precise comparison informations. Focusing-based algorithms look really competitive. Forward algorithms are sometimes more efficient than backward ones concerning the number of rules applied, but require more management of data structures so take longer execution time.

6 Conclusion

We have presented new sequent-calculus proof-systems for orthologic, mainly: OL which is the simplest such system we know, and OL_f which is based on focusing to constrain the structure of proofs. With some complementary analysis on the structure of proofs in OL_f we have proposed efficient proof search algorithms for orthologic which look quicker than the state of the art [5] (but additional studies in this direction must be done to obtain fully convincing evaluations).

Our new systems open the door for additional proof-theoretical studies of orthologic (and the possibility of extracting counter-models from proof-search failures should be investigated). We hope also this will lead to results in the theory of ortholattices (free ones in particular) in the spirit of Whitman's work [15]. The present work could be extended to second-order quantifiers on the logic side in relation with complete ortholattices. We plan also to work on the application of focusing to other lattice-related logics [14].

Finally, the proof theory of orthologic seems to be mature enough to try to develop some Curry-Howard correspondence aiming at exhibiting the computational content of orthologic.

Additional Material

A `Coq` development formalising the main proofs of the paper is available at:

`https://hal.archives-ouvertes.fr/hal-01306132/file/olf.v.txt`

The `OCaml` code for the benchmark of Section 5.4 is available at:

`https://hal.archives-ouvertes.fr/hal-01306132/file/olf.ml.txt`

References

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

2 Garrett Birkhoff. *Lattice Theory*, volume 25 of *Colloquium Publications*. American Mathematical Society, third edition, 1967.

3 Günter Bruns. Free ortholattices. *Canadian Journal of Mathematics*, 28(5):977–985, October 1976.

4 Uwe Egly and Hans Tompits. Gentzen-like methods in quantum logic. Technical Report 99-1, Institute for Programming and Logics, University at Albany - SUNY, 1999. Position Papers of TABLEAUX '99.

5 Uwe Egly and Hans Tompits. On different proof-search strategies for orthologic. *Studia Logica*, 73(1):131–152, February 2003.

6 Claudia Faggian and Giovanni Sambin. From basic logic to quantum logics with cut-elimination. *International Journal of Theoretical Physics*, 37(1):31–37, January 1998.

7 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

8 Jean-Yves Girard. A new constructive logic: classical logic. *Mathematical Structures in Computer Science*, 1(3):255–296, 1991.

9 Robert Goldblatt. Semantic analysis of orthologic. *Journal of Philosophical Logic*, 3(1–2):19–35, 1974.

10 Robert Goldblatt. Orthomodularity is not elementary. *Journal of Symbolic Logic*, 49(2):401–404, 1984.

11 Olivier Laurent. A proof of the focalization property of linear logic. Unpublished note, May 2004.

12 William McCune. Automatic proofs and counterexamples for some ortholattice identities. *Information Processing Letters*, 65(6):285–291, 1998.

13 Hirokazu Nishimura. Proof theory for minimal quantum logic I. *International Journal of Theoretical Physics*, 33(1):103–113, January 1994.

14 Jürgen Schulte Mönting. Cut elimination and word problems for varieties of lattices. *Algebra Universalis*, 12:290–321, December 1981.

15 Philip Whitman. Free lattices. *Annals of Mathematics*, 42(1):325–330, January 1941.

Functions-as-Constructors Higher-Order Unification

Tomer Libal[1] **and Dale Miller**[2]

1 Inria Saclay & LIX/École Polytechnique, Palaiseau, France
2 Inria Saclay & LIX/École Polytechnique, Palaiseau, France

Abstract

Unification is a central operation in the construction of a range of computational logic systems based on first-order and higher-order logics. First-order unification has a number of properties that dominates the way it is incorporated within such systems. In particular, first-order unification is decidable, unary, and can be performed on untyped term structures. None of these three properties hold for full higher-order unification: unification is undecidable, unifiers can be incomparable, and term-level typing can dominate the search for unifiers. The so-called *pattern* subset of higher-order unification was designed to be a small extension to first-order unification that respected the basic laws governing λ-binding (the equalities of α, β, and η-conversion) but which also satisfied those three properties. While the pattern fragment of higher-order unification has been popular in various implemented systems and in various theoretical considerations, it is too weak for a number of applications. In this paper, we define an extension of pattern unification that is motivated by some existing applications and which satisfies these three properties. The main idea behind this extension is that the arguments to a higher-order, free variable can be more than just distinct bound variables: they can also be terms constructed from (sufficient numbers of) such variables using term constructors and where no argument is a subterm of any other argument. We show that this extension to pattern unification satisfies the three properties mentioned above.

1998 ACM Subject Classification F.4.1 [Mathematical Logic] Mechanical theorem proving

Keywords and phrases higher-order unification, pattern unification

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.26

1 Introduction

Unification is the process of solving equality constraints by the computation of substitutions. This process is used in computational logic systems ranging from automated theorem provers, proof assistants, type inference systems, and logic programming. The first-order unification – that is, unification restricted to first-order terms – enjoys at least three important computational properties, namely, (1) decidability, (2) determinacy, and (3) type-freeness. These properties of unification shaped the way it can be used within computational logic systems. The first two of these properties ensures that unification – as a process – will either fail to find a unifier for a given set of disagreement pairs or will succeed and return the *most general unifier* that solves all those disagreement pairs. The notion of type-freeness simply means that unification can be done independently of the possible typing discipline that might be employed with terms. Thus, first-order unification can be performed on untyped first-order terms (as terms are usually considered in, say, Prolog). This property is important since it means that unification can be used with any typing discipline that might be adopted. Since typing is usually an open-ended design issue in many languages (consider,

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: D. Kesner and B. Pientka; Article No. 26; pp. 26:1–26:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for example, higher-order types, subtypes, dependent types, parametric types, linear types, etc.), the type-freeness of unification makes it possible for it to be applied to a range of typing disciplines.

Of course, many syntactic objects are not most naturally considered as purely first-order terms: this is the case when that syntax contains bindings. Instead, many systems have adopted the approach used by Church in his Simple Theory of Types [11] where terms and term equality comes directly from the λ-calculus. All binding operations – quantification in first-order formulas, function arguments in functional programs, local variables, etc. – can be represented using the sole binder of the λ-calculus. Early papers showed that second-order pattern matching could be used to support interesting program analysis and program transformation [18] and that a higher-order version of Prolog could be used to do more general manipulations of programs and formulas [21]. Today, there is a rich collection of computational logic systems that have moved beyond first-order term unification and rely on some form of higher-order unification. These include the theorem provers TPS [3], Leo [7] and Satallax [10]; the proof assistants Isabelle [26], Coq [36], Matita [4], Minlog [31], Agda [9], Abella [5], and Beluga [29]; the logic programming languages λProlog [23] and Twelf [28]; and various application domains such as natural language processing [12].

The integration of full higher-order unification into computational logic systems is not as simple as it is in first-order systems since the three properties mentioned above do not hold. The unification of simply typed λ-terms is undecidable [15, 16] and there can be incomparable unifiers, implying that no most general unifiers exist in the general situation. Also, types matter a great deal in determining the search space of unifiers. For example, let i and j be primitive types, let a be a constant of type i, and let F and X be variables of type $\alpha \to i$ and α, respectively, where α is a type variable. Consider the unification problem $(F\ X) = a$. If we set α to j, then there is an mgu for this problem, namely $[F \mapsto \lambda w.a]$. If we set α to i, then there are two incomparable solutions, namely $[F \mapsto \lambda w.a]$ and $[F \mapsto \lambda w.w,\ X \mapsto a]$. If we set α to $i \to i$, then there is an infinite number of incomparable solutions: $[F \mapsto \lambda f.a]$ and, for each natural number n, $[F \mapsto \lambda f.f^n a,\ X \mapsto \lambda w.w]$. If higher order values for α are considered, the possibility of unifiers becomes dizzying.

For these reasons, the integration of unification for simply typed λ-terms into computational logic systems is complex: most such integration efforts attempt to accommodate the (pre-)unification search procedure of Huet [17].

Instead of moving from first-order unification to full higher-order unification, it is possible to move to an intermediate space of unification problems. Given that higher-order unification is undecidable, there is an infinite number of decidable classes that one could consider. The setting of *higher-order pattern unification* (proposed in [20] and called L_λ-unification there) could be seen as the weakest extension of first-order unification in which the equations of α, β, and η conversion hold. In this fragment, a free variable cannot be applied to general terms but only to bound variables that cannot appear free in eventual instantiations for the free variable. This restriction means that all forms of β-reduction encountered during unification are actually (α-equivalent) to the rule $(\lambda x.B)x = B$ (a conversion rule called β_0). Notice that in this setting, β-reduction reduces the size of terms: hence, unification takes on a much simpler nature. The unification problems that result retain all three properties we listed for first-order unification [20]. As a result, the integration of pattern unification into a prover is usually much simpler than incorporating all the search behavior implied by Huet's (pre-)unification procedure.

A somewhat surprising fact about pattern unification is that many computational logic systems actually need only this subset of higher-order unification in order to be "practically complete”: that is, restricting unification to just this subset did not stop the bulk of specifications from being properly executed. For example, while both early implementations of λProlog and LF [24, 28] implemented full higher-order unification, the most recent versions of those languages implement only pattern unification [1, 30]. A design feature of both of those systems is to treat any unification problem that is not in the pattern fragment as a suspended constraint: usually, subsequent substitutions will cause such delayed problems to convert into the pattern fragment. Processing of constraints may also be possible: the application of *pruning* to flexible-flexible constraints in [22] is such an example. Also since pattern unification does not require typing information, it has been possible to describe variants of such unification in settings where types can play a role during unification: see for example, generalizations of pattern unification for dependent and polymorphic types [27], product types [13, 14], and sum types [2].

Since pattern unification is a weak fragment of higher-order unification, it is natural to ask if it can be extended and still keep the same high-level properties. There have been extensions of pattern unification considered in the literature. The generalization (mentioned above) of pattern unification to patterns by Fettig and Löchner [14] and Duggan [13] allows for constructors denoting projections to be admitted in the scope of free functional variables. These projections are specific unary functions which are closed under a number of properties, such as associativity. When attempting to encode the meta-theory of sequent calculus in which eigenvariables are seen as abstractions over sequents [19], a single bound variable was intended to be used as a list of bound (eigen)variables. Thus, in order to encode the sequent judgment $x_0, \ldots, x_n \vdash C x_0 \ldots x_n$ (for $n \geq 0$ and all variables being of the same primitive type) one would instead use the simply typed term $\lambda l.C(\text{fst } l) \ldots (\text{fst}(\text{snd}^n l))$, where the environment abstraction l has type, say, evs, and fst and snd are constructors of type evs $\to i$ and evs $\to$ evs, respectively. Tiu showed how to lift pattern unification to this setting [33]. The Coq proof assistant allows for some limited forms of unification and many simple unification problems can appear that should be automatically solved. A typical such example is of the form $\lambda x.Y(gx) \doteq \lambda x.f(gx)$, where Y is a free variable of type $i \to i$ and f and g are constructors of the type $i \to i$. Clearly, this problem has the mgu $Y \mapsto \lambda z.fz$ but it falls outside the pattern restriction. There are certain uses of Coq (for example, with the `bigop` library of SSReflect) which produce a number of non-pattern unification problems.[1]

Let us return to the definition of pattern unification problems. The restriction on occurrences of the free variable, say, M is that (1) it can be applied only to variables that cannot appear free in terms that are used to instantiate M and (2) that those arguments are distinct. Condition (1) essentially says that the arguments of M form a primitive pattern that allows one to form an abstraction to solve a unification problem. Thus, $M\ x\ y$ can equal, say, $(s\ x) + y$ simply by forming the abstraction $\lambda x \lambda y.(s\ x) + y$. Condition (2) implies that such abstracts are unique.

The examples of needing richer unification problems above illustrate that it is also natural to consider arguments built using variables *and* term constructors: that is, we should consider generalizing condition (1) above by allowing the application $\lambda l(M\ (\text{fst } l)\ (\text{fst } (\text{snd } l)))$. If this application is required to unify with a term of the form $\lambda l.t$ then all occurrence of l in t must occur in subterms of the form $(\text{fst } l)$ or $(\text{fst } (\text{snd } l))$. In that case, forming an abstraction of t by replacing all occurrences of $(\text{fst } l)$ and $(\text{fst } (\text{snd } l))$ with separate bound variables gives a solution to this unification problem. To guarantee uniqueness of such solutions, we shall also generalize condition (2) so that the arguments of M cannot be subterms of each

[1] Personal communication with Enrico Tassi.

other. This additional constraint is required here (but not in the papers by Duggan [13] and Tiu [33]) since we wish to handle richer signatures than just those with monadic constraints.

Many of the examples leading to this generalization of pattern unification arise in situations where operators (such as fst and snd) are really functions and *not* constructors: the intended meaning of those two operators are as functions that map lists to either their first element or to their tail. When they arise in unification problems, however, we can only expect to treat them as constructors. Thus, we shall name this extended pattern unification as *function-as-constructor* (pattern) unification, or just FCU for short.

The rest of this paper is structured as follows. We cover the basic concepts related to higher-order unification in Section 2. The class of unification problems addressed in this paper, the *functions-as-constructor* class, is defined in Section 3 as is a unification algorithm for that class. We prove the correctness of that algorithm in Section 4. We conclude in Section 5.

2 Preliminaries

2.1 The Lambda-Calculus

In this section we will present the logical language that will be used throughout the paper. The language is a version of Church's simple theory of types [11] with an η-conversion rule as presented in [6] and [32] and with implicit α-conversions. Unless stated otherwise, all terms are implicitly converted into β-normal and η-expanded form. Most of the definitions in this section are adapted from [32].

Let $\mathfrak{T}_0$ be a set of basic types, then the set of types $\mathfrak{T}$ is generated by $\mathfrak{T} := \mathfrak{T}_0 \mid \mathfrak{T} \to \mathfrak{T}$. Let $\mathfrak{C}$ be a signature of function symbols and let $\mathfrak{V}$ be a countably infinite set of variable symbols. *Variables* are normally denoted by the letters $l, x, y, w, z, X, Y, W, Z$ and *function symbols* by the letters f, g, h, k, a or typed names like `cons` . W e sometimes use subscripts and superscripts as well. We sometimes add a superscript to symbols in order to specify their type. The set $\texttt{Term}^{\alpha}$ of terms of type α is generated by $\texttt{Term}^{\alpha} := f^{\alpha} \mid x^{\alpha} \mid (\lambda x^{\beta}.\texttt{Term}^{\gamma}) \mid (\texttt{Term}^{\beta\to\alpha}\texttt{Term}^{\beta})$ where $f \in \mathfrak{C}, x \in \mathfrak{V}$ and $\alpha \in \mathfrak{T}$ (in the abstraction, $\alpha = \beta \to \gamma$). Applications throughout the paper will be associated to the left. We will sometimes omit brackets when the meaning is clear. We will also normally omit typing information when it is not crucial for the correctness of the results. $\tau(t^{\alpha}) = \alpha$ refers to the type of a term. The set `Term` denotes the set of all terms. *Subterms* and *positions* are defined as usual. We denote the fact that t is a (strict) subterm of s using the infix binary symbol ($\sqsubset$) $\sqsubseteq$. *Sizes of positions* denote the length of the path to the position. We denote the subterm of t at position p by $t|_p$. *Bound* and *free variables* are defined as usual. We will use the convention of denoting bound and universally quantified variables by lower letters while existentially quantified variables will be denoted by capital letters. Given a term t, we denote by $\texttt{hd}(t)$ its *head symbol* and distinguish between *flex* terms, whose head is a free variable and *rigid* terms, whose head is a function symbol or a bound variable.

We will use both *set union* ($\cup$) and *disjoint set union* ($\uplus$) in the text.

Substitutions and their *composition* ($\circ$) are defined as usual. Namely, $(\sigma \circ \theta)X = \theta(\sigma X)$. `id` denotes the trivial substitution mapping each variable to itself. We denote by $\sigma|_W$ the substitution obtained from substitution σ by restricting its domain to variables in W. We denote by $\sigma[X \mapsto t]$ the substitution obtained from σ by mapping X to t, where X might already exist in the domain of σ. We extend the application of substitutions to terms in the usual way and denote it by postfix notation. Variable capture is avoided by implicitly renaming variables to fresh names upon binding. A substitution σ is *more general* than a substitution θ, denoted $\sigma \leq \theta$, if there is a substitution δ such that $\sigma \circ \delta = \theta$. The *domain* of a substitution σ is denoted by $\texttt{dom}(\sigma)$.

We introduce also a vector notation $\overline{t_n}$ for the sequence of terms $t_1, \ldots, t_n$. This notation also holds for nesting of sequences. For example, the term $f\ (X_1\ z_1\ z_2)\ (X_2\ z_1\ z_2)\ (X_3\ z_1\ z_2)$ will be denoted by $f\overline{X_3\overline{z_2}}$. The meaning of the notation $\lambda\overline{z_n}$ is $\lambda z_1, \ldots, \lambda z_n$. When the order of the sequence is not important, we will use this notation also for multisets.

2.2 Higher-order Pre-unification

In this section we present Huet's pre-unification procedure [17] as defined in [32]. The procedure will be proven, in Section 3.2, to be deterministic for the class of FCU problems. This result, together with the completeness of the procedure, implies the existence of most-general unifiers for unifiable problems of this class.

The presentation in this paper of both the pattern and FCU unification algorithms is much simplified if the following non-standard normal form is being used. All terms, including functional existential variables but excluding the arguments of these variables, are considered to be in η-expanded form. The arguments of these variables are expected to be in η-normal forms. In a similar manner to the one in [32], one can prove that all substitutions used in this paper preserve this normal form.

▶ **Definition 1** (Unification Problem). An equation is a formula $t \doteq s$ where t and s are $\beta\eta$-normalized (see remark above) terms. A unification problem is a formula of the form $\exists\overline{X_m}.e_1 \wedge \ldots \wedge e_n$ where e_i for $0 < i \leq n$ is an equation. Sets of equations are always closed under symmetry, i.e. if $t \doteq s$ is in the set, then also $s \doteq t$.

▶ **Definition 2** (Unification System). A unification system over a signature $\mathfrak{C}$ is the following quadruple $\langle Q_\exists, Q_\forall, S, \sigma\rangle$ where $Q_\exists$ and $Q_\forall$ are disjoint sets of variables, S is a set of equations and σ a substitution. Given a unification problem $\exists\overline{X_m}.e_1 \wedge \ldots \wedge e_n$ we consider the unification system over signature $\mathfrak{C}$ by setting $Q_\exists = \overline{X_m}$, $Q_\forall = \{\}$, $S = \{e_1, \ldots, e_n\}$ and $\sigma = \texttt{id}$. Let $\texttt{bvars}(e_i) = \overline{z_n}$ for $e_i = (\lambda\overline{z_n}.t_i \doteq \lambda\overline{z_n}.s_i)$.

An important property of terms in which we will be interested later is the subterm property between terms of different equations. Such a property is not closed under α-renaming of bound variables for our current definition of unification systems. Since we implicitly assume such renaming in order to avoid variables capture, we have to add the following additional requirement.

▶ **Definition 3** (Regular Unification Systems). A regular unification system is a unification system in which each bound variable which is bound in a different context has a different name.

▶ **Example 4.** The system $\langle\emptyset, \emptyset, \{\lambda x.fx \doteq \lambda x.gx, \lambda y.hy \doteq \lambda y.ky\}, id\rangle$ is regular while the system $\langle\emptyset, \emptyset, \{\lambda x.fx \doteq \lambda x.gx, \lambda x.hx \doteq \lambda x.kx\}, id\rangle$ is not.

The next lemma is easily proven.

▶ **Lemma 5.** *The subterm property is closed under α-renaming of bound variables for regular unification systems.*

Regular unification systems will also be called systems.

Before presenting Huet's procedure for pre-unification, we will repeat the definition of partial bindings as given in [32].

$$\begin{aligned}
\langle Q_\exists, Q_\forall, S \uplus \{t \doteq t\}, \sigma\rangle &\rightarrow \langle Q_\exists, Q_\forall, S, \sigma\rangle && \texttt{(Delete)}\\
\langle Q_\exists, Q_\forall, S \uplus \{\lambda\overline{z_k}.f\overline{t_n} \doteq \lambda\overline{z_k}.f\overline{s_n}\}, \sigma\rangle &\rightarrow \langle Q_\exists, Q_\forall, S \uplus \{\lambda\overline{z_k}.t_1 \doteq \lambda\overline{z_k}.s_1, \ldots, \lambda\overline{z_k}.t_n \doteq \lambda\overline{z_k}.s_n\}, \sigma\rangle && \texttt{(Decomp)}\\
\langle Q_\exists, Q_\forall, S \uplus \{\lambda\overline{z_k}.X\overline{z_k} \doteq \lambda\overline{z_k}.t\}, \sigma\rangle &\rightarrow \langle Q_\exists, Q_\forall, S\theta \uplus \{X \doteq \lambda\overline{z_k}.t\}, \sigma \circ \theta\rangle && \texttt{(Bind)}\\
&\text{where } X \notin \texttt{fvars}(t) \text{ and } \theta = [X \mapsto \lambda\overline{z_k}.t]\\
\langle Q_\exists, Q_\forall, S \uplus \{\lambda\overline{z_k}.X^\alpha(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{t_m})\}, \sigma\rangle &\rightarrow \langle Q_\exists, Q_\forall, S \uplus \{X \doteq u, \lambda\overline{z_k}.X^\alpha(\overline{s_n}) \doteq \lambda\overline{z_k}.f(\overline{t_m})\}, \sigma\rangle && \texttt{(Imitate)}\\
&\text{where } u = \texttt{PB}(f, \alpha) \text{ and } f \in \mathfrak{C}\\
\langle Q_\exists, Q_\forall, S \uplus \{\lambda\overline{z_k}.X^\alpha(\overline{s_n}) \doteq \lambda\overline{z_k}.a(\overline{t_m})\}, \sigma\rangle &\rightarrow \langle Q_\exists, Q_\forall, S \uplus \{X \doteq u, \lambda\overline{z_k}.X^\alpha(\overline{s_n}) \doteq \lambda\overline{z_k}.a(\overline{t_m})\}, \sigma\rangle && \texttt{(Project)}\\
&\text{where } 0 < i \leq k, u = \texttt{PB}(i, \alpha) \text{ and either } a \in \mathfrak{C} \text{ or } a = z_i \text{ for some } 0 < i \leq k
\end{aligned}$$

Figure 1 `PUA`- Huet's pre-unification procedure.

▶ **Definition 6** (Partial bindings). A partial binding of type $\alpha_1 \rightarrow \ldots \rightarrow \alpha_n \rightarrow \beta$ where $\beta \in \mathfrak{T}_o$ is a term of the form
$\lambda\overline{y_n}.a(\lambda\overline{z^1_{p_1}}.X_1(\overline{y_n}, \overline{z^1_{p_1}}), \ldots, \lambda\overline{z^m_{p_m}}.X_m(\overline{y_n}, \overline{z^m_{p_m}}))$ for some atom a where

- $\tau(y_i) = \alpha_i$ for $0 < i \leq n$.
- $\tau(a) = \gamma_1 \rightarrow \ldots \rightarrow \gamma_m \rightarrow \beta$ where $\gamma_i = \delta^i_1 \rightarrow \ldots \rightarrow \delta^i_{p_i} \rightarrow \gamma'_i$ for $0 < i \leq m$.
- $\gamma'_1, \ldots, \gamma'_m \in \mathfrak{T}_o$.
- $\tau(z^i_j) = \delta^i_j$ for $0 < i \leq m$ and $0 < j \leq p_i$.
- X_i is a fresh variable and $\tau(X_i) = \alpha_1 \rightarrow \ldots \rightarrow \alpha_n \rightarrow \delta^i_1 \rightarrow \ldots \rightarrow \delta^i_{p_i} \rightarrow \gamma'_i$ for $0 < i \leq m$.

Partial bindings fall into two categories, imitation bindings, which for a given atom a and type α, are denoted by $\texttt{PB}(a, \alpha)$ and projection bindings, which for a given index $0 < i \leq n$ and a type α, are denoted by $\texttt{PB}(i, \alpha)$ and in which the atom a is equal to the bound variable y_i. Since partial bindings are uniquely determined by an index, a type and an atom (up to renaming of the fresh variables $\overline{X_m}$), this defines a particular term.

▶ **Definition 7** (Huet's Pre-unification Procedure). Huet's pre-unification procedure is given in Figure 1. Note that the sets $Q_\exists$ and $Q_\forall$ are fixed during the execution and are mentioned explicitly just for compatibility with the algorithms given later in the paper.

The next theorem states the completeness of this procedure.

▶ **Theorem 8** ([32]). *Given a system $\langle Q_\exists, Q_\forall, S, \texttt{id}\rangle$ and assuming it is unifiable by σ, then there is a sequence of rule applications in Def. 7 resulting in $\langle Q'_\exists, Q'_\forall, \emptyset, \theta\rangle$ such that $\theta \leq \sigma$.*

2.3 Pattern Unification

In this section we describe the higher-order pattern unification algorithm in [20]. The notation used is similar to the one in [25]. This algorithm forms the basis for our algorithm.

▶ **Definition 9** (Pattern Systems). A system $\langle Q_\exists, Q_\forall, S, \sigma\rangle$ is called a pattern system if for all equations $e_i \in S$ and for all subterms $X\overline{z_n}$ in these equations such that $X \in Q_\exists$ we have that $\overline{z_n} \subseteq Q_\forall \cup \texttt{bvars}(e_i)$ and $z_i \neq z_j$ for all $0 < i < j \leq n$.

The following simplification will be called during the execution of the algorithm given in Def. 11.

▶ **Definition 10** (Pruning). Given a pattern system $\langle Q_\exists, Q_\forall, S, \sigma\rangle$ such that $\lambda\overline{z_n}.X\overline{z^1_n} \doteq \lambda\overline{z_n}.r \in S$, r contains an occurrence y such that $y \in Q_\forall \cup \overline{z_n}$ and $y \notin \overline{z^1_n}$:

- if there is a subterm $W\overline{z^2_m}$ of r such that $y = z^2_i$ for some $0 < i \leq m$, then return $\langle Q_\exists \uplus \{W'\} \setminus \{W\}, Q_\forall, S\theta, \sigma \circ \theta\rangle$ where $\theta = [W \mapsto \lambda\overline{z^2_m}.W'\overline{z^3_{m-1}}]$ and $\overline{z^3_{m-1}} = \overline{z^2_m} \setminus \{z^2_i\}$.
- otherwise, return $\langle Q_\exists, Q_\forall, \bot, \texttt{id}\rangle$.

$$\begin{array}{rcll}
\langle Q_\exists, Q_\forall, S \uplus \{t \doteq t\}, \sigma\rangle & \rightarrow & \langle Q_\exists, Q_\forall, S\}, \sigma\rangle & (0)\\
\langle Q_\exists, Q_\forall, S \uplus \{\lambda x.s \doteq \lambda x.t\}, \sigma\rangle & \rightarrow & \langle Q_\exists, Q_\forall \uplus \{x\}, S \uplus \{s \doteq t\}, \sigma\rangle & (1)\\
\langle Q_\exists, Q_\forall, S \uplus \{f\overline{t_n} \doteq f\overline{s_n}\}, \sigma\rangle & \rightarrow & \langle Q_\exists, Q_\forall, S \uplus \{t_1 \doteq s_1, \ldots, t_n \doteq s_n\}, \sigma\rangle & (2)\\
\multicolumn{4}{l}{\qquad\text{where } f \in \mathfrak{C} \cup Q_\forall}\\
\langle Q_\exists \uplus \{X\}, Q_\forall, S \uplus \{X\overline{z_n} \doteq f\overline{s_m}\}, \sigma\rangle & \rightarrow & \langle Q_\exists, Q_\forall, S\theta, \sigma \circ \theta\rangle & (3)\\
\multicolumn{4}{l}{\qquad\text{where } f \in \mathfrak{C},\ X \notin \mathtt{fvars}(f\overline{s_m}) \text{ and } \theta = [X \mapsto \lambda\overline{z_n}.f\overline{s_m}]}\\
\langle Q_\exists \uplus \{X\}, Q_\forall, S \uplus \{X\overline{z_n^1} \doteq X\overline{z_n^2}\}, \sigma\rangle & \rightarrow & \langle Q_\exists \uplus \{W\}, Q_\forall, S\theta, \sigma \circ \theta\rangle & (4)\\
\multicolumn{4}{l}{\qquad\theta = [X \mapsto \lambda\overline{z_n^1}.W\overline{z_k^3}] \text{ and } \overline{z_k^3} = \{z_i^1 \mid z_i^1 = z_i^2\}}\\
\langle Q_\exists \uplus \{Y\}, Q_\forall, S \uplus \{X\overline{z_n^1} \doteq Y\overline{z_m^2}\}, \sigma\rangle & \rightarrow & \langle Q_\exists, Q_\forall, S\theta, \sigma \circ \theta\rangle & (5)\\
\multicolumn{4}{l}{\qquad\text{where } X \neq Y,\ \theta = [Y \mapsto \lambda\overline{z_m^2}.X\overline{z_{\phi(m)}^2}] \text{ and } \phi \text{ is a permutation}}\\
\multicolumn{4}{l}{\qquad\text{such that } \phi(j) = i \text{ if } z_i^1 = z_j^2 \text{ for } 0 < i \leq n \text{ and } 0 < j \leq m}
\end{array}$$

Figure 2 Pattern Unification Algorithm.

▶ **Definition 11** (Pattern Unification Algorithm). The pattern unification algorithm is the application of the rules from Figure 2 such that before the application of rules (3) and (5) we apply exhaustively pruning.

Paper [20] contains a proof that the algorithm from Def. 11 is terminating, sound and complete.

3 A Unification Algorithm for FC Higher-order Unification Problems

3.1 FC Higher-order Unification (FCU) Problems

The main difference between pattern and FCU problems is in the form of arguments of existentially quantified variables. While in pattern unification problems, these arguments must be a list of distinct universally quantified variables which occur in the scope of the existentially quantified one, we relax this requirement for FCU problems. This relaxation still ensures the existence of mgus if the problems are unifiable.

▶ **Definition 12** (Restricted Terms). Given $\mathfrak{C}$, $Q_\forall$ and an equation e, a restricted term in e is defined inductively as follows:

- $a \in Q_\forall \cup \mathtt{bvars}(e)$ is a restricted term.
- $f\overline{t_n}$ is a restricted term if $n > 0$, $f \in \mathfrak{C} \cup Q_\forall \cup \mathtt{bvars}(e)$ and t_i is a restricted term for all $0 < i \leq n$.

When e is clear from the context, we will refer to these terms just as restricted terms.

We will use examples over $\mathfrak{C} = \{\mathtt{cons}\ ,\ \mathtt{fst}\ , \mathtt{snd}\ , \mathtt{nil}\}$ and $Q_\forall = \{l, z\}$ to explain the definition and algorithms presented in the paper.

▶ **Example 13.** The terms l, $(\mathtt{cons}\ z, l\ )$, $(\mathtt{cons}\ (\mathtt{fst}\ l)\ l)$ and $(\mathtt{snd}\ (\mathtt{cons}\ z\ l))$ are restricted terms over the above $\mathfrak{C}$ and $Q_\forall$, while $\mathtt{nil}$ and $(\mathtt{cons}\ z\ \mathtt{nil})$ are not.

▶ **Definition 14** (FCU Systems). A system $\langle Q_\exists, Q_\forall, S, \sigma\rangle$ is an FCU system if the following three conditions are satisfied:

- `argument restriction` - for all occurrences $X\overline{t_n}$ in S where $X \in Q_\exists$, t_i for all $0 < i \leq n$ is a restricted term.
- `local restriction` - for all occurrences $X\overline{t_n}$ in S where $X \in Q_\exists$ and for each t_i and t_j such that $0 < i, j \leq n$ and $i \neq j$, $t_i \not\sqsubseteq t_j$.

- `global restriction` - for each two different occurrences $X\overline{t_n}$ and $Y\overline{s_m}$ in S where $X, Y \in Q_\exists$ and for each $0 < i \leq n$ and $0 < j \leq m$, $t_i \not\sqsubseteq s_j$.

▶ **Example 15.** Some examples of the equations of FCU systems are $\{X\ l\ z \doteq \texttt{fst}\ (\texttt{snd}\ l)\}$ and $\{\texttt{cons}\ (X\ (\texttt{fst}\ l))\ (\texttt{snd}\ l) \doteq \texttt{snd}\ (Y\ (\texttt{fst}\ l)\ (\texttt{fst}\ (\texttt{snd}\ l)))\}$. Note that only the first example is a pattern unification problem. Examples of non-FCU problems are $\{X\ (\texttt{cons}\ z\ \texttt{nil}) \doteq \texttt{snd}\ l\}$ which violates the `argument restriction`, $\{X\ (\texttt{fst}\ l)\ l \doteq \texttt{cons}\ z\ l\}$ which violates the `local restriction` and $\{X\ (\texttt{fst}\ l) \doteq \texttt{snd}\ (Y\ (\texttt{cons}\ (\texttt{fst}\ l)\ (\texttt{snd}\ l)))\}$ which violates (only) the `global restriction`.

It is important to note that when dealing with *restricted* unification problems, the `global restriction` from above cannot be violated by occurrences within different equations. The name `global` only refers, therefore, to the context of one equation.

The next proposition is easy to verify.

▶ **Proposition 16.** *Pattern systems are FCU systems.*

Before going on to show the properties of these problems, we would like to present a short discussion about the motivation behind the restrictions above. The three restrictions are required in order to maintain uniqueness of the result and will be used in the next section in order to prove the determinacy of Huet's procedure over FCU problems. Nevertheless, we do not prove that this result does not hold when weakening the above restrictions. The `local restriction` and `global restriction` can easily be shown to be required even for very simple examples. This is not the case for the `argument restriction`. One alternative is to weaken the restricted term definition from above to require only one subterm in the second condition to be restricted. I.e. to allow terms such as $(\texttt{cons}\ z\ \texttt{nil})$ as arguments of existential variables. In the following, we will give a counter-example to this weaker restriction. Still, it should be noted that the counter-example depends on allowing inductive definitions containing more than one base case (in particular, we allow for different empty list constructors $\texttt{nil}_1$ and $\texttt{nil}_2$). When such definitions are not allowed, it may be possible to prove the results given in this paper for a stronger class of problems.

▶ **Example 17.** $\langle\{X, Y\}, \{z, z_2\}, \{X\ (\texttt{cons}\ z\ \texttt{nil}_1)\ (\texttt{cons}\ z\ \texttt{nil}_2) \doteq \texttt{cons}\ z\ (Y\ z_2)\}, \texttt{id}\rangle$ is unifiable by the following two incompatible substitutions:

1. $[Y \mapsto \lambda z_1.\texttt{nil}_1, X \mapsto \lambda z_1, z_2.z_1]$.
2. $[Y \mapsto \lambda z_1.\texttt{nil}_2, X \mapsto \lambda z_1, z_2.z_2]$.

3.2 The Existence of Most-general Unifiers

From this section on, an FCU problem will be referred to simply as system, unless indicated otherwise.

In [32] it is claimed that the only "don't-know" non-determinism in the general higher-order procedure stems from the choice between the different applications of `(Imitate)` and `(Project)`. We prove that fulfilling the three restrictions in Def. 14 makes these choices deterministic.

We first prove a couple of auxiliary lemmas.

▶ **Lemma 18.** *let t be a restricted term, s a term containing the subterm $X\overline{r_n}$ and σ a substitution such that $t = s\sigma$, then there is a restricted term t' such that $t' = X\overline{r_n}\sigma$.*

Proof. Let k be the length of the position of $X\overline{r_n}$ in s, we prove by induction on k.

- $k = 0$, then $t' = t$.

- $k > 0$, then $s = f\overline{s_m}$ and $f\overline{s_m}\sigma = f\overline{t_m} = t$. Since t is restricted, by definition so are $t_1, \ldots, t_m$. Assume $X\overline{r_n}$ occurs in s_i, then, according to the inductive hypothesis, there is a restricted term t' such that $t' = X\overline{r_n}\sigma$. ◀

▶ **Lemma 19.** *Given a unifiable equation $X\overline{t_n} \doteq r$, where r is a restricted term. Then, there is $0 < i \leq n$ such that t_i is a subterm of r.*

Proof. Assume the contrary and let σ be the unifier. Then, $X\overline{t_n}\sigma = r$. By definition, r contains a symbol $a \in Q_\forall$ and we get a contradiction. ◀

▶ **Lemma 20.** *Let $t = t'\overline{t_k}$ and $s = f\overline{s_n}$ such that t' is a restricted term and $f \in \mathfrak{C} \cup Q_\forall$. If $t \doteq s$ is unifiable, then $t = f\overline{v_{n-k}}\overline{t_k}$ for restricted terms $\overline{v_{n-k}}$.*

Proof. Since t' is restricted, it does not contain abstractions and variables and as t is unifiable with s, it can be written as $f\overline{v_{n-k}}$. Since t' is restricted, all its subterms are restricted as well. ◀

The next two lemmas prove the determinism claim on applications of `(Project)` and `(Imitate)`.

▶ **Lemma 21.** *Given the equation $X\overline{t_n} \doteq f\overline{s_m}$ where X does not occur in $f\overline{s_m}$ and assuming we can obtain the following two equations by applying the substitutions $\sigma_0 = X \mapsto \lambda\overline{z_n}.z_i\overline{X_m\overline{z_n}}$ and $\theta_0 = X \mapsto \lambda\overline{z_n}.z_j\overline{Y_k\overline{z_n}}$ for some $0 < i < j \leq n$:*

$$t_i\overline{X_l\overline{t_n}} \doteq f\overline{s_m} \tag{1}$$

and

$$t_j\overline{Y_k\overline{t_n}} \doteq f\overline{s_m} \tag{2}$$

Then, there are no substitutions σ and θ such that σ unifies equation 1 and θ unifies equation 2.

Proof. Assume the existence of the two unifiers and obtain a contradiction. According to Lemma 20, we can rewrite the two equations as

$$f\overline{v_{m-l}}\overline{X_l\overline{t_n}} \doteq f\overline{s_m} \tag{3}$$

and

$$f\overline{u_{m-k}}\overline{Y_k\overline{t_n}} \doteq f\overline{s_m} \tag{4}$$

for restricted terms $\overline{v_{m-l}}$ and $\overline{u_{m-k}}$. Assume, wlog, that $l \geq k$. Note also, that since $t_i \neq t_j$ and t_i, t_j have f as head symbol, $m \geq m - k > 0$. We consider two cases:

- $s_1, \ldots, s_{m-k}$ are all ground terms. In this case and since both equations are unifiable, we get the equation

  $$f\overline{v_{m-l}}\overline{X_{l-k}\overline{t_n}}\sigma = f\overline{s_{m-k}} = f\overline{u_{m-k}} \tag{5}$$

  Clearly, $k \neq l$ since otherwise $t_i = t_j$ which violates the `local restriction` from Def. 14. We can now conclude that

  $$X_1\overline{t_n}\sigma = u_{m-l+1} \tag{6}$$

  Since u_{m-l+1} is a restricted term then, according to Lemma 19, there is $0 < k_1 \leq n$ such that t_{k_1} is a subterm of u_{m-l+1}. Since u_{m-l+1} is a subterm of t_j, we get that t_{k_1} is a subterm of t_j which contradicts the `local restriction`.

- There is s_{k_1} for $0 < k_1 \leq m - k$ which contains an occurrence of $Z\overline{r_{k_2}}$. This must occur as a subterm of s_{k_1} as otherwise the subterm $Z\overline{r_{k_2}}r'$ where r' is not a restricted term, violates the `argument restriction`. Since $s_{k_1}\theta = u_{k_1}$ and u_{k_1} is a restricted term, we have, according to Lemma 18, that there exist a restricted term u' such that $Z\overline{r_{k_2}}\theta = u'$. Using Lemma 19, we can conclude that there is $0 < k_3 \leq k_2$ such that r_{k_3} is a subterm of u', which is a subterm of u_{k_1} which is a strict subterm of t_j, which violates the `global restriction`. ◀

▶ **Lemma 22.** *Given the equation $X\overline{t_n} \doteq f\overline{s_m}$ where X does not occur in $f\overline{s_m}$ and assuming we can obtain the following two equations by applying the substitutions $\sigma_0 = X \mapsto \lambda\overline{z_n}.f\overline{X_m\overline{z_n}}$ and $\theta_0 = X \mapsto \lambda\overline{z_n}.z_j\overline{Y_k\overline{z_n}}$ for some $0 < j \leq n$:*

$$f\overline{X_m\overline{t_n}} \doteq f\overline{s_m} \tag{7}$$

and

$$t_j\overline{Y_k\overline{t_n}} \doteq f\overline{s_m} \tag{8}$$

Then, there are no substitutions σ and θ such that σ unifies equation 7 and θ unifies equation 8.

Proof. Assume the existence of the two unifiers and obtain a contradiction. Using Lemma 20, we can rewrite Eq. 8 as:

$$f\overline{v_{m-k}}\overline{Y_k\overline{t_n}} \doteq f\overline{s_m} \tag{9}$$

where $v_1, \ldots, v_{m-k}$ are restricted terms and strict subterms of t_j. Since f is imitated, it is not a restricted term and $f \neq t_j$ which implies that $m - k > 0$. Eq. 9 tells us that $v_1 = s_1\theta$ which implies that $s_1\theta$ is a strict subterm of t_j and a restricted term. On the other hand, we have that $X_1\overline{t_n} = s_1\sigma$ from Eq. 7. We consider two cases:

- s_1 is ground. In this case we can use Lemma 19 and the fact that s_1 is a restricted term to conclude that there is $0 < k_1 \leq n$ such that t_{k_1} is a subterm of s_1. On the other hand, we know that s_1 is a strict subterm of t_j and therefore we get that t_{k_1} is a strict subterm of t_j, which contradicts the `local restriction`..
- If s_1 is not ground, it must contain an occurrence $Z\overline{r_l}$. This occurrence cannot occur as the subterm $Z\overline{r_l}r'$ where r' is not a restricted term as it violates the `argument restriction`. Therefore, $Z\overline{r_l}$ is a subterm of s_1. Since $s_1\theta = v_1$ and since v_1 is a restricted term, we can use Lemma 18 to get that there is a restricted term v' such that $Z\overline{r_l} = v'$. Now we use Lemma 19 and get that there is $0 < k_1 \leq l$ such that r_{k_1} is a subterm of v', which is a strict subterm of t_j. We get again a contradiction to the `global restriction`. ◀

▶ **Theorem 23** (The existence of most-general unifiers)**.** *Given a unifiable FCU system S, then applying the procedure in Def. 7 to S terminates and returns a most-general unifier for S.*

Proof. The procedure in Def. 7 computes complete sets of unifiers and terminates with an element in this set [32]. Using the lemmas 21 and 22 we obtain that all transformations are deterministic. Therefore, the complete set contains only one element, which is the most-general unifier of S. ◀

$$
\begin{aligned}
\langle Q_\exists, Q_\forall, S \uplus \{t \doteq t\}, \sigma\rangle &\quad\rightarrow\quad \langle Q_\exists, Q_\forall, S, \sigma\rangle && (0)\\
\langle Q_\exists, Q_\forall, S \uplus \{\lambda x.s \doteq \lambda x.t\}, \sigma\rangle &\quad\rightarrow\quad \langle Q_\exists, Q_\forall \uplus \{x\}, S \uplus \{s \doteq t\}, \sigma\rangle && (1)\\
\langle Q_\exists, Q_\forall, S \uplus \{f\overline{t_n} \doteq f\overline{s_n}\}, \sigma\rangle &\quad\rightarrow\quad \langle Q_\exists, Q_\forall, S \uplus \{t_1 \doteq s_1, \dots, t_n \doteq s_n\}, \sigma\rangle && (2)\\
&\text{where } f \in \mathfrak{C} \uplus Q_\forall\\
\langle Q_\exists \uplus \{X\}, Q_\forall, S \uplus \{X\overline{t_n} \doteq f\overline{s_m}\}, \sigma\rangle &\quad\rightarrow\quad \langle Q_\exists, Q_\forall, S\theta, \sigma \circ \theta\rangle && (3)\\
&\text{where } f \in \mathfrak{C},\ X \notin \texttt{fvars}(f\overline{s_m}) \text{ and } \theta = [X \mapsto \lambda\overline{z_n}.f\overline{s_m}\ |^{\overline{t_n}}_{\overline{z_n}}]\\
\langle Q_\exists \uplus \{X\}, Q_\forall, S \uplus \{X\overline{t_n} \doteq X\overline{s_n}\}, \sigma\rangle &\quad\rightarrow\quad \langle Q_\exists \uplus \{W\}, Q_\forall, S\theta, \sigma \circ \theta\rangle && (4)\\
&\text{where } W \notin Q_\exists,\ \overline{t_n} \neq \overline{s_n},\ \theta = [X \mapsto \lambda\overline{z_n}.W\overline{z_{r_k}}] \text{ and } \overline{r_k} = \{i \mid 0 < i \leq n \wedge t_i = s_i\}\\
\langle Q_\exists \uplus \{Y\}, Q_\forall, S \uplus \{X\overline{t_n} \doteq Y\overline{s_m}\}, \sigma\rangle &\quad\rightarrow\quad \langle Q_\exists, Q_\forall, S\theta, \sigma \circ \theta\rangle && (5)\\
&\text{where } X \neq Y,\ \theta = [Y \mapsto \lambda\overline{z_m}.X\overline{z_{\phi(m)}}] \text{ and } \phi \text{ is a permutation (see Lemma 36)}\\
&\text{such that } \phi(j) = i \text{ if } t_i = s_j \text{ for } 0 < i \leq n \text{ and } 0 < j \leq m
\end{aligned}
$$

Figure 3 An algorithm for FCU problems.

3.3 The Unification Algorithm

For defining the unification algorithm, we need to slightly extend the definition of pruning.

▶ **Definition 24** (Covers)**.** A cover for $X\overline{t_n}$ and a restricted term q is a substitution σ such that $X\overline{t_n}\sigma = q$.

Note, uniqueness of covers follows from Theorem 23

▶ **Example 25.** The following substitution $[X \mapsto \lambda z_1 \lambda z_2.\texttt{cons}\ (\texttt{fst}\ z_1)\ z_2]$ is a cover for $(X\ l\ z)$ and $(\texttt{cons}\ (\texttt{fst}\ l)\ z)$.

▶ **Definition 26** (Pruning)**.** Given an FCU system $\langle Q_\exists, Q_\forall, S, \sigma\rangle$ such that $(X\overline{t_n} \doteq r) \in S$ and r contains an occurrence of a maximal restricted term q such that $q \notin \overline{t_n}$:

- if there is a subterm $W\overline{s_m}$ of r such that $q = s_i$ for some $0 < i \leq m$, then return $\langle Q_\exists \uplus \{W'\} \setminus \{W\}, Q_\forall, S\theta, \sigma \circ \theta\rangle$ where $\theta = [W \mapsto \lambda\overline{z_m}.W'\overline{z'_{m-1}}]$ and $\overline{z'_{m-1}} = \overline{z_m} \setminus \{z_i\}$.
- else if there is no cover ρ for $X\overline{t_n}$ and q, then return $\langle Q_\exists, Q_\forall, \bot, \texttt{id}\rangle$.
- else, do nothing.

▶ **Example 27.** Given the system $\langle\{X, Y, W, Z\}, \{l, w, z\}, \{X\ (\texttt{snd}\ l)\ z \doteq Y\ z\ (\texttt{fst}\ l),$ $W\ (\texttt{fst}\ l)\ z \doteq \texttt{snd}\ (Z\ w\ (\texttt{fst}\ l))\}, \texttt{id}\rangle$, we can apply the following three prunings, $\sigma_1 = [Z \mapsto \lambda z_1, z_2.Z'z_1]$, $\sigma_2 = [Y \mapsto \lambda z_1, z_2.Y'z_1]$ and $\sigma_3 = [x \mapsto \lambda z_1, z_2.X'z_2]$ and obtain the system $\langle\{X', Y', W, Z'\}, \{l, w, z\}, \{X'\ z \doteq Y'\ z, W\ (\texttt{fst}\ l)\ z \doteq \texttt{snd}\ (Z'\ z)\}, \sigma_1 \circ \sigma_2 \circ \sigma_3\rangle$.

For the next definition, we will use the following replacement operator $r\ |^{\overline{t_n}}_{\overline{z_n}}$ to denote the replacement of each occurrence t_i in r with z_i for $0 < i \leq n$.

▶ **Definition 28** (Algorithm for FCU Systems)**.** The rules of an algorithm for the unification of FCU systems is given in Figure 3 where before the application of rules (3) and (5) we apply exhaustively pruning.

▶ **Example 29.** The following problem is contained in one of the classes of problems discussed in the introduction: $\exists X \exists Y \lambda l_1 \lambda l_2.X\ (\texttt{fst}\ l_1)\ (\texttt{fst}\ (\texttt{snd}\ l_1)) \doteq \lambda l_1 \lambda l_2.\texttt{snd}\ (Y\ (\texttt{fst}\ l_2)\ (\texttt{fst}\ l_1))$ Figure 4 gives a full execution of the algorithm on it.

Notice that this algorithm can also work with terms that are essentially untyped: it is the presence or absence of constructors and bound variables that matters in this algorithm and not the types of those constructors and variables. Rich typing can, of course, be used to disallow unifiers that are created by considering terms to be type-less.

$$\begin{array}{ll}
\langle \{X,Y\}, \emptyset, \{\lambda l_1 \lambda l_2 . X\ (\texttt{fst}\ l_1)\ (\texttt{fst}\ (\texttt{snd}\ (l_1))) \doteq \lambda l_1 \lambda l_2 . \texttt{snd}\ (Y\ (\texttt{fst}\ l_2)\ (\texttt{fst}\ l_1))\}, \texttt{id}\rangle & \to^{(1)\times 2} \\
\langle \{X,Y\}, \{l_1,l_2\}, \{X\ (\texttt{fst}\ l_1)\ (\texttt{fst}\ (\texttt{snd}\ (l_1))) \doteq \texttt{snd}\ (Y\ (\texttt{fst}\ l_2)\ (\texttt{fst}\ l_1))\}, \texttt{id}\rangle & \to^{\texttt{prun}} \\
\langle \{X,Y'\}, \{l_1,l_2\}, \{X\ (\texttt{fst}\ l_1)\ (\texttt{fst}\ (\texttt{snd}\ (l_1))) \doteq \texttt{snd}\ (Y'\ (\texttt{fst}\ l_1))\}, [Y \mapsto \lambda z_1 \lambda z_2 . Y' z_2]\rangle & \to^{(3)} \\
\langle \{Y'\}, \{l_1,l_2\}, \{\texttt{snd}\ (Y'\ (\texttt{fst}\ l_1)) \doteq \texttt{snd}\ (Y'(\texttt{fst}\ l_1))\}, [Y \mapsto \lambda z_1 \lambda z_2 . Y' z_2, X \mapsto \lambda z_1 \lambda z_2 . \texttt{snd}\ (Y' z_1)]\rangle & \to^{(0)} \\
\langle \{Y'\}, \{l_1,l_2\}, \emptyset, [Y \mapsto \lambda z_1 \lambda z_2 . Y' z_2, X \mapsto \lambda z_1 \lambda z_2 . \texttt{snd}\ (Y' z_1)]\rangle &
\end{array}$$

Figure 4 An example of a reduction on an FCU.

4 Correctness of the Algorithm

The unification algorithm transforms systems by the application of substitutions and by the elimination of equations. We prove next that the application of rules of the algorithm in Def. 28 on FCU problems results in FCU problems as well.

▶ **Lemma 30.** *Given an FCU problem, then the application of rules from Def. 28 results in ain FCU problem.*

Proof. Removing equations from the system clearly preserves the restrictions of FCU problems. This result is also immediate when applying substitutions as the only change to the arguments of the variables in the problem is to eliminate them and we have already claimed that the subterm property is closed under α-renaming for these problems in Lemma 5. ◀

The following lemma states that projected arguments of variables on one side of the equation must always match arguments on the other side.

▶ **Lemma 31.** *Let $X\overline{t_n} \doteq r$ be an equation such that r contains an occurrence of $Y\overline{s_m}$ where $r \neq Y\overline{s_m}$ and let σ be a unifier of this equation such that $\sigma Y = \lambda \overline{z_m}.s$. Then, for each occurrence z_i in s for $0 < i \leq m$, there is $0 < j \leq n$ such that $s_i = t_j$.*

Proof. We prove by induction on the number of occurrences. If s does not contain such occurrence, then the lemma clearly holds. Assume s contains an occurrence z_i for $0 < i \leq m$ and that there is no $0 < j \leq n$ such that $s_i = t_j$. In case there is more than one such occurrence in s, choose this occurrence to be in a minimal such subterm, i.e. z_i occurs in a subterm $z_i\overline{v_k}$ such that all occurrences of $z \in \overline{z_m}$ in $\overline{v_k}$ fulfill the requirement that there is $t_j = z$ for some $0 < j \leq n$. Let $\lambda \overline{z_m}.z_i\overline{v_k}(\overline{s_m}) = s_i\overline{v'_k}$. Since $r \neq Y\overline{s_m}$ and the `argument restriction`, we have that $Y\overline{s_m} \sqsubset r$. Since $X\overline{t_n}\sigma = r\sigma$, we get that $s_i\overline{v'_k} \sqsubset X\overline{t_n}\sigma$. Since s_i is a restricted term, we get that there is $0 < j \leq n$ such that either

- $s_i\overline{v'_k} \sqsubseteq t_j$. By the minimality assumption, if $\overline{v'_k}$ contains a restricted term, then it must be equal to some t_l $0 < l \leq n$ and therefore, that $t_l \sqsubset t_j$, which contradicts the `local restriction`. Therefore, since t_j is a restricted term, $k = 0$. We obtain that $s_i \sqsubseteq t_j$ and since $s_i \neq t_j$ by assumption, we get, again, a contradiction to the `global restriction`.
- $t_j \sqsubseteq s_i$. Again, since $s_i \neq t_j$, we get a contraction to the `global restriction`. ◀

We now prove, for each rule in the FCU algorithm, a relative completeness result. We start by the non-unifiability of problems with a positive occur check.

▶ **Lemma 32.** *Let $\langle Q_\exists, Q_\forall, S \cup \{X\overline{t_n} \doteq f\overline{s_m}\}, \sigma\rangle$ be a system such that X occurs in $\overline{s_m}$, then the system is not unifiable.*

Proof. Assume it is unifiable by θ and $\theta X = \lambda \overline{z_n}.s$. Consider two cases:

- s does not contain any occurrence of a variable z_i for $0 < i \leq n$. Let $\#t$ be the number of occurrences of symbols from $\mathfrak{C}$ in t. Then, $\#(X\overline{t_n}\theta) = \#(\theta X) \leq \#(\overline{s_m}\theta) < \#(f\overline{s_m}\theta)$ and we get a contradiction to $X\overline{t_n}\theta = f\overline{s_m}\theta$.

- In case s contains such an occurrence and let $X\overline{q_n}$ be the occurrence in $\overline{s_m}$. According to Lemma 31, we know that for all occurrences of z_i in s for $0 < i \leq n$, there is an index $0 < j \leq n$ such that $t_j = q_i$. Let ρ be the mapping between indices defined as above such that $\rho(i) = j$. Let $\overline{r_k}$ be the set of indices $0 < i \leq n$ which occur in s for some $k \leq n$. Let p' be the non-trivial position of $X\overline{q_n}$ in $f\overline{s_m}$ and let p be the maximal position of a z_i in s for $i \in \overline{r_k}$. This means we have q_i at position $p' \circ p$ in $f\overline{s_m}\theta$ and since $t_{\rho(i)} = q_i$ and $X\overline{t_n}\theta = f\overline{s_m}\theta$, we get that $t_{\rho(i)}$ occurs at position $p' \circ p$ in $X\overline{t_n}\theta$. Since $t_{\rho(i)}$ is a restricted term, there is an occurrence of $z_{\rho(i)}$ in s at position $p' \circ p$, in contradiction to the maximality of p. ◀

▶ **Lemma 33.** *Given a system $\langle Q_\exists, Q_\forall, S, \rho\}$ where $X\overline{t_n} \doteq r \in S$ and assuming we apply pruning in order to obtain system $\langle Q_\exists, Q_\forall, S', \rho'\}$ as defined in Def. 26, then, if S is unifiable by substitution σ, then there is a substitution σ', such that $\sigma = \theta \circ \sigma'$. If S is not unifiable, then S' is not unifiable.*

Proof. The rule is applicable only if there is such an occurrence q. Otherwise, $S = S'$ and $\theta = \mathtt{id}$. We consider the two cases in the lemma:

- there is a subterm $Y\overline{s_m}$ of r such that $q = s_i$ for $0 < i \leq m$. If S is not unifiable, then assume S' is unifiable by σ' and since $S' = S\theta$, we get that S is unifiable by $\theta \circ \sigma'$, a contradiction. Assume the system is unifiable and let $\sigma Y = \lambda\overline{z_m}.s$. Then, according to Lemma 31, either there is $0 < j \leq n$, such that $t_j = s_i$, which contradicts the assumption, or s does not contain an occurrence of s_i. In the second case, by taking $\sigma' = \sigma|_{\mathfrak{V}(S)\setminus\{Y\}}[W \mapsto \lambda\overline{z_{r_{m-1}}}.Y\overline{z_m}\sigma]$ where $\overline{z_m} \setminus \overline{z'_{m-1}} \not\subseteq Q_\forall$, we get that $Y\overline{s_m}\sigma = W\overline{s_{r_{m-1}}}\sigma' = Y\overline{s_m}\theta \circ \sigma'$.
- If there is no such cover, then there is no substitution which unifies this equation. ◀

▶ **Lemma 34.** *Given a system $\langle Q_\exists, Q_\forall, S, \rho\rangle$ where $X\overline{t_n} \doteq s \in S$ and X does not occur in s and assuming we apply the substitution θ as defined in rule (3) in Figure 3. Then, if σ a unifier of S, then there is σ' such that $\sigma = \theta \circ \sigma'$.*

Proof. We prove by induction on the structure of s. Note that two base cases are also defined in the last two cases below for $m = 0$.

- $s = Y\overline{s_m}$ for $0 \leq m$. Note, that in this case, since $t_i \not\sqsubseteq s_j$ for $0 < i \leq n$ and $0 < j \leq m$ and that since $m \leq n$ due to pruning, we get that $X\overline{t_n}\theta = Y\overline{t_{\phi(m)}}$ for ϕ defined as in rule (5) in Figure 3. The rest of the proof is similar to the proof of Lemma 36.
- $s = t_i\overline{s_m}$ for some $0 < i \leq n$ and $0 \leq m$ and therefore $\theta X = \lambda\overline{z_n}.z_i\overline{s'_m}$ for some $\overline{s'_m}$. Assume applying (`Project`) with the substitution $\theta' = \lambda\overline{z_n}.z_i\overline{X_m\overline{z_n}}$ as defined in Def. 7. After applying (`Bind`) and possibly also several (`Decomp`), we get the problem, since X cannot occur in s,

$$S'\theta' \cup \{X_1\overline{t_n} \doteq s_1, \ldots, X_m\overline{t_n} \doteq s_m\} \tag{10}$$

Since, by assumption, $X\overline{t_n} \doteq t_i\overline{s_m}$ is unifiable and t_i is a restricted term, it follows, using an argument similar to the one in the proof of Lemma 21, that also each of the $X_j\overline{t_n} \doteq s_j$ is unifiable by some σ^o_j for $0 < j \leq m$. By following lemmas 21 and 22, we have that applying any other projection or imitation to $X\overline{t_n} \doteq t_i\overline{s_m}$ will render it non-unifiable. By using Theorem 8, we have that σ^o_j can be extended into a unifier σ_j of $S'\theta' \cup \{X_j\overline{t_n} \doteq s_j\}$ for all $0 < j \leq m$ and $\sigma = \theta' \circ \sigma_1 \circ \ldots \circ \sigma_m$. By applying the induction hypothesis, we get that there are substitutions σ'_j unifying $S'\theta' \cup \{X_j\overline{t_n} \doteq s_j\}$ for all $0 < j \leq m$ such that $\sigma_j = \theta_j \circ \sigma'_j$ for $\theta_j X_j = \lambda\overline{z_n}.s'_j$. I.e. that $\sigma = \theta' \circ \theta_1 \circ \sigma'_1 \circ \ldots \circ \theta_m \circ \sigma'_m$. Since each σ_j is a unifier of the above equations and σ is a unifier of S, we get that for each $m \geq j' > j$,

$\sigma'_{j'} \leq \sigma'_j|_{\mathtt{dom}(\sigma'_{j'})}$. From this, together with the fact that the domain and range of each σ'_j do not contain variables from the domain of each $\theta_{j'}$ for $0 < j < j' \leq m$, we get that $\sigma = \theta' \circ \theta_1 \circ \ldots \circ \theta_m \circ \sigma'_1 \circ \ldots \circ \sigma'_m$. On the other hand, by applying θ, we get the problem

$$S'\theta \cup \{s'_1 \doteq s_1, \ldots, s'_m \doteq s_m\} \tag{11}$$

But, since $\theta = \theta' \circ \theta_1 \circ \ldots \circ \theta_m$, we just need to choose $\sigma' = \sigma'_1 \circ \ldots \circ \sigma'_m$ and we have $\sigma = \theta \circ \sigma'$.

- $s = f\overline{s_m}$ for $0 \leq m$ and therefore $\theta X = \lambda\overline{z_n}.f\overline{s'_m}$. Assume applying `(Imitate)` with the substitution $\theta' = \lambda\overline{z_n}.f\overline{X_m\overline{z_n}}$. After applying `(Bind)` and a `(Decomp)`, we get the problem, since x cannot occur in s,

$$S'\theta' \cup \{X_1\overline{t_n} \doteq s_1, \ldots, X_m\overline{t_n} \doteq s_m\} \tag{12}$$

From here we follow as before and use again Theorem 8 and Lemma 22. ◀

▶ **Lemma 35.** *Given a system $\langle Q_\exists, Q_\forall, S' \cup \{X\overline{t_n} \doteq X\overline{s_n}\}, \rho\rangle$ and assuming we apply the substitution θ as defined in rule (4) in Figure 3. Then, if the system is unifiable by a substitution σ, then there is σ' such that $\sigma = \theta \circ \sigma'$.*

Proof. Assume that $\sigma X = \lambda\overline{z_n}.s$, we first prove that there is no occurrence z_i in s such that there is $0 < j \leq k$ where $i = r_j$ and $t_i \neq s_i$. Assume on the contrary, then $X\overline{t_n}\sigma = X\overline{s_n}\sigma$ which implies that $t_i = s_i$. Now we can define $\sigma' = \sigma|_{\mathfrak{V}(S)\setminus\{X\}}[W \mapsto \lambda\overline{z_{r_k}}.X\overline{z_n}\sigma]$ where $\overline{z_n} \setminus \overline{z_{r_k}} \not\subseteq Q_\forall$ are new variables. ◀

▶ **Lemma 36.** *Given a system $\langle Q_\exists, Q_\forall, S' \cup \{X\overline{t_n} \doteq Y\overline{s_m}\}, \rho\rangle$ where $X \neq Y$ and assuming we apply the substitution θ as defined in rule (5) in Figure 3. Then, if the system is unifiable by a substitution σ, then there is σ' such that $\sigma = \theta \circ \sigma'$.*

Proof. First note that since we apply pruning beforehand (in a symmetric way), $n = m$ and ϕ is indeed a permutation. Assume, wlog, that $\sigma X = \lambda\overline{z_n}.s$. We know that for each occurrence z_i in s for $0 < i \leq n$, $s_i = t_{\phi(i)}$. By choosing $\sigma' = \sigma|_{\mathfrak{V}(S)\setminus\{Y\}}$, we get that $Y\overline{s_m}\sigma = X\overline{s_{\phi(m)}}\sigma = X\overline{s_{\phi(m)}}\sigma' = Y\overline{s_m}\theta \circ \sigma'$. Therefore, $\sigma = \theta \circ \sigma'$. ◀

▶ **Theorem 37** (Termination)**.** *Given a system $\langle Q_\exists, Q_\forall, S, \sigma\rangle$, the algorithm in Def. 28 always terminates.*

Proof. Let the tuple $m = \langle m_1, m_2\rangle$ where m_1 is the size of the set $Q_\exists$ and m_2 is the number of all symbols except $\doteq$ in S. Consider its lexicographical order, it is clear that m is well founded. We show that it decreases with every rule application of the algorithm:

- rules (0), (1) and (2) decrease m_2 and do no increase m_1.
- rule (3) decreases m_1.
- rule (4) decreases m_2 and does not increase m_1.
- rule (5) decreases m_1.
- pruning decreases m_2 and does not increase m_1. ◀

▶ **Theorem 38** (Completeness)**.** *Given a system $\langle Q_\exists, Q_\forall, S, \boldsymbol{id}\rangle$ and assuming it is unifiable by σ, there there is a sequence of rule applications in Def. 28 resulting in $\langle Q'_\exists, Q'_\forall, \emptyset, \theta\rangle$ such that $\theta \leq \sigma$.*

Proof. Since the algorithm terminates and since it has a rule application for each unifiable equation, we obtain at the end the above system. Lemmas 32, 33, 34, 35 and 36 then give us that for any unifier σ of S, there is a substitution σ' such that $\sigma = \theta \circ \sigma'$. Therefore, $\theta \leq \sigma$. ◀

The next theorem is an easy corollary of the completeness theorem.

▶ **Theorem 39** (Most-general unifier). *Given a system $\langle Q_\exists, Q_\forall, S, \mathtt{id}\rangle$, if the algorithm defined in Def. 28 terminates with system $\langle Q'_\exists, Q'_\forall, \emptyset, \sigma\rangle$, then σ is an mgu of S.*

Proof. Since the algorithm in Def. 28 is deterministic, then we can use Theorem 38 in order to prove that σ is an mgu. ◀

The next theorem is proved by simulating the algorithm in Def. 28 using the procedure in Def. 7.

▶ **Theorem 40** (Soundness). *Given a system $\langle Q_\exists, Q_\forall, S, \mathtt{id}\rangle$ and assuming there is a sequence of rule applications in Def. 28 resulting in $\langle Q'_\exists, Q'_\forall, \emptyset, \theta\rangle$, then θ is a unifier of S.*

Proof. It is obvious we can simulate each of the rules (0), (1), (2) and (3) using the procedure in Def. 7. We get the required result by using Theorem 23. For rules (4), (5) and the first case of pruning, assume there is another substitution ρ such that ρ unifies the problem and $\rho \not< \theta$. This can only happen if $\rho X = \lambda \overline{z_n}.W'\overline{r_{k'}}$ such that $\overline{r_k} \subset \overline{r'_{k'}}$. Lemma 35 states that there is no unifier ω and a substitution γ such that $\omega = \rho \circ \gamma$. Since the second case of pruning results in failure, we are done. ◀

5 Conclusion

We have described an extension of pattern unification called *function-as-constructor* unification. Such unification problems typically show up in situations where functions are applied to bound variables and where such functions are treated as term constructors (at least during the process of unification). We have shown that the properties that make first-order and pattern unification desirable for implementation – decidability and the existence of mgus for unifiable pairs – also hold for this class of unification problems. We are planning an implementation within the Leo-III theorem prover [35] and we then plan to compare this approach to unification with the implementation of Huet's pre-unification algorithm available in Leo-III when exercised against the THF set of problems within TPTP [8].

Another possible extension of the work is to improve its complexity class. The current algorithm, like the one in [20] and first-order unification algorithms which are used in practice, is of an exponential complexity. We would like to follow the work of Qian [34] and prove that FCU is of linear complexity.

Acknowledgments. We thank Enrico Tassi for discussions related to this paper. We benefited greatly from the many comments of the anonymous reviewers of an earlier draft of this paper. This work has been funded by the ERC Advanced Grant ProofCert.

References

1 The Twelf project, 2016. `http://twelf.org/`.

2 Andreas Abel and Brigitte Pientka. Higher-order dynamic pattern unification for dependent types and records. In *Typed Lambda Calculi and Applications*, pages 10–26. Springer, 2011.

3 P. B. Andrews, F. Pfenning, S. Issar, and C. P. Klapper. The TPS theorem proving system. In Jörg H. Siekmann, editor, CADE 8, LNCS 230, pages 663–664. Springer, July 1986.

4 Andrea Asperti, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. Crafting a proof assistant. In *Types for Proofs and Programs*, pages 18–32. Springer, 2006.

5 D. Baelde, K. Chaudhuri, A. Gacek, D. Miller, G. Nadathur, A. Tiu, and Y. Wang. Abella: A system for reasoning about relational specifications. *J. of Formalized Reasoning*, 2014.

6 Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, New York, revised edition, 1984.

7 Christoph Benzmüller and Michael Kohlhase. LEO – a higher order theorem prover. In *15th Conf. on Automated Deduction (CADE)*, pages 139–144, 1998.

8 Christoph Benzmüller, Florian Rabe, and Geoff Sutcliffe. THF0–the core of the TPTP language for higher-order logic. In *Automated Reasoning*, pages 491–506. Springer, 2008

9 Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of Agda - A functional language with dependent types. In *TPHOLs*, volume 5674, pages 73–78. Springer, 2009.

10 Chad E Brown. Satallax: An automatic higher-order prover. In *Automated Reasoning*, pages 111–117. Springer, 2012.

11 A. Church. A formulation of the Simple Theory of Types. *J. of Symbolic Logic*, 1940.

12 Mary Dalrymple, Stuart M. Shieber, and Fernando C. N. Pereira. Ellipsis and higher-order unification. *Linguistics and Philosophy*, 14:399–452, 1991.

13 Dominic Duggan. Unification with extended patterns. *Theoretical Computer Science*, 206(1):1–50, 1998.

14 R. Fettig and B. Löchner. Unification of higher-order patterns in a simply typed lambda-calculus with finite products and terminal type. In RTA 1996, LNCS 1103, pages 347–361.

15 Warren Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981.

16 Gérard Huet. The undecidability of unification in third order logic. *Information and Control*, 22:257–267, 1973.

17 Gérard Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.

18 Gérard Huet and Bernard Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, 11:31–55, 1978.

19 Raymond McDowell and Dale Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Trans. on Computational Logic*, 3(1):80–136, 2002.

20 Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. of Logic and Computation*, 1(4):497–536, 1991.

21 Dale Miller and Gopalan Nadathur. Some uses of higher-order logic in computational linguistics. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pages 247–255, 1986.

22 Dale Miller. Unification under a mixed prefix. *Journal of Symbolic Computation*, 14(4):321–358, 1992.

23 Dale Miller and Gopalan Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, June 2012.

24 Gopalan Nadathur and Dustin J. Mitchell. System description: Teyjus – A compiler and abstract machine based implementation of λProlog. CADE 16, LNAI 1632, pp. 287–291, 1999.

25 Tobias Nipkow. Functional unification of higher-order patterns. In M. Vardi, editor, *8th Symp. on Logic in Computer Science*, pages 64–74. IEEE, June 1993.

26 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*. Number 2283 in LNCS. Springer, 2002.

27 Frank Pfenning. Unification and anti-unification in the Calculus of Constructions. In G. Kahn, editor, *6th Symp. on Logic in Computer Science*, pages 74–85. IEEE, July 1991.

28 Frank Pfenning and Carsten Schürmann. System description: Twelf – A meta-logical framework for deductive systems. CADE 16, LNAI 1632, pages 202–206, Trento, 1999.

29 Brigitte Pientka and Joshua Dunfield. Beluga: A framework for programming and reasoning with deductive systems (system description). In J. Giesl and R. Hähnle, editors, IJCAR, LNCS 6173, pages 15–21, 2010.

30 Xiaochu Qi, Andrew Gacek, Steven Holte, Gopalan Nadathur, and Zach Snow. The Teyjus system – version 2, 2015. `http://teyjus.cs.umn.edu/`.

31 Helmut Schwichtenberg. Minlog. In Freek Wiedijk, editor, *The Seventeen Provers of the World*, volume 3600 of *LNCS*, pages 151–157. Springer, 2006.

32 Wayne Snyder and Jean H. Gallier. Higher order unification revisited: Complete sets of transformations. *Journal of Symbolic Computation*, 8(1-2):101–140, 1989.

33 Alwen F. Tiu. An extension of L-lambda unification. `http://www.ntu.edu.sg/home/atiu/llambdaext.pdf`, September 2002.

34 Zhenyu Qian. Linear Unification of Higher-Order Patterns. Proc. Coll. Trees in Algebra and Programming, 1993.

35 Max Wisniewski, Alexander Steen, and Christoph Benzmüller. The Leo-III project. In Alexander Bolotov and Manfred Kerber, editors, *Joint Automated Reasoning Workshop and Deduktionstreffen*, page 38, 2014.

36 Beta Ziliani and Matthieu Sozeau. A unification algorithm for Coq featuring universe polymorphism and overloading. ICFP 2015, pp. 179–191.

Homological Computations for Term Rewriting Systems

Philippe Malbos[1] **and Samuel Mimram**[2]

1 **LIX, École Polytechnique, Palaiseau, France**
samuel.mimram@lix.polytechnique.fr

2 **ICJ, Université de Lyon, Lyon, France**
malbos@math.univ-lyon1.fr

Abstract

An important problem in universal algebra consists in finding presentations of algebraic theories by generators and relations, which are as small as possible. Exhibiting lower bounds on the number of those generators and relations for a given theory is a difficult task because it a priori requires considering all possible sets of generators for a theory and no general method exists. In this article, we explain how homological computations can provide such lower bounds, in a systematic way, and show how to actually compute those in the case where a presentation of the theory by a convergent rewriting system is known. We also introduce the notion of coherent presentation of a theory in order to consider finer homotopical invariants. In some aspects, this work generalizes, to term rewriting systems, Squier's celebrated homological and homotopical invariants for string rewriting systems.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases term rewriting system, Lawvere theory, Tietze equivalence, resolution, homology, convergent presentation, coherent presentation

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.27

1 Introduction

An algebraic theory is a mathematical structure specified by operations, with given arities, and relations between those, i.e. a term rewriting system if we consider the relations as being oriented. For instance, the theory of *groups* is given by three operations m of arity two (the multiplication), e of arity zero (the neutral element) and i of arity one (the inverse), subject to the five expected relations:

$$m(e,x_1)=x_1 \qquad m(x_1,e)=x_1 \qquad m(m(x_1,x_2),x_3)=m(x_1,m(x_2,x_3))$$
$$m(i(x_1),x_1)=e \qquad m(x_1,i(x_1))=e$$

Of course there are many ways of specifying, or *presenting*, an algebraic theory. For instance, the relations in the second column are derivable from the other, and we could therefore as well remove them and still get a presentation for the theory of groups, with only three relations. This observation is in fact the starting point of the work of Knuth and Bendix in rewriting theory [10]: by adding derivable relations, in good cases one can obtain a set of relations which are much better behaved from a computational point of view, such as being confluent and terminating, without changing the presented theory. In the case of the theory of groups, one can actually come up with an even smaller presentation by considering other generators; it can be axiomatized with only two generators a of arity zero (standing for any

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 27; pp. 27:1–27:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

element, in order to exclude the "empty group") and d of arity two (standing for division) subject to only one relation [7]:

$$d(x_1, d(d(d(d(x_1, x_1), x_2), x_3), d(d(d(x_1, x_1), x_1), x_3))) \quad = \quad x_2 \tag{1}$$

see also [24, 23] for other possible axiomatizations of the theory of groups with one relation.

This quest for small presentations was initiated by Tarski who first gave a similar presentation of abelian groups with one rule [30]: those with only one relation are of particular interest and are sometimes called *one-based* theories in the literature. As illustrated in the example above, this is not an easy task: in the case of groups, one had to think of completely changing the set of generators and relations... Let us briefly recall some achievements in the field, a detailed overview of the subject can be found in [17]. The theories of semi-lattices [27] and distributive lattices [20] are not one-based. In contrast the theory of lattices is one-based: first, a unique relation was shown to exist by general methods [20], giving rise to a relation of length 300000 on 34 variables, and was then reduced to one of length 29 on 8 variables [18]. Similarly, for boolean algebras a single axiom was provided [26], but its size was more than 40 million symbols [17], and shorter axioms (around a dozen of symbols) were found afterward [19] by using intensive combinatorial computations.

In this article, we provide a novel method of showing that a theory is not one-based, using homological invariants, when the theory is given by a convergent rewriting system. The results mentioned above (such as semi- and distributive lattices) required lots of inventivity and are specific to the considered cases. By contrast, our methods are completely mechanical: by performing a series of computations, one obtains a lower bound on the number of rules in any presentation of the theory, and if this lower bound is greater than two, we know that we need at least two relations to present it, and therefore that the theory is not one-based. Of course, our method does not always gives interesting results: it might answer zero as a lower bound, from which we cannot conclude anything.

Homological invariants

The homology of a space consists in a sequence of groups which encode the number of "holes" in each dimension, and moreover they constitute invariants of the spaces in the sense that two homotopy equivalent spaces have the same associated groups [15, 6]. Homology can also be computed for algebraic structures which are not obviously spaces, such as monoids, groups, algebras, operads, etc. In the case of monoids, Squier has shown how to compute those invariants in small dimensions [28, 12] when the monoid is presented by a convergent string rewriting system, and this construction has since then been generalized in every dimension [11]. Here also, the interest of this construction lies in the fact that, even though it is constructed from a particular presentation, it does not actually depend on the choice of the presentation, only the presented monoid. In particular, the rank of the second homology group is, by construction, a natural number which is smaller than the number of relations of the presentation used to compute it, and thus a lower bound for the number of relations of any presentation since it is an invariant.

In this article, we generalize this approach from monoids presented by convergent string rewriting systems to algebraic theories presented by convergent term rewriting systems, and use the resulting homology computations to provide lower bounds on the number of generators or relations required to present an algebraic theory. This work is based on Jibladze and Pirasvili's definition of a cohomology for algebraic theories [8, 9], as well as Malbos' PhD thesis [16]. The first contribution of this article is to reformulate in concrete terms the fairly abstract categorical definitions used in those works. We also introduce a resolution when the algebraic theory admits a convergent presentation, which allows us to compute the homology in practice, using classical constructions in rewriting theory and linear algebra. Finally, we also explain how those invariants can be refined into ones of more homotopical nature by introducing a notion of coherent presentation for algebraic theories. Due to space constraints we cannot detail all the constructions performed here, and advise the reader willing to grasp the details to first understand the simpler case of monoids [28, 12], of which this construction is largely inspired; details shall be given somewhere else, and the present article focuses mainly on computations and applications.

2 Presentations of Lawvere algebraic theories

2.1 Term rewriting systems

Terms

A *signature* (P_1, σ_0) consists of a set P_1 of *operations* together with a function $\sigma_0 : \mathsf{P}_1 \to \mathbb{N}$ associating to each operation its *arity*. Supposing fixed an infinite countable set $\mathcal{X} = \{x_1, x_2, \ldots\}$ of variables, one can consider *terms* generated by operations with variables in this set, which are defined as usual. We write $\mathrm{FV}(t)$ for the set of indices of *free variables* occurring in a term, e.g. $\mathrm{FV}(f(x_2, g(x_2, x_5)) = \{2, 5\}$. Parallel *substitution* of x_i by t_i in a term u is denoted $u[t_1/x_1, \ldots, t_n/x_n]$.

The terms generated by the signature form a category, denoted P_1^*, whose objects are natural numbers and morphisms in $\mathsf{P}_1^*(m, n)$ are n-uples $\langle t_1, \ldots, t_n\rangle$ of terms t_i with free variables in $\{x_1, \ldots, x_m\}$. Composition of two morphisms $\langle t_1, \ldots, t_n\rangle : m \to n$ and $\langle u_1, \ldots, u_p\rangle : n \to p$ is induced by substitution as follows:

$$\langle u_1, \ldots, u_p\rangle \circ \langle t_1, \ldots, t_n\rangle \quad = \quad \langle u_1[t_1/x_1, \ldots, t_n/x_n], \ldots, u_p[t_1/x_1, \ldots, t_n/x_n]\rangle$$

and the identity on n is $\langle x_1, \ldots, x_n\rangle : n \to n$. We sometimes overload the notation and denote by $\mathsf{P}_1^* = \coprod_{m,n\in\mathbb{N}} \mathsf{P}_1^*(m, n)$ the class of all morphisms of this category and by $\sigma_0^*, \tau_0^* : \mathsf{P}_1^* \to \mathbb{N}$ the functions respectively associating to a morphism its source and target, also called its *arity* and *coarity*. Note that terms are the morphisms of coarity 1. We write $\iota_1 : \mathsf{P}_1 \to \mathsf{P}_1^*$ for the canonical inclusion, sending an n-ary operation f to the term $f(x_1, \ldots, x_n)$.

Term rewriting systems

We suppose fixed a signature as above. A *term rewriting system* on the signature P_1 consists of a set P_2, whose elements are called *rewriting rules*, together with two functions $\sigma_1, \tau_1 : \mathsf{P}_2 \to \mathsf{P}_1^*$ associating to each rule its source and target which should be (1-uples of) terms. The source and target of a rule should have the same arity, which is called the *arity* of the rule. A rewriting system together with the corresponding signature thus consists of a diagram of sets and functions

$$P \quad = \quad \begin{tikzcd} & \mathsf{P}_1 \arrow[dl, shift right, "\sigma_0"'] \arrow[dl, shift left, "\tau_0"] \arrow[d, "\iota_1"] & \mathsf{P}_2 \arrow[dl, shift right, "\sigma_1"'] \arrow[dl, shift left, "\tau_1"] \\ \mathbb{N} & \mathsf{P}_1^* \arrow[l, shift right, "\sigma_0^*"'] \arrow[l, shift left, "\tau_0^*"] & \end{tikzcd} \qquad \text{such that } \sigma_0^* \circ \sigma_1 = \sigma_0^* \circ \tau_1.$$

We sometimes write $R : t \Rightarrow u$ to denote a rule R with $\sigma_1(R) = t$ and $\tau_1(R) = u$. Note that we consider the signature as being part of the rewriting system.

▶ **Example 1** (Monoids). The rewriting system corresponding to monoids has operations $\mathsf{P}_1 = \{m, e\}$, with $\sigma_0(m) = 2$ and $\sigma_0(e) = 0$, and rewriting rules $\mathsf{P}_2 = \{A, L, R\}$ with $\sigma_1(A) = m(m(x_1, x_2), x_3)$, $\tau_1(A) = m(x_1, m(x_2, x_3))$, $\sigma_1(L) = m(e, x_1)$, $\sigma_1(R) = m(x_1, e)$, $\tau_1(L) = \tau_1(R) = x_1$. Such a rewriting system will often be written more concisely

$$\langle m : 2, e : 0 \mid A : m(m(x_1, x_2), x_3) \Rightarrow m(x_1, m(x_2, x_3)), L : m(e, x_1) \Rightarrow x_1, R : m(x_1, e) \Rightarrow x_1\rangle$$

▶ **Example 2** (Groups). The rewriting system for groups is obtained from the rewriting system for monoids by adding a generator i of arity one and relations $I : m(i(x), x) \Rightarrow e$ and $I' : m(x, i(x)) \Rightarrow e$.

Contexts

An n-ary *context* C is a term with variables in $\{x_1, \ldots, x_n, \Box\}$, in which the "variable" $\Box$ occurs exactly once and is called the *hole* of C. Given a term t, the substitution $C[t/\Box]$ is often denoted $C[t]$. The n-ary contexts form a category $\mathcal{K}_n$ with one object, morphisms being n-ary contexts, with composition given by substitution $D \circ C = D[C]$ and neutral element by the identity context $\Box$. In turn, these categories induce a functor $\mathcal{K} : (\mathsf{P}_1^*)^{\mathrm{op}} \to \mathbf{Cat}$, sending an object n to $\mathcal{K}_n$ and a morphism $u = \langle u_1, \ldots, u_n\rangle : m \to n$ to the functor $\mathcal{K}_u : \mathcal{K}_n \to \mathcal{K}_m$ such that the image of an n-ary context C is the m-ary context $\mathcal{K}_u C = C[u_1/x_1, \ldots, u_n/x_n]$. In the following, we simply write Cu instead of $\mathcal{K}_u C$, and the previous categorical discussion boils down to the simple facts that $(Cu)v = C(u \circ v)$, $C\mathrm{id} = C$, $(D \circ C)u = D \circ (Cu)$ and $\Box u = \Box$. An *occurrence* of a variable x_i in a term t is a context obtained from t by replacing exactly one instance of the variable x_i by $\Box$, those will be formally defined (in a linear context) in Definition 13.

We write $\mathbb{K}$ for the category whose objects are natural numbers and morphisms in $\mathbb{K}(m, n)$ are *bicontexts*, i.e. pairs (C, u) consisting of a context C with variables in $\{x_1, \ldots, x_n\}$, and a morphism $u \in \mathsf{P}_1^*(n, m)$. The composition of two morphisms $(C, u) : m \to n$ and $(D, v) : n \to p$ is given by $(D, v) \circ (C, u) = (D \circ Cv, u \circ v)$ and the identity on n is $(\Box, \mathrm{id}_n) : n \to n$. Note that composition is reversed in the second component. A bicontext $(C, u) : m \to n$ induces a function $C[-]u : \mathsf{P}_1^*(m, 1) \to \mathsf{P}_1^*(n, 1)$ which to a term t associates the term $C[t \circ u]$, which we will write $C[t]u$ in the following; for this reason, we will sometimes abusively write $C[-]u$ for a context in order to avoid introducing heavy notations. This function is easily shown to be compatible with composition and substitution:

$$D[C[t]u]v = (D \circ C)[t](u \circ v) \qquad \Box[t]\mathrm{id} = t \qquad (C[t]u) \circ v = (Cv)[t](u \circ v)$$

In particular, we have $C[t]\mathrm{id} = C[t]$ which makes the notation unambiguous on this point, and we will always write composition symbol "$\circ$" in order to avoid confusion in wrt the equation on the right above.

In the following, when we need to distinguish between multiple rewriting systems, we will add those in exponent to the constructions, i.e. we write $\mathcal{K}^P$ instead of simply $\mathcal{K}$ for the contexts of P, etc.

Rewriting

Suppose fixed a rewriting system P. We say that a term t *rewrites in one step* into t', what we write $t \longrightarrow t'$, when there exists a rule $R : u \Rightarrow u'$ of arity m and a bicontext $(C, v) : m \to n$ such that $t = C[u]v$ and $t' = C[u']v$. In this situation, we often write $C[R]v : t \longrightarrow t'$ and the term t is said to be *reducible* by the rule R. We write $\overset{*}{\longrightarrow}$ for the reflexive and transitive closure, and $\xleftrightarrow{*}$ for the generated equivalence relation. Note that the latter relation is a congruence, in the sense that it is compatible with composition, identities and taking tuples.

A rewriting system is *terminating* when there is no infinite sequence $t_0 \longrightarrow t_1 \longrightarrow \ldots$ of rewriting steps and *confluent* (resp. *locally confluent*) when for every terms t, u_1, u_2 such that $u_1 \xleftarrow{*} t \xrightarrow{*} u_2$ (resp. $u_1 \longleftarrow t \longrightarrow u_2$) there exists a term v such that $u_1 \xrightarrow{*} v \xleftarrow{*} u_2$. A confluent rewriting system is always locally confluent, and Newman's lemma [25] ensures the converse implication when the rewriting system is terminating. A rewriting system is *convergent* when it is both terminating and confluent. In this case, any maximal sequence of rewriting steps starting from a term t will end on the same term $\hat{t}$, called the *normal form* of t, and two terms t and t' are such that $t \xleftrightarrow{*} t'$ if and only if $\hat{t} = \hat{t}'$: normal forms provide canonical representatives of equivalence classes under the equivalence relation $\xleftrightarrow{*}$.

Critical pairs

Local confluence of a rewriting system can be tested by considering minimal obstructions to confluence. Generalizing the above notion of context, a *context with two holes* E is a term using usual variables as well as $\square$ and $\square'$, in which both $\square$ and $\square'$ occur exactly once; we write $E[t, t']$ instead of $E[t/\square, t'/\square']$. Consider a pair of rewriting steps $C_1[R_1]v_1 : t \longrightarrow u_1$ and $C_2[R_2]v_2 : t \longrightarrow u_2$, with $R_i : t'_i \Rightarrow u'_i$, rewriting the same term t. The pair of rewriting steps is *non-overlapping* when there exists a context E with two holes such that $C_1 = E[\square, t'_2 \circ v_2]$ and $C_2 = E[t'_1 \circ v_1, \square]$. In this situation, the two reductions are always confluent:

$$\begin{tikzcd}
& t = E[t'_1 \circ v_1, t'_2 \circ v_2] \arrow[dl, "C_1{[R_1]}v_1"'] \arrow[dr, "C_2{[R_2]}v_2"] & \\
u_1 = E[u'_1 \circ v_1, t'_2 \circ v_2] \arrow[dr, "*"'] & & E[t'_1 \circ v_1, u'_2 \circ v_2] = u_2 \arrow[dl, "*"] \\
& \hat{t} = E[u'_1 \circ v_1, u'_2 \circ v_2] &
\end{tikzcd} \tag{2}$$

Given a pair of rewriting steps as above, a context (C, v) induces another pair of rewriting steps rewriting the same terms: $(C \circ C_i)[R](v_i \circ v) : C[t]v \Rightarrow C[u_i]v$. In this case, we say that the former pair is smaller than the latter, and this induces a partial order on pairs of rewriting steps rewriting the same term.

▶ **Definition 3.** A pair of rewriting steps $C_1[R_1]v_1 : t \longrightarrow u_1$ and $C_2[R_2]v_2 : t \longrightarrow u_2$ rewriting the same term t is *critical* when the two steps are distinct, i.e. $(C_1, R_1, v_1) \neq (C_2, R_2, v_2)$, overlapping, and minimal wrt the above partial order. It is *confluent* when there exists a term v such that $u_1 \xleftarrow{*} v \xrightarrow{*} u_2$.

This reformulates with our formalism the classical notion of critical pair, and the usual associated lemma holds: a rewriting system is locally confluent if and only if all its critical pairs are confluent. In particular, a terminating rewriting system with confluent critical pairs is convergent.

2.2 Lawvere algebraic theories

A rewriting system P induces a category, noted $\overline{\mathsf{P}^*}$ and called the *category presented by the rewriting system*, defined as the quotient of the category of terms P_1^* under the congruence $\xleftrightarrow{*}$ generated by the rules: given a tuple of terms $t \in P_1^*$, we write $\overline{t}$ (or sometimes even simply t) for its equivalence class. This presented category is easily shown to be a a Lawvere theory [13]:

▶ **Definition 4.** A *Lawvere theory* $\mathcal{T}$ (also sometimes called an *algebraic theory*) is a category with finite products whose objects are natural numbers, products are given on objects by addition, and the terminal object is 0.

In particular, when the rewriting system has no rules, the associated Lawvere theory is P_1^* and called the *Lawvere theory freely generated by the signature.* It can in fact be shown to correspond to a left adjoint to the suitable forgetful functor from Lawvere theories to signatures (for space constraints, we do not detail this construction nor even morphisms of signatures and Lawvere theories because they do not play an important rôle here).

▶ **Lemma 5.** *Any Lawvere theory $\mathcal{T}$ admits a presentation* P*, called the* standard presentation*, with*

$$\mathsf{P}_1 = \coprod_{n\in\mathbb{N}} \mathcal{T}(n,1) \qquad\qquad \mathsf{P}_2 = \{t \Rrightarrow u \mid t,u \in \mathsf{P}_1^* \text{ and } \epsilon(t) = \epsilon(u)\}$$

where we take all morphisms of $\mathcal{T}$ of coarity 1 *as operations in* P_1*, with the expected arity, and write $\epsilon : \mathsf{P}_1^* \to \mathsf{P}_1$ for the morphism which to a term, seen as a formal composite of morphisms in $\mathcal{T}$, associates the result of its compositions. The rules are thus all pairs of formal composites whose result is the same.*

A *model* of a Lawvere theory $\mathcal{T}$ is a functor $\mathcal{T} \to \mathbf{Set}$ which preserves finite products. In the case where $\mathcal{T}$ is presented by a rewriting system P, this amounts to the specification of a set X, of a function $[\![f]\!] : X^n \to X$ for each operation f of arity n, in such a way that $[\![t]\!] = [\![u]\!]$ for each rule $R : t \Rrightarrow u$.

▶ **Example 6.** A model for the theory of monoids (Example 1) consists of a set X together with functions $[\![m]\!] : X \times X \to X$ and $[\![e]\!] : 1 \to X$ such that for every $x, y, z \in X$, $[\![m]\!]([\![m]\!](x,y),z) = [\![m]\!](x,[\![m]\!](y,z))$, $[\![m]\!]([\![e]\!](),x) = x = [\![m]\!](x,[\![e]\!]())$. The models for this theory are thus precisely monoids in the usual sense. Similarly, the models for the theory of groups (Example 2) are groups.

2.3 Tietze transformations

Two rewriting systems are *Tietze equivalent* when they present isomorphic Lawvere theories, which implies that they have the same models (in fact, the converse is also true). For instance, the theory of groups can be presented by the rewriting system of Example 2. As explained in the introduction, it also admits a presentation with two generators d of arity 2 and a of arity 0, with one rewriting rule corresponding to the equation (1). In the context of presentations of groups, Tietze has shown that the corresponding equivalence is generated by two transformations and their inverse [31]. This property can be adapted to the context of presentation of Lawvere theories as follows.

▶ **Definition 7.** The *Tietze transformations* are the two following operations, transforming a rewriting system P into another one P', as well as their converse (transforming P' into P):

1. *adding a superfluous operation*: given a symbol f not occurring in P_1, a symbol R not occurring in P_2, and a term $t \in \mathsf{P}_1^*$ of arity n, we set

$$\mathsf{P}'_1 = \mathsf{P}_1 \uplus \{f\} \qquad\qquad \mathsf{P}'_2 = \mathsf{P}_2 \uplus \{R\}$$

where f is an operation of arity n and $R : t \Rrightarrow f(x_1,\ldots,x_n)$,

2. *adding a derivable relation*: given a symbol R not occurring in P_2 and two terms t, u such that $t \overset{*}{\longleftrightarrow}_P u$, we set

$$\mathsf{P}'_1 = \mathsf{P}_1 \qquad\qquad \mathsf{P}'_2 = \mathsf{P}_2 \uplus \{R\}$$

with $R : t \Rightarrow u$.

▶ **Proposition 8.** *Two rewriting systems* P *and* Q *are* Tietze equivalent *if and only if* Q *can be obtained from* P *by applying a series of Tietze transformations.*

In our quest for minimizing the number of relations (and generators) of a Lawvere theory, the Tietze transformations can be helpful, as illustrated in the following simple example.

▶ **Example 9.** Consider the string rewriting system with generators a, b and c of arity one and two rules:

$$\mathsf{P} \quad = \quad \langle a:1, b:1, c:1 \mid A : a(x_1) \Rightarrow x_1, B : a(b(x_1)) \Rightarrow c(x_1)\rangle$$

We can then apply the following sequence of Tietze transformations:

$$\begin{aligned}
\mathsf{P}' &= \langle a:1, b:1, c:1 \mid A : a(x_1) \Rightarrow x_1, B : a(b(x_1)) \Rightarrow c(x_1), C : b(x_1) \Rightarrow c(x_1)\rangle \\
\mathsf{P}'' &= \langle a:1, b:1, c:1 \mid A : a(x_1) \Rightarrow x_1, C : b(x_1) \Rightarrow c(x_1)\rangle \\
\mathsf{P}''' &= \langle b:1, c:1 \mid C : b(x_1) \Rightarrow c(x_1)\rangle \\
\mathsf{P}'''' &= \langle c:1 \mid \rangle
\end{aligned}$$

We have first added the derivable relation C, then removed the derivable relation B, then removed the definable operation a, then removed the definable operation b. So, in fact, our theory can be presented without any relation and only one operation.

It is clear that, in above example, we had to first add a new relation in order to remove all of them: one cannot simply hope to always reduce the number of relations by Tietze transformations in order to obtain a minimal one (and for similar reasons, one might be forced to add new generators before reducing the presentation, as illustrated for the theory of groups in the introduction). For this reason, it is quite difficult to minimize the number of relations in general, or to decide whether a presentation is minimal wrt to relations or generators.

Note in particular that, in a convergent rewriting system, a critical pair witnesses a derivable relation, and Newman's lemma ensures that any derivable relation can be obtained via critical pairs: if a presentation has a removable relation, then such a relation can be obtained by inspecting critical pairs.

3 Homology of Lawvere algebraic theories

In this section, we introduce the notion of homology of a Lawvere theory by adapting the general methodology which is now classical for monoids, groups, algebras [15], operads [14], etc. This construction associates to a Lawvere theory a sequence of groups which are invariants of the Lawvere theory: we will see that these can be computed from any presentation with suitable properties, however these groups only depend on the presented theory, and not on the presentation. It thus provides interesting information about all the possible presentations of the theory: its relevance will be illustrated in Section 3.6, where we use it to show that a particular theory admits no presentation with only one rule, *whichever possible signature we use.*

The basic idea of homology is to "count" the number of times a thing is used (positively when it occurs in the target and negatively in the source). For example, a rule $R : g(f(x_1), f(x_1)) \Rightarrow h(x_1)$ "consumes" two instances of f and one of g to "produce" one of h. Therefore, we can think of the associated balance to be $h - 2f - g$. Since this is a relation, it induces the equation $h = 2f + g$ when counting operations, which indicates that the operation h might be superfluous, i.e. we might be able to remove it using a Tietze transformation. A similar process for critical pairs will allow us to provide an "over-approximation" of the superfluous relations, and therefore give lower bounds on necessary relations.

Note that above, we formally consider the "ring" of operations (actually a "ringoid" since operations are typed by their arities) in order to be able to consider sums of operations. Much care is however needed in order to ensure that this way of counting is compatible with duplication and erasure of variables, and independent of the presentation. As customary in homological algebra, we thus begin by introducing the notion of resolution for a Lawvere theory, which is easily shown to be invariant (in a suitable sense) under Tietze equivalences and derive homology from those. Roughly, the resolution amounts to perform a similar linearization process as above, but keeping track of the contexts, i.e. the rule R would give rise to a relation of the form $\underline{h}(x_1) - \underline{g}(f(x_1), f(x_1)) - g(\underline{f}(x_1), f(x_1)) + g(f(x_1), \underline{f}(x_1))$, and to ensure that all (higher-)relations are present.

One could be tempted to use standard notions of homology for a category in order to study Lawvere theories. However, because such a theory contains a terminal object, its homology in this sense will always be trivial. Therefore, one has to adapt the setting of homology in order to take in account the cartesian structure. Following the general methodology of Barr and Beck [3, 2], Jibladze and Pirashvili have been able to define a suitable ringoid of coefficients for cohomology [8, 9], which was later on reworked by Malbos [16]. The section 3.1 to 3.4 are a reformulation, in operational terms, of those (to simplify the presentation, the framework is also less general: we use bimodules instead of cartesian natural systems). We suppose fixed a rewriting system P and write $\mathcal{T} = \overline{\mathsf{P}^*}$ for the theory it presents.

3.1 Modules over ringoids

Ringoids

A monoid is the same as a category with only one object, or thinking backward, a category is a "monoid with multiple objects". Similarly, a ringoid can be thought of as a "ring with multiple objects". We briefly introduce here this algebraic structure and refer the reader to seminal paper [22] for details. The category $\mathbf{Ab}$ of abelian groups is monoidal when equipped with the usual tensor product $\otimes$ of abelian groups, with $(\mathbb{Z}, +, 0)$ as unit (in the following, we always denote abelian groups additively).

▶ **Definition 10.** A *ringoid* $\mathcal{R}$ is a small category enriched in the monoidal category $\mathbf{Ab}$.

More explicitly, a ringoid consists of a category $\mathcal{C}$ in which each hom-set $\mathcal{C}(A, B)$ is equipped with a structure of abelian group, in such a way that composition is bilinear, i.e. respects addition and zero. For instance, given $f, f' : A \to B$ and $g, g' : B \to C$, we have

$$(g + g') \circ (f + f') = g \circ f + g \circ f' + g' \circ f + g' \circ f' \qquad 0 \circ f = 0 \qquad f \circ 0 = 0 \tag{3}$$

▶ **Lemma 11.** *The category of ringoids with one object is equivalent to the category of rings.*

Any category $\mathcal{C}$ freely generates a ringoid that we denote $\mathbb{Z}\mathcal{C}$. It always exists for general arguments [1] and can be explicitly described as follows. It has the same objects as $\mathcal{C}$ and, given objects A and B, $\mathbb{Z}\mathcal{C}(A,B)$ is the free abelian group over $\mathcal{C}(A,B)$, which is the same as the free $\mathbb{Z}$-module, i.e. formal sums of morphisms in $\mathcal{C}(A,B)$ with coefficients in $\mathbb{Z}$, quotiented by the usual axioms of groups, and composition is induced by the one of $\mathcal{C}$ and satisfies the axioms of ringoids such as (3).

Modules

The usual notion of module over a ring, can also easily be generalized to "multiple objects" as follows.

▶ **Definition 12.** A *(left) module* $\mathcal{M}$ over a ringoid $\mathcal{R}$, or $\mathcal{R}$*-module*, is a functor $\mathcal{M} : \mathcal{R} \to \mathbf{Ab}$ which is enriched in $\mathbf{Ab}$. A morphism $f : \mathcal{M} \to \mathcal{N}$ of $\mathcal{R}$-modules, or $\mathcal{R}$*-linear map*, is an enriched natural transformation: it consists of a group morphism $f_A : \mathcal{M}A \to \mathcal{N}A$ for every object A of $\mathcal{R}$, satisfying naturality conditions. We write $\mathbf{Mod}(\mathcal{R})$ for the category of $\mathcal{R}$-modules.

A right $\mathcal{R}$-module is defined as a left $\mathcal{R}^{\mathrm{op}}$-module, which explains why we will only need to consider left modules in the following. More explicitly, an $\mathcal{R}$-module $\mathcal{M}$ consists of an abelian group $\mathcal{M}A$ for every object A of $\mathcal{R}$, and a morphism $\mathcal{M}f : \mathcal{M}A \to \mathcal{M}B$ of groups for every morphism $f : A \to B$ in such a way that $\mathcal{M}(f + f') = \mathcal{M}f + \mathcal{M}f'$ (we are considering the pointwise addition on the right) and $\mathcal{M}0 = 0$ (on the right, 0 is the constant map). The category $\mathbf{Mod}(\mathcal{R})$ is enriched in $\mathbf{Ab}$ and can be shown to have enough structure to support usual computations in homological algebra: it is abelian and has enough projectives [22].

Free modules

Suppose given a set X_A for every object A of $\mathcal{R}$. The *free module* generated by this family of sets, written $\mathcal{R}\underline{X}$, can be described as the functor which to every object A associates the formal finite sums $\sum_i f_i \underline{x}_i$, with $x_i \in X_{A_i}$ and coefficients $f_i : A_i \to A$, subject to the usual laws of left modules, e.g.

$$g\left(\sum_i f_i \underline{x}_i\right) = \sum_i (g \circ f_i)\underline{x}_i \qquad g \circ 0 = 0 \qquad \left(\sum_i g_i\right)(f\underline{x}) = \sum_i (g_i \circ f)\underline{x} \qquad 0(f\underline{x}) = 0$$

Above, the "underline" notation is here only to make the distinction between the elements of $\mathcal{R}$ and those of X, and as customary we write $g(f\underline{x})$ instead of $((\mathcal{R}\underline{X})g)(f\underline{x})$ for the left action.

Tensor product of modules

The usual definition of the tensor product of modules can be generalized to modules over ringoids as follows. Given a right $\mathcal{R}$-module $\mathcal{M} : \mathcal{R}^{\mathrm{op}} \to \mathbf{Ab}$ and a left $\mathcal{R}$-module $\mathcal{M} : \mathcal{R} \to \mathbf{Ab}$, their tensor product is the ringoid defined by the (enriched) coend $\mathcal{M} \otimes \mathcal{N} = \int^A \mathcal{M}A \otimes \mathcal{N}A$. This means that an element of $(\mathcal{M} \otimes \mathcal{N})(B)$ is a quotient of $\bigoplus_{A \in \mathcal{R}} \mathcal{M}A \otimes \mathcal{N}A$ by the relation identifying elements of the form $(f^{\mathrm{op}}x) \otimes y$ and $x \otimes (fy)$, for any suitably typed morphism f of $\mathcal{R}$.

3.2 The ringoid of bicontexts

The ringoid we will be mainly interested in is a quotient of $\mathbb{Z}\mathbb{K}$, the free ringoid over bicontexts. We begin by first defining a similar quotient on contexts.

▶ **Definition 13.** We define $\kappa_i : \mathbb{Z}\mathsf{P}_1^* \to \mathbb{Z}\mathcal{K}$ as the linear map which sends a term t to the formal sum of occurrences of the variable x_i in t. Formally, it is defined, for $j \neq i$ and $t = \langle t_1, \ldots, t_n \rangle$, by

$$\kappa_i(x_i) = \square \qquad\qquad \kappa_i(x_j) = 0 \qquad\qquad \kappa_i(u \circ t) = \sum_{j \in \mathrm{FV}(u)} (\kappa_j(u)t)[\kappa_i(t_j)]$$

On the right, the notations for contexts introduced in Section 2.1 are implicitly extended by linearity, e.g. $(C + D)[t] = C[t] + D[t]$, $C[t + u] = C[t] + C[u]$, etc.

▶ **Example 14.** Consider the term $t = f(g(x_1, x_2), x_1)$. We have

$$\kappa_1(t) = f(g(\square, x_2), x_1) + f(g(x_1, x_2), \square) \qquad \kappa_2(t) = f(g(x_1, \square), x_1) \qquad \kappa_3(t) = 0$$

We write $\overline{\mathbb{Z}\mathcal{K}}$ for the quotient of $\mathbb{Z}\mathcal{K}$ by the ideal generated by all elements of the form $\kappa_i(u) - \kappa_i(t)$ for a rule $R : t \Rightarrow u$ of arity n and $1 \leq i \leq n$; we thus have a well-defined quotient morphism $\kappa_i : \mathbb{Z}\overline{\mathsf{P}_1^*} \to \overline{\mathbb{Z}\mathcal{K}}$. The ringoid of bicontexts $\overline{\mathbb{Z}\mathbb{K}}$ is defined as the quotient of the free ringoid $\mathbb{Z}\mathbb{K}$ by quotienting contexts as above and morphisms by the rewriting rules: we identify element $\sum_i n_i(C_i, u)$ to 0 whenever $\sum_i n_i C_i = 0$ in $\overline{\mathbb{Z}\mathcal{K}}$, and $\sum_i n_i(C, u_i)$ to 0 whenever $\sum_i n_i u_i = 0$ in $\mathbb{Z}\mathcal{C}$.

▶ **Example 15.** Consider the rewriting system with operations and arities $a : 0$, $b : 0$, $f : 1$, $g : 2$, and two rules $A : a \Rightarrow b$ and $B : f(x_1) \Rightarrow g(x_1, x_1)$. The quotient on contexts is generated by $g(\square, x_1) + g(x_1, \square) - f(\square)$.

In the rest of the paper, we write $\mathcal{R} = \overline{\mathbb{Z}\mathbb{K}}$ for the ringoid of bicontexts of $\mathcal{T}$, which will be where coefficients will be taken in. In a free module, of the form $\mathcal{R}\underline{X}$, the elements are sums of monomials of the form $(C, u)\underline{x}$ where (C, u) is an equivalence class of bicontexts and $x \in X_n$ for some $n \in \mathbb{N}$. In the following, we will adopt the notation $C\underline{x}u$ instead: this makes it clear that contexts $C \in \overline{\mathbb{Z}\mathcal{K}}_n$ are acting on the left and morphisms in $\mathcal{T}$ are acting on the right (since their composition is reversed in the composition of bicontexts). In fact, the definition of $\overline{\mathcal{R}}$ does not depend on the choice of the presentation P, but only on the presented theory $\mathcal{T}$. Since every Lawvere theory admits a presentation (Lemma 5), the notions developed here will apply to any theory.

▶ **Lemma 16.** *Given two Tietze equivalent rewriting systems* P *and* Q*, the ringoids* $\mathcal{R}^{\mathsf{P}}$ *and* $\mathcal{R}^{\mathsf{Q}}$ *are isomorphic.*

3.3 Resolutions for Lawvere algebraic theories

The *trivial* $\mathcal{R}$*-module* $\mathcal{Z} : \mathcal{R} \to \mathbf{Ab}$ is the quotient of the free $\mathcal{R}$-module $\mathcal{R}\underline{X}$, with $X_n = \{\star_n\}$ for $n \in \mathbb{N}$, quotiented by relations of the form $\sum_i \kappa_i(u)t\underline{\star}t_i = \underline{\star_n}$ for every term $u \circ t$ of arity n (we write $\star$ instead of $\star_1$).

▶ **Example 17.** Given a signature with a binary operation m, since $m \circ \mathrm{id}_2 = \mathrm{id} \circ \langle m(x_1, x_2) \rangle$, we have the following relation in $\mathcal{Z}$: $m(\square, x_2)\underline{\star}\,\langle x_1 \rangle + m(x_1, \square)\underline{\star}\,\langle x_2 \rangle = \underline{\star_2} = \underline{\star}m(x_1, x_2)$.

The general idea of a free resolution is to start with the trivial module $\mathcal{Z}$ and equip it with a sequence of free $\mathcal{R}$-modules $\mathcal{M}_i$ such that $\mathcal{M}_0$ contains the sorts of the theory (there is always only one in our setting), $\mathcal{M}_1$ the operations, $\mathcal{M}_2$ the relations, $\mathcal{M}_3$ the relations between relations, and so on:

▶ **Definition 18.** A *free resolution* $\mathcal{M}_\bullet$ of $\mathcal{Z}$ consists of a sequence

$$\begin{tikzcd} \cdots \arrow[r, "\partial_1"] & \mathcal{M}_1 \arrow[r, "\partial_0"] & \mathcal{M}_0 \arrow[r, "\varepsilon"] & \mathcal{Z} \arrow[r] & 0 \end{tikzcd}$$

of free $\mathcal{R}$-modules $\mathcal{M}_i = \mathcal{R}\underline{X_i}$ and $\mathcal{R}$-linear maps ∂_i and ε such that for any two successive arrows the image of the first is equal to the kernel of the second: $\operatorname{im}\partial_{i+1} = \ker\partial_i$, $\operatorname{im}\partial_0 = \ker\varepsilon$ and $\operatorname{im}\varepsilon = \mathcal{Z}$. The resolution is *partial* when it is finite on the left.

Note that the relation $\operatorname{im}\varepsilon = \mathcal{Z}$ means that ε is surjective and therefore $\mathcal{M}_0$ is free on at least one generator. Suppose that the theory contains an operation, for instance m as in Example 17: the kernel of ε will contain $m(\Box, x_2)\underline{1}\langle x_1\rangle + m(x_1,\Box)\underline{1}\langle x_2\rangle - \underline{1}m(x_1,x_2)$ as non-trivial element, and therefore $\mathcal{M}_1$ will need to contain a generator for m in order for the relation $\operatorname{im}\partial_0 = \ker\varepsilon$ to be satisfied. More generally, $\mathcal{M}_1$ should be free on a set of operations generating $\mathcal{T}$. For similar reasons, $\mathcal{M}_2$ should be free on a set of elements generating all the relations of $\mathcal{T}$, and $\mathcal{M}_3$ should be free on enough generators so that any two relations between the same (linearized) terms should be equal modulo them.

A major interest of free resolutions is that they can be shown to be essentially unique:

▶ **Proposition 19.** *Any two free resolutions of $\mathcal{Z}$ are homotopy equivalent.*

We do not detail further here the meaning of the above classical equivalence. Its main interest is that it will enable us to show that the definition of homology makes sense in next section (Proposition 21).

3.4 Homology of Lawvere algebraic theories

We are now in position to introduce the notion of homology of a Lawvere theory. The chain complex $(\mathcal{M}_\bullet, \partial_\bullet)$ of a resolution by $\mathcal{R}$-modules is acyclic, that is $\operatorname{im}\partial_{i+1} = \ker\partial_i$ holds. We are going to tensor it by $\mathcal{Z}^{\mathrm{op}}$, which means that we "erase" the coefficients in $\mathcal{R}$ everywhere, e.g. if $\mathcal{M}_i$ is free on the set X_i (i.e. $\mathcal{M}_i = \mathcal{R}\underline{X_i}$) then we have $\mathcal{Z}^{\mathrm{op}} \otimes \mathcal{M}_i = \mathbb{Z}X_i$. The resulting chain complex $(\mathcal{Z}^{\mathrm{op}} \otimes \mathcal{M}_\bullet, \tilde{\partial}_\bullet)$ still satisfies $\operatorname{im}\tilde{\partial}_{i+1} \subseteq \ker\tilde{\partial}_i$, but the converse inclusion is not true anymore in general. It thus makes sense to consider the following homology groups:

▶ **Definition 20.** Suppose given a Lawvere theory $\mathcal{T}$ and a resolution of the associated trivial $\mathcal{R}$-module $\mathcal{Z}$ as in Definition 18. The *homology* $H_\bullet(\mathcal{T})$ of $\mathcal{T}$ (with coefficients in the trivial $\mathcal{R}$-module $\mathcal{Z}$) is the homology of the chain complex

$$\begin{tikzcd} \cdots \arrow[r, "\tilde{\partial}_2"] & \mathcal{Z}^{\mathrm{op}} \otimes \mathcal{M}_2 \arrow[r, "\tilde{\partial}_1"] & \mathcal{Z}^{\mathrm{op}} \otimes \mathcal{M}_1 \arrow[r, "\tilde{\partial}_0"] & \mathcal{Z}^{\mathrm{op}} \otimes \mathcal{M}_0 \end{tikzcd}$$

where $\tilde{\partial}_i = \mathcal{Z}^{\mathrm{op}} \otimes \partial_i$. More explicitly, the homology consists of a sequence $H_\bullet(\mathcal{T})$ of groups defined by $H_i(\mathcal{T}) = \ker\tilde{\partial}_{i-1} / \operatorname{im}\tilde{\partial}_i$ where, by convention, $\tilde{\partial}_{-1}$ is the constant null map.

Proposition 19 ensures that it is well defined, because two homotopic chain complexes will give rise to the same homology:

▶ **Proposition 21.** *The homology $H_\bullet(\mathcal{T})$ of a Lawvere theory $\mathcal{T}$ does not depend on the choice of the resolution.*

Any theory can be shown to admit a resolution, called the *standard resolution*, by easily adapting usual constructions performed for monoids. More generally any partial resolution can be extended into a full one. We do not detail it here however, because it involves modules of infinite rank, and difficult to work with: in order to actually compute the homology of a theory, one should start with a resolution which is reasonably small. The purpose of next section is to construct such a (partial) resolution in the case where we start from a convergent presentation of the Lawvere theory.

3.5 A partial resolution for convergent Lawvere theories

In this section, we suppose that the Lawvere theory $\mathcal{T}$ we are considering is presented by a convergent reduced rewriting system P, and construct from it a partial free resolution of the trivial $\mathcal{R}$-module $\mathcal{Z}$. Writing $\mathsf{P}_0 = \{1\}$ for the set with one element and P_3 for the set of critical pairs of the rewriting system, the resolution we consider is of the form

$$\begin{tikzcd} \mathcal{R}\underline{\mathsf{P}_3} \arrow[r, "\partial_2"] & \mathcal{R}\underline{\mathsf{P}_2} \arrow[r, "\partial_1"] & \mathcal{R}\underline{\mathsf{P}_1} \arrow[r, "\partial_0"] & \mathcal{R}\underline{\mathsf{P}_0} \arrow[r, "\varepsilon"] & \mathcal{Z} \arrow[r] & 0 \end{tikzcd} \tag{4}$$

where the maps are defined as follows, and will be illustrated in next section. The map $\varepsilon : \mathcal{R}\underline{\mathsf{P}_0} \to \mathcal{Z}$ is the $\mathcal{R}$-linear map preserving the unit, i.e. such that $\varepsilon(\underline{1}) = \underline{\star}$. More generally, because of relations defining $\mathcal{Z}$, we have $\varepsilon(C\underline{1}u) = \underline{\star_n}$. The map $\partial_0 : \mathcal{R}\underline{\mathsf{P}_1} \to \mathcal{R}\underline{\mathsf{P}_0}$ is the $\mathcal{R}$-linear map such that for each operation $f \in \mathsf{P}_1$ of arity n, we have

$$\partial_0(\underline{f}) \quad = \quad \left(\sum_{i=1}^{n} \kappa_i(f)\underline{1}\langle x_i\rangle\right) - \underline{1}\langle f\rangle$$

The map $\partial_1 : \mathcal{R}\underline{\mathsf{P}_2} \to \mathcal{R}\underline{\mathsf{P}_1}$ is the $\mathcal{R}$-linear map such that for each rule $R : t \Rightarrow u$ in P_2 we have

$$\partial_1(R) \quad = \quad \underline{u} - \underline{t}$$

where the notation $\underline{t}$ generalizes the notation $\underline{f}$ for operations, and is defined inductively by

$$\underline{u \circ t} \quad = \quad \underline{u}t + \sum_{i=1}^{n}(\kappa_i(u)t)[\underline{t_i}] \qquad\qquad \underline{\mathrm{id}} = 0$$

for $t = \langle t_1, \ldots, t_n\rangle$. The map $\partial_2 : \mathcal{R}\underline{\mathsf{P}_3} \to \mathcal{R}\underline{\mathsf{P}_2}$ is the $\mathcal{R}$-linear map such that the image of a critical pair $(C_1[R_1]v_1, C_2[R_2]v_2)$, with $C_i[R_1]v_i : t \longrightarrow u_i$, is

$$\partial_2(C_1[R_1]v_1, C_2[R_2]v_2) \quad = \quad C_2\underline{R_2}v_2 - C_1\underline{R_1}v_1 + \underline{S_2} - \underline{S_1} \tag{5}$$

where $S_i : u_i \xrightarrow{*} \hat{t}$ are a choice of rewriting paths from u_i to $\hat{t}$, the normal form of t, see (2), which exist because the rewriting system is supposed to be convergent. Again, writing $\cdot$ for the concatenation of rewriting paths and Id for the empty one, the notation $\underline{T}$ is extended to rewriting paths by

$$\underline{C[R]v} = C\underline{R}v \qquad\qquad \underline{T' \cdot T} = \underline{T'} + \underline{T} \qquad\qquad \underline{\mathrm{Id}} = 0$$

The main result of this article is the following one:

▶ **Theorem 22.** *The sequence* (4) *as defined above is a partial free resolution of the trivial $\mathcal{R}$-module $\mathcal{Z}$.*

This theorem allows us to explicitly compute low-dimensional homology of a theory with a convergent presentation. Moreover, since the homology is independent of the choice of the presentation (Proposition 21), and any partial resolution can be extended into a full one (Section 3.4), it provides us with invariants for any presentation of $\mathcal{T}$. In particular, since $H_1(\mathcal{T})$ is defined as a quotient of $\mathbb{Z}\mathsf{P}_1$, and similarly for $H_2(\mathcal{T})$, we have

▶ **Proposition 23.** *The rank of $H_1(\mathcal{T})$ (resp. $H_2(\mathcal{T})$) is a lower bound of the number of operations (resp. relations) in any presentation of $\mathcal{T}$.*

3.6 An example

Let us illustrate the previous definitions on a simple example. Consider the term rewriting system with two generators f and g of arity 2, three generators a, b and c of arity 0 and four rules

$$\begin{array}{lcllcl} A & : & f(a,x_1) \Rightarrow g(a,x_1) & A' & : & f(x_1,a) \Rightarrow g(x_1,a) \\ B & : & f(b,b) \Rightarrow g(b,b) & C & : & f(c,c) \Rightarrow g(c,c) \end{array}$$

The rewriting system is locally confluent, the only critical pair being

$$\begin{tikzcd} f(a,a) \arrow[d, bend right, "{[A]\langle a\rangle}"'] \arrow[d, bend left, "{[A']\langle a\rangle}"] \arrow[d, phantom, "\Phi"] \\ g(a,a) \end{tikzcd}$$

It is also terminating, because all the rules decrease the number of occurrences of f in terms, and thus convergent. Therefore we can construct a resolution (4), as described in Section 3.5. The boundary maps are given on operations by

$$\begin{array}{ll} \partial_0(\underline{f}) = f(\square,x_2)\underline{1}\,\langle x_1\rangle + f(x_1,\square)\underline{1}\,\langle x_2\rangle - \underline{1}\,\langle f(x_1,x_2)\rangle & \partial_0(\underline{a}) = -\underline{1}\,\langle a\rangle \\ \partial_0(\underline{g}) = g(\square,x_2)\underline{1}\,\langle x_1\rangle + g(x_1,\square)\underline{1}\,\langle x_2\rangle - \underline{1}\,\langle g(x_1,x_2)\rangle & \partial_0(\underline{b}) = -\underline{1}\,\langle b\rangle \\ & \partial_0(\underline{c}) = -\underline{1}\,\langle c\rangle \end{array}$$

on relations by

$$\begin{aligned} \partial_1(\underline{A}) &= \underline{g}\,\langle a,x_1\rangle + g(\square,x_1)\underline{a} - \underline{f}\,\langle a,x_1\rangle - f(\square,x_1)\underline{a} \\ \partial_1(\underline{A'}) &= \underline{g}\,\langle x_1,a\rangle + g(x_1,\square)\underline{a} - \underline{f}\,\langle x_1,a\rangle - f(x_1,\square)\underline{a} \\ \partial_1(\underline{B}) &= \underline{g}\,\langle b,b\rangle + g(\square,b)\underline{b} + g(b,\square)\underline{b} - \underline{f}\,\langle b,b\rangle - f(\square,b)\underline{b} - f(b,\square)\underline{b} \\ \partial_1(\underline{C}) &= \underline{g}\,\langle c,c\rangle + g(\square,c)\underline{c} + g(c,\square)\underline{c} - \underline{f}\,\langle c,c\rangle - f(\square,c)\underline{c} - f(c,\square)\underline{c} \end{aligned}$$

and on critical pairs by

$$\partial_2(\underline{\Phi}) = \underline{A'}\,\langle a\rangle - \underline{A}\,\langle a\rangle$$

The homology is the one of the chain complex obtained by tensoring with $\mathcal{Z}^{\mathrm{op}}$, which amounts to "erase contexts", i.e. all symbols which are not elements of $\mathbb{Z}$ or underlined:

$$\begin{tikzcd} \ldots \arrow[r] & \mathbb{Z}\underline{\{\Phi\}} \arrow[r, "\tilde{\partial}_2"] & \mathbb{Z}\underline{\{A,A',B,C\}} \arrow[r, "\tilde{\partial}_1"] & \mathbb{Z}\underline{\{f,g,a,b,c\}} \arrow[r, "\tilde{\partial}_0"] & \mathbb{Z} \end{tikzcd}$$

above, $\mathbb{Z}\underline{X}$ denotes the free abelian group (or equivalently $\mathbb{Z}$-module) on a set X and the linear maps are defined by

$$\begin{aligned} &\tilde{\partial}_0(\underline{f}) = \tilde{\partial}_0(\underline{g}) = \tilde{\partial}_0(\underline{a}) = \tilde{\partial}_0(\underline{b}) = \tilde{\partial}_0(\underline{c}) = 0 \\ &\tilde{\partial}_1(\underline{A}) = \tilde{\partial}_1(\underline{A'}) = \tilde{\partial}_1(\underline{B}) = \tilde{\partial}_1(\underline{C}) = \underline{g} - \underline{f} \\ &\tilde{\partial}_2(\underline{\Phi}) = \underline{A'} - \underline{A} \end{aligned}$$

Therefore the homology groups are

$$\begin{array}{lclcl} H_0 & = & \mathbb{Z}\underline{\{1\}}/(\underline{1}) & = & 0 \\ H_1 & = & \mathbb{Z}\underline{\{f,g,a,b,c\}}/(\underline{g}-\underline{f}) & = & \mathbb{Z}^4 \\ H_2 & = & \mathbb{Z}\underline{\{A'-A,B-A,C-A\}}/(\underline{A'}-\underline{A}) & = & \mathbb{Z}^2 \end{array}$$

The homology groups are of the form $\mathbb{Z}^{r_i}$ and their rank is r_i. From $r_1 = 4$, we deduce that any presentation of the theory will have at least four generating operations, and from $r_2 = 2$ that any presentation has at least two relations: it is thus not one-based.

4 Coherent presentations

A resolution of a Lawvere theory is obtained by an "abelianization" process: in $\mathcal{M}_1$ we only recall which operations in which context are used, but not the order they are used in, similarly for the rules in $\mathcal{M}_2$, etc. This suggests extending the notion of presentation, so that the module $\mathcal{M}_3$ is the abelianization of something too, as we briefly mention.

▶ **Definition 24.** An *extended rewriting system* consists of a rewriting system P together with a set P_3 of *homotopy generators* and two functions $\sigma_2, \tau_2 : \mathsf{P}_3 \to \mathsf{P}_2^\top$:

$$P \quad = \quad \begin{tikzcd} \mathsf{P}_0 \arrow[d, "\iota_0"] & & \mathsf{P}_1 \arrow[d, "\iota_1"] \arrow[dll, shift right, "\sigma_0"'] \arrow[dll, shift left, "\tau_0"] & & \mathsf{P}_2 \arrow[d, "\iota_2"] \arrow[dll, shift right, "\sigma_1"'] \arrow[dll, shift left, "\tau_1"] & & \mathsf{P}_3 \arrow[dll, shift right, "\sigma_2"'] \arrow[dll, shift left, "\tau_2"] \\ \mathsf{P}_0^* & & \mathsf{P}_1^* \arrow[ll, shift right, "\sigma_0^*"'] \arrow[ll, shift left, "\tau_0^*"] & & \mathsf{P}_2^\top \arrow[ll, shift right, "\sigma_1^\top"'] \arrow[ll, shift left, "\tau_1^\top"] & & \end{tikzcd} \qquad \text{such that} \quad \begin{array}{l} \sigma_1^\top \circ \sigma_2 = \sigma_1^\top \circ \tau_2 \\ \tau_1^\top \circ \sigma_2 = \tau_1^\top \circ \tau_2 \end{array}$$

where $\mathsf{P}_0 = \{1\}$, thus $\mathsf{P}_0^* = \mathbb{N}$ as before, and $\mathsf{P}_2^\top$ is the set of 2-cells of the cartesian (2,1)-category freely generated by adding the elements of P_2 as invertible 2-cells to the free cartesian category P_1^*. It is *coherent* when any 2-cells with same source and target are related by the smallest congruence generated by P_3.

Intuitively, in a coherent rewriting system the set P_3 is big enough to relate two possible rewriting paths (or zig-zags) between the same terms. Newman's lemma thus reformulate as follows in this context:

▶ **Lemma 25.** *Given a convergent rewriting system* P, *its extension obtained by taking the set of confluence diagrams induced by critical pairs as* P_3 *is coherent.*

Finally, the abelianization process mentioned above can be formulated as follows:

▶ **Proposition 26.** *To any coherent presentation* P *one can associate a partial free resolution with* $\mathcal{R}\mathsf{P}_i$ *as modules, for* $0 \leq i \leq 3$, *as in* (4).

Notice that by Lemma 25, we recover the construction of Section 3.5 as a particular case. Also, it can be noticed that all the constructions we have been performing are compatible with the laws induced by the cartesian structure on cells (the definitions have in fact been chosen so that this is true).

▶ **Example 27.** Consider the theory $\mathcal{T}$ presented by the following term rewriting system

$$\mathsf{P} \quad = \quad \langle d : 2, t : 0, f : 0 \quad | \quad T : d(t, x_1) \Rightarrow t, \quad T' : d(x_1, t) \Rightarrow t, \quad F : d(f, f) \Rightarrow f \rangle$$

corresponding to the famous *parallel* implementation of the disjunction, sometimes called *por* (d stands for disjunction, t for true and f for false), which was used by Melliès as a central example for standardization [21] (incidentally, this paper notices the similarity with algebraic topology...). There is one confluent critical pair $\begin{tikzcd} d(t,t) \arrow[r, bend left, "{[T]\langle t\rangle}"] \arrow[r, bend right, "{[T']\langle t\rangle}"'] \arrow[r, phantom, "\ominus"] & t \end{tikzcd}$ and thus the presentation can be extended into a coherent one by setting $\mathsf{P}_3 = \{\Theta\}$ with expected source and target. By Proposition 26, we recover the resolution of Section 3.5, and one can easily compute the associated homology: we have $H_0(\mathcal{T}) = 0$, $H_1(\mathcal{T}) = \mathbb{Z}$ and $H_2(\mathcal{T}) = 0$.

The above definition of extended rewriting systems constitutes a generalization of polygraphs to Lawvere theories, using which one can show an analogous of Squier's homotopical theorem [29], which implies the homological one. Burroni's original paper on polygraphs shows that those theories can be described by polygraphs by considering explicitly the cartesian structure (duplications and erasures of variables) [4]. By contrast, this structure is implicit in this work, thus giving rise to much smaller and manipulable rewriting systems. It would be interesting to compare the two resulting homologies though. Also, as in the case of polygraphs, the formulation of Definition 24 should make it clear that this definition can be generalized in any dimension. Finally, we should mention that in the case of presentations of monoids, Tietze transformations and completion procedures can be generalized to coherent presentations [5]; we expect that similar constructions can be performed in the setting developed in this paper.

5 Extensions and future work

In conclusion, we would like to mention some other possible generalizations of this work, which we plan to investigate and detail in future work. For instance, the generalization to term rewriting systems with multiple sorts is easy (it roughly consists in allowing P_0 to contain multiple elements and use P_0^* instead of $\mathbb{N}$ for the objects of our ringoids).

One should be able to continue the resolution in higher dimension, as done for monoids [11], by using critical n-uples for P_{n+1}. In particular, the next dimension of the resolution can easily be done and allows to compute $H_3(\mathcal{T})$ for a theory $\mathcal{T}$, whose rank provides a lower bound on the number of critical pairs of any convergent presentation of $\mathcal{T}$. Consider the rewriting system with generators i, h and k of arity one, generators a and f of arity two, and three rules

$$h(a(x_1,x_2)) \Rrightarrow a(f(x_1,x_1),x_2) \quad k(a(x_1,x_2)) \Rrightarrow a(f(x_1,x_1),x_2) \quad a(f(i(x_1),i(x_1)),x_2) \Rrightarrow a(x_1,x_2)$$

One can compute that $H_3(\mathcal{T})$ is not finitely generated, showing that it cannot be presented by a finite convergent rewriting system (since any such would have a finite number of critical pairs), even though this theory $\mathcal{T}$ has a decidable equality. This generalizes to terms rewriting systems Squier's example for monoids [28].

Computations are quite time-consuming: we plan on implementing those to be able to study more full-fledged examples. Also, many natural examples (e.g. lattices) contain commutative operations, for which there is no hope of obtaining a terminating rewriting system, which suggests that we should investigate a generalization of the construction for rewriting modulo.

Finally, homological invariants are quite rough, in the sense that they do not provide interesting information about some interesting rewriting systems. In order to handle those, we believe that investigating properties of homotopical nature would provide much more precise insights.

References

1 Jiří Adámek and Jiří Rosicky. *Locally presentable and accessible categories*, volume 189. Cambridge University Press, 1994.

2 Michael Barr. Cartan-Eilenberg cohomology and triples. *Journal of Pure and Applied Algebra*, 112(3):219–238, 1996.

3 Jonathan Mock Beck. *Triples, algebras and cohomology.* PhD thesis, Columbia Univ, 1967.

4 Albert Burroni. Higher-dimensional word problems with applications to equational logic. *Theoretical computer science*, 115(1):43–62, 1993.

5 Yves Guiraud, Philippe Malbos, and Samuel Mimram. A homotopical completion procedure with applications to coherence of monoids. In *RTA*, volume 21, pages 223–238. Dagstuhl, 2013.

6 Allen Hatcher. *Algebraic Topology.* Cambridge University Press, 2002.

7 Graham Higman and B. H. Neumann. Groups as groupoids with one law. *Publicationes Mathematicae Debrecen*, 2(215-227):228, 1952.

8 Mamuka Jibladze and Teimuraz Pirashvili. Cohomology of algebraic theories. *Journal of Algebra*, 137(2):253–296, 1991.

9 Mamuka Jibladze and Teimuraz Pirashvili. Quillen cohomology and Baues-Wirsching cohomology of algebraic theories. *Cahiers de top. et géom. diff. cat.*, 47(3):163–205, 2006.

10 Donald E Knuth and Peter B Bendix. Simple word problems in universal algebras. In *Automation of Reasoning*, pages 342–376. Springer, 1983.

11 Yuji Kobayashi. Complete rewriting systems and homology of monoid algebras. *Journal of Pure and Applied Algebra*, 65(3):263–275, 1990.

12 Yves Lafont and Alain Prouté. Church-rooser property and homology of monoids. *Mathematical Structures in Computer Science*, 1(03):297–326, 1991.

13 F William Lawvere. Functorial semantics of algebraic theories. *Proceedings of the National Academy of Sciences of the United States of America*, 50(5):869, 1963.

14 Jean-Louis Loday and Bruno Vallette. *Algebraic operads*, volume 346 of *Grundlehren der Mathematischen Wissenschaften.* Springer, Heidelberg, 2012.

15 Saunders Mac Lane. *Homology.* Springer-Verlag, Berlin, 1995.

16 Philippe Malbos. *Critères de finitude homologique pour la non convergence des systèmes de réécriture de termes.* PhD thesis, Université de Montpellier II, 2004.

17 William Mccune et al. Single Axioms: With and Without Computers. In *Computer Mathematics: Proceedings of the Fourth Asian Symposium (ASCM 2000)*, page 83. World Scientific, 2000.

18 William McCune, Ranganathan Padmanabhan, and Robert Veroff. Yet another single law for lattices. *Algebra Universalis*, 50(2):165–169, 2003.

19 William McCune, Robert Veroff, Branden Fitelson, Kenneth Harris, Andrew Feist, and Larry Wos. Short single axioms for Boolean algebra. *Journal of Automated Reasoning*, 29(1):1–16, 2002.

20 Ralph McKenzie. Equational Bases for Lattice Theories. *Math. Scandinavica*, 27:24–38, 1970.

21 Paul-André Melliès. Axiomatic rewriting theory VI: Residual theory revisited. In *Rewriting techniques and applications*, pages 24–50. Springer, 2002.

22 Barry Mitchell. Rings with several objects. *Advances in Mathematics*, 8(1):1–161, 1972.

23 B. H. Neumann et al. Yet another single law for groups. *Illinois Journal of Mathematics*, 30(2):295–300, 1986.

24 Bernhard Hermann Neumann. Another single law for groups. *Bulletin of the Australian Mathematical Society*, 23(01):81–102, 1981.

25 Maxwell Herman Alexander Newman. On theories with a combinatorial definition of "equivalence". *Annals of mathematics*, pages 223–243, 1942.

26 R Padmanabhan and RW Quackenbush. Equational theories of algebras with distributive congruences. *Proceedings of the American Mathematical Society*, 41(2):373–377, 1973.

27 DH Potts. Axioms for semi-lattices. *Canad. Math Bulletin*, 8:519, 1965.

28 Craig Squier and Friedrich Otto. The word problem for finitely presented monoids and finite canonical rewriting systems. In *Rewriting Techniques and Applications*, pages 74–82. Springer, 1987.

29 Craig C Squier, Friedrich Otto, and Yuji Kobayashi. A finiteness condition for rewriting systems. *Theoretical Computer Science*, 131(2):271–294, 1994.

30 Alfred Tarski. Ein Beitrag zur Axiomatik der Abelschen Gruppen. *Fundamenta Mathematicae*, 1(30):253–256, 1938.

31 Heinrich Tietze. Über die topologischen Invarianten mehrdimensionaler Mannigfaltigkeiten. *Monatsh. Math. Phys.*, 19(1):1–118, 1908.

Reversible Term Rewriting[*]

Naoki Nishida[1], Adrián Palacios[2], and Germán Vidal[3]

1 Graduate School of Information Science, Nagoya University, Nagoya, Japan
nishida@is.nagoya-u.ac.jp

2 MiST, DSIC, Universitat Politècnica de València, Valencia, Spain
apalacios@dsic.upv.es

3 MiST, DSIC, Universitat Politècnica de València, Valencia, Spain
gvidal@dsic.upv.es

Abstract

Essentially, in a reversible programming language, for each forward computation step from state S to state S', there exists a constructive and deterministic method to go backwards from state S' to state S. Besides its theoretical interest, reversible computation is a fundamental concept which is relevant in many different areas like cellular automata, bidirectional program transformation, or quantum computing, to name a few. In this paper, we focus on term rewriting, a computation model that underlies most rule-based programming languages. In general, term rewriting is not reversible, even for injective functions; namely, given a rewrite step $t_1 \to t_2$, we do not always have a decidable and deterministic method to get t_1 from t_2. Here, we introduce a conservative extension of term rewriting that becomes reversible. Furthermore, we also define a transformation to make a rewrite system reversible using standard term rewriting.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases term rewriting, reversible computation, program transformation

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.28

1 Introduction

The notion of reversible computation can be traced back to Landauer's pioneering work [14]. Although Landauer was mainly concerned with the energy consumption of erasing data in irreversible computing (only recently experimentally measured [5]), he also claimed that every computer can be made reversible by saving the *history* of the computation. However, as Landauer himself pointed out, this would only postpone the problem of erasing the tape of a reversible Turing machine before it could be reused. Bennett [3] improved the original proposal so that the computation now ends with a tape that only contains the output of a computation and the initial source, thus deleting all remaining "garbage" data, though it performs twice the usual computation steps. More recently, Bennett's result is extended in [6] to nondeterministic Turing machines, where it is also proved that transforming an irreversible Turing machine into a reversible one can be done with a quadratic loss of space.

[*] This work has been partially supported by the EU (FEDER) and the Spanish *Ministerio de Economía y Competitividad* (MINECO) under grant TIN2013-44742-C4-1-R, by the *Generalitat Valenciana* under grant PROMETEO-II/2015/013 (SmartLogic) and by the COST Action IC1405 on Reversible Computation. A. Palacios was partially supported by the the EU (FEDER) and the Spanish *Ayudas para contratos predoctorales para la formación de doctores de la Sec. Estado de Investigación, Desarrollo e Innovación del MINECO* under FPI grant BES-2014-069749. Part of this research was done while the second and third authors were visiting Nagoya University; they gratefully acknowledge their hospitality.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 28; pp. 28:1–28:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the last decades, reversible computing and *reversibilization* – transforming an irreversible computation device into a reversible one – have been the subject of intense research, giving rise to successful applications in many different fields ranging from cellular automata [19] and bidirectional program transformation [15] to quantum computing [29], to name a few. We refer the interested reader to, e.g., [4, 9, 30] for a high level account of the principles of reversible computation.

In this work, we focus on *term rewriting* [2, 26], a computation model that underlies most rule-based programming languages. Essentially, there are two approaches to designing a reversible language: one can either restrict the language to only contain reversible constructs, or one can include some additional information (typically, the *history* of the computation so far) so that all constructs become reversible, which is called a *Landauer's embedding.* The first approach is considered, e.g., by Abramsky in the context of pattern matching automata [1]. There, *biorthogonality* is required to ensure reversibility, which would be a very significant restriction for term rewriting systems. Thus, we follow the second, more general approach by introducing the information required for the reductions to become reversible.

To be more precise, we introduce a general and intuitive notion of *reversible* term rewriting by following essentially a Landauer's embedding. Given a rewrite system $\mathcal{R}$ and its associated (standard) rewrite relation $\to_{\mathcal{R}}$, we define a reversible extension of rewriting with two components: a forward relation $\rightharpoonup_{\mathcal{R}}$ and a backward relation $\leftharpoondown_{\mathcal{R}}$, such that $\rightharpoonup_{\mathcal{R}}$ is a conservative extension of $\to_{\mathcal{R}}$ and, moreover, $(\rightharpoonup_{\mathcal{R}})^{-1} = \leftharpoondown_{\mathcal{R}}$. We note that the inverse rewrite relation, $(\to_{\mathcal{R}})^{-1}$, is not an appropriate basis for "reversible" rewriting since we aim at defining a technique to *undo* a given reduction. In other words, given a rewriting reduction $s \to^*_{\mathcal{R}} t$, a reversible relation aims at computing the term s from t and $\mathcal{R}$ in a decidable and deterministic way, which is not possible using $(\to_{\mathcal{R}})^{-1}$ since it is generally non-deterministic, non-confluent, and non-terminating, even for systems defining injective functions (see Example 8). In contrast, our backward relation $\leftharpoondown_{\mathcal{R}}$ is deterministic (thus confluent) and terminating.

We then introduce a *flattening* transformation for rewrite systems so that the reduction at top positions of terms suffices to get a normal form in the transformed systems. For instance, given the following rewrite system $\mathcal{R} = \{\mathsf{a}(\mathsf{0}, y) \to y,\ \mathsf{a}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{a}(x, y))\}$ defining the addition on natural numbers built from constructors $\mathsf{0}$ and $\mathsf{s}(\ )$, we produce the following *basic* (conditional) system: $\mathcal{R}' = \{\mathsf{a}(\mathsf{0}, y) \to y,\ \mathsf{a}(\mathsf{s}(x), y) \to \mathsf{s}(z) \Leftarrow \mathsf{a}(x, y) \twoheadrightarrow z\}$ (see Example 16 for more details). This allows us to provide an improved notion of reversible rewriting in which some information – namely, the positions where reduction takes place – is not required anymore. This opens the door to *compile* the reversible extension of rewriting into the system rules. Loosely speaking, given a system $\mathcal{R}$, we produce new systems $\mathcal{R}_f$ and $\mathcal{R}_b$ such that *standard* rewriting in $\mathcal{R}_f$, i.e., $\to_{\mathcal{R}_f}$, coincides with the forward reversible extension $\rightharpoonup_{\mathcal{R}}$ in the original system, and analogously $\to_{\mathcal{R}_b}$ is equivalent to $\leftharpoondown_{\mathcal{R}}$. E.g., for the system $\mathcal{R}'$ above, we would produce

$$\begin{aligned}\mathcal{R}_f = \{ \quad & \mathsf{a}^i(\mathsf{0}, y) \to \langle y, \beta_1 \rangle, \\ & \mathsf{a}^i(\mathsf{s}(x), y) \to \langle \mathsf{s}(z), \beta_2(w) \rangle \Leftarrow \mathsf{a}^i(x, y) \twoheadrightarrow \langle z, w \rangle \ \}\end{aligned}$$

$$\begin{aligned}\mathcal{R}_b = \{ \quad & \mathsf{a}^{-1}(y, \beta_1) \to \langle \mathsf{0}, y \rangle, \\ & \mathsf{a}^{-1}(\mathsf{s}(z), \beta_2(w)) \to \langle \mathsf{s}(x), y \rangle \Leftarrow \mathsf{a}^{-1}(z, w) \to \langle x, y \rangle \quad \}\end{aligned}$$

where a^i is an injective version of function a, a^{-1} is the inverse of a^i, and β_1, β_2 are fresh symbols introduced to label the rules of the original system.

We consider *conditional* rewrite systems in this work, not only to have a more general notion of reversible rewriting, but also to define a reversibilization technique for unconditional rewrite systems, since the application of *flattening* (cf. Section 4) may introduce conditions in a system that is originally unconditional, as illustrated above. We refer the interested reader to [21] for a definition of reversible term rewriting for unconditional systems.

The paper is organized as follows. After introducing some preliminaries in Section 2, we present our approach to reversible term rewriting in Section 3. Then, Section 4 introduces a transformation to *basic* systems, and Section 5 presents injectivization and inversion transformations in order to make a rewrite system reversible with standard rewriting. Finally, Section 6 discusses some related work and Section 7 concludes and points out some ideas for future research. More details and missing proofs can be found in [21].

2 Preliminaries

We assume familiarity with basic concepts of term rewriting. We refer the reader to, e.g., [2] and [26] for further details.

Terms and Substitutions. A *signature* $\mathcal{F}$ is a set of function symbols. Given a set of variables $\mathcal{V}$ with $\mathcal{F} \cap \mathcal{V} = \varnothing$, we denote the domain of *terms* by $\mathcal{T}(\mathcal{F}, \mathcal{V})$. We use $\mathsf{f}, \mathsf{g}, \ldots$ to denote functions and $x, y, \ldots$ to denote variables. Positions are used to address the nodes of a term viewed as a tree. A *position* p in a term t, in symbols $p \in \mathcal{P}os(t)$, is represented by a finite sequence of natural numbers, where ϵ denotes the root position. We let $t|_p$ denote the *subterm* of t at position p and $t[s]_p$ the result of *replacing the subterm* $t|_p$ by the term s. $\mathcal{V}ar(t)$ denotes the set of variables appearing in t. We also let $\mathcal{V}ar(t_1, \ldots, t_n)$ denote $\mathcal{V}ar(t_1) \cup \cdots \cup \mathcal{V}ar(t_n)$. A term t is *ground* if $\mathcal{V}ar(t) = \varnothing$.

A *substitution* $\sigma : \mathcal{V} \mapsto \mathcal{T}(\mathcal{F}, \mathcal{V})$ is a mapping from variables to terms such that $\mathcal{D}\mathsf{om}(\sigma) = \{x \in \mathcal{V} \mid x \neq \sigma(x)\}$ is its domain. A substitution σ is *ground* if $x\sigma$ is ground for all $x \in \mathcal{D}\mathsf{om}(\sigma)$. Substitutions are extended to morphisms from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ to $\mathcal{T}(\mathcal{F}, \mathcal{V})$ in the natural way. We denote the application of a substitution σ to a term t by $t\sigma$ rather than $\sigma(t)$. The identity substitution is denoted by *id*. We let "$\circ$" denote the composition of substitutions, i.e., $\sigma \circ \theta(x) = (x\theta)\sigma = x\theta\sigma$. The *restriction* $\theta\!\restriction_V$ of a substitution θ to a set of variables V is defined as follows: $x\theta\!\restriction_V = x\theta$ if $x \in V$ and $x\theta\!\restriction_V = x$ otherwise. *TRSs and Rewriting.* A set of rewrite rules $l \to r$ such that l is a nonvariable term and r is a term whose variables appear in l is called a *term rewriting system* (TRS for short); terms l and r are called the left-hand side and the right-hand side of the rule, respectively. We restrict ourselves to finite signatures and TRSs. Given a TRS $\mathcal{R}$ over a signature $\mathcal{F}$, the *defined* symbols $\mathcal{D}_\mathcal{R}$ are the root symbols of the left-hand sides of the rules and the *constructors* are $\mathcal{C}_\mathcal{R} = \mathcal{F} \setminus \mathcal{D}_\mathcal{R}$. *Constructor terms* of $\mathcal{R}$ are terms over $\mathcal{C}_\mathcal{R}$ and $\mathcal{V}$, denoted by $\mathcal{T}(\mathcal{C}_\mathcal{R}, \mathcal{V})$. We sometimes omit $\mathcal{R}$ from $\mathcal{D}_\mathcal{R}$ and $\mathcal{C}_\mathcal{R}$ if it is clear from the context. A substitution σ is a *constructor substitution* (of $\mathcal{R}$) if $x\sigma \in \mathcal{T}(\mathcal{C}_\mathcal{R}, \mathcal{V})$ for all variables x.

For a TRS $\mathcal{R}$, we define the associated rewrite relation $\to_\mathcal{R}$ as the smallest binary relation satisfying the following: given terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we have $s \to_\mathcal{R} t$ iff there exist a position p in s, a rewrite rule $l \to r \in \mathcal{R}$, and a substitution σ such that $s|_p = l\sigma$ and $t = s[r\sigma]_p$; the rewrite step is sometimes denoted by $s \to_{p, l \to r} t$ to make explicit the position and rule used in this step. The instantiated left-hand side $l\sigma$ is called a *redex.* A term t is called *irreducible* or in *normal form* w.r.t. a TRS $\mathcal{R}$ if there is no term s with $t \to_\mathcal{R} s$. A substitution is called *normalized* w.r.t. $\mathcal{R}$ if every variable in the domain is replaced by a normal form w.r.t. $\mathcal{R}$. We sometimes omit "w.r.t. $\mathcal{R}$" if it is clear from the context. A *derivation* is a (possibly empty) sequence of rewrite steps. Given a binary relation $\to$, we denote by $\to^*$ its reflexive and transitive closure, i.e., $s \to^*_\mathcal{R} t$ means that s can be reduced to t in $\mathcal{R}$ in zero or more steps; we also use $s \to^n_\mathcal{R} t$ to denote that s can be reduced to t in exactly n steps.

In this paper, we consider *conditional* term rewrite systems (CTRSs); namely oriented 3-CTRSs, i.e., CTRSs where extra variables are allowed as long as $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l) \cup \mathcal{V}ar(C)$ for any rule $l \to r \Leftarrow C$ [17]. In *oriented* CTRSs, a conditional rule $l \to r \Leftarrow C$ has the form $l \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_n \twoheadrightarrow t_n$, where each oriented equation $s_i \twoheadrightarrow t_i$ is interpreted as reachability ($\to^*_\mathcal{R}$). In the following, we denote by $\overline{o_n}$ a sequence of elements $o_1, \ldots, o_n$ for some n. We also write $\overline{o_{i,j}}$ for the sequence $o_i, \ldots, o_j$ when $i \leq j$ (and the empty sequence otherwise). We write $\overline{o}$ when the number of elements is not relevant. In addition, we denote $o_1 \twoheadrightarrow o'_1, \ldots, o_n \twoheadrightarrow o'_n$ by $\overline{o_n \twoheadrightarrow o'_n}$. Moreover, we assume that rewrite rules are labelled, i.e., given a CTRS $\mathcal{R}$, we denote by $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}$ a rewrite rule with label β. Labels are unique in a CTRS. Also, to relate label β to fixed variables, we consider that the variables of the rewrite rules are not renamed and that the reduced terms are always ground.[1] We often write $s \to_{p,\beta} t$ instead of $s \to_{p, l \to r \Leftarrow \overline{s_n \to t_n}} t$ if rule $l \to r \Leftarrow \overline{s_n \to t_n}$ is labeled with β.

For a CTRS $\mathcal{R}$, the associated rewrite relation $\to_\mathcal{R}$ is defined as the smallest binary relation satisfying the following: given ground terms $s, t \in \mathcal{T}(\mathcal{F})$, we have $s \to_\mathcal{R} t$ iff there exist a position p in s, a rewrite rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, and a ground substitution σ such that $s|_p = l\sigma$, $s_i\sigma \to^*_\mathcal{R} t_i\sigma$ for all $i = 1, \ldots, n$, and $t = s[r\sigma]_p$.

In order to simplify the presentation, we only consider *deterministic* CTRSs (DCTRSs), i.e., oriented 3-CTRSs where, for each rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}$, we have $\mathcal{V}\mathsf{ar}(s_i) \subseteq \mathcal{V}\mathsf{ar}(l, \overline{t_{i-1}})$ for all $i = 1, \ldots, n$. Intuitively speaking, the use of DCTRs allows us to compute the bindings for the variables in the condition of a rule in a deterministic way. E.g., given a ground term t and a rule $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}$ with $t|_p = l\theta$, we have that $s_1\theta$ is ground. Therefore, one can reduce $s_1\theta$ to some term s'_1 such that s'_1 is an instance of $t_1\theta$ with some ground substitution θ_1. Now, we have that $s_2\theta\theta_1$ is ground and we can reduce $s_2\theta\theta_1$ to some term s'_2 such that s'_2 is an instance of $t_2\theta\theta_1$ with some ground substitution θ_2, and so forth. If all equations in the condition hold using $\theta_1, \ldots, \theta_n$, we have that $t \to_{p,\beta} t[r\sigma]_p$ with $\sigma = \theta\theta_1 \ldots \theta_n$.

▶ **Example 1.** Consider the following DCTRS $\mathcal{R}$ that defines the function double that doubles the value of its argument when it is an even natural number:

$$\begin{array}{rrcll@{\qquad}rrcl}
\beta_1 : & \mathsf{add}(0, y) & \to & y & & \beta_4 : & \mathsf{even}(0) & \to & \mathsf{true} \\
\beta_2 : & \mathsf{add}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{add}(x, y)) & & \beta_5 : & \mathsf{even}(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{even}(x) \\
\beta_3 : & \mathsf{double}(x) & \to & \mathsf{add}(x, x) & \Leftarrow \ \mathsf{even}(x) \twoheadrightarrow \mathsf{true} & & & &
\end{array}$$

Given the term $\mathsf{double}(\mathsf{s}(\mathsf{s}(0)))$ we have, for instance, the following derivation:

$$\begin{array}{rll}
\mathsf{double}(\mathsf{s}(\mathsf{s}(0))) \to_{\epsilon,\beta_3} & \mathsf{add}(\mathsf{s}(\mathsf{s}(0)), \mathsf{s}(\mathsf{s}(0))) & \text{since } \mathsf{even}(\mathsf{s}(\mathsf{s}(0))) \to^*_\mathcal{R} \mathsf{true} \\
 & & \text{with } \sigma = \{x \mapsto \mathsf{s}(\mathsf{s}(0))\} \\
\to_{\epsilon,\beta_2} & \mathsf{s}(\mathsf{add}(\mathsf{s}(0), \mathsf{s}(\mathsf{s}(0)))) & \text{with } \sigma = \{x \mapsto \mathsf{s}(0),\ y \mapsto \mathsf{s}(\mathsf{s}(0))\} \\
\to_{1,\beta_2} & \mathsf{s}(\mathsf{s}(\mathsf{add}(0, \mathsf{s}(\mathsf{s}(0))))) & \text{with } \sigma = \{x \mapsto 0, y \mapsto \mathsf{s}(\mathsf{s}(0))\} \\
\to_{1.1,\beta_1} & \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(0)))) & \text{with } \sigma = \{y \mapsto \mathsf{s}(\mathsf{s}(0))\}
\end{array}$$

3 Reversible Term Rewriting

In this section, we present a conservative extension of the rewrite relation which becomes reversible. In the following, we use $\rightharpoonup_\mathcal{R}$ to denote our *reversible* (forward) term rewrite relation, and $\leftharpoondown_\mathcal{R}$ to denote its application in the reverse (backward) direction. Note that, in

[1] Equivalently, one could require terms to be variable disjoint with the variables of the rewrite system, but we require groundness for simplicity.

principle, we do not require $\leftharpoondown_{\mathcal{R}} = \rightharpoonup_{\mathcal{R}}^{-1}$, i.e., we provide independent (constructive) definitions for each relation. Nonetheless, we will prove that $\leftharpoondown_{\mathcal{R}} = \rightharpoonup_{\mathcal{R}}^{-1}$ indeed holds (cf. Theorem 10). In some approaches to reversible computing, both forward and backward relations should be deterministic. Here, we will only require deterministic *backward* steps, while forward steps might be non-deterministic, as it is often the case in term rewriting. We note that considering DCTRSs is not enough to make conditional rewriting deterministic. In general, given a rewrite step $s \to_{p,\beta} t$ with p a position of s, $\beta : l \to r \Leftarrow \overline{s_n \to t_n}$ a rule, and σ a substitution such that $s|_p = l\sigma$ and $s_i\sigma \to_{\mathcal{R}}^{*} t_i\sigma$ for all $i = 1, \ldots, n$, there are three sources of non-determinism: the selected position p, the selected rule β, and the substitution σ. The use of DCTRSs can only make deterministic the last one, but the choice of a position and the selection of a rule may still be non-deterministic.

Reversible rewriting is then defined on pairs $\langle t, \pi \rangle$, where t is a term and π is a trace:

▶ **Definition 2** (trace)**.** Given a CTRS $\mathcal{R}$, a *trace* in $\mathcal{R}$ is recursively defined as follows:[2]

- the empty list is a trace;
- if $\pi, \pi_1, \ldots, \pi_n$ are traces in $\mathcal{R}$, $n \geq 0$, there is a rule $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, p is a position, and σ is a ground substitution, then $\beta(p, \sigma, \pi_1, \ldots, \pi_n) : \pi$ is a trace in $\mathcal{R}$.

We refer to each component $\beta(p, \sigma, \pi_1, \ldots, \pi_n)$ in a trace as a *trace term*.

Intuitively speaking, a trace term $\beta(p, \sigma, \pi_1, \ldots, \pi_n)$ stores the position of a reduction step, a substitution with the bindings that are required for the step to be reversible (e.g., the bindings for the erased variables, but not only; see below) and the traces associated to the subcomputations in the condition. Our trace terms have some similarities with *proof terms* [26]. However, proof terms do not store the bindings of erased variables (and, to the best of our knowledge, are only defined for unconditional TRSs).

Our reversible term rewriting relation is only defined on *safe* pairs. This notion will be clarified below, after introducing the definition of reversible rewriting.

▶ **Definition 3** (safe pair)**.** Let $\mathcal{R}$ be a DCTRS. The pair $\langle s, \pi \rangle$ is *safe* in $\mathcal{R}$ iff, for all trace terms $\beta(p, \sigma, \overline{\pi_n})$ in π, σ is a ground substitution with $\mathcal{D}\mathsf{om}(\sigma) = (\mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(t_i) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})$ and $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$.

In the following, we often omit $\mathcal{R}$ when referring to traces and safe pairs if the underlying CTRS is clear from the context.

▶ **Definition 4** (reversible rewriting)**.** Let $\mathcal{R}$ be a DCTRS. The reversible rewrite relation $\rightharpoonup_{\mathcal{R}}$ is defined on pairs $\langle t, \pi \rangle$, where t is a ground term and π is a trace in $\mathcal{R}$. The reversible rewrite relation extends standard rewriting as follows:

$$\langle s, \pi \rangle \rightharpoonup_{\mathcal{R}} \langle t, \beta(p, \sigma', \pi_1, \ldots, \pi_n) : \pi \rangle$$

iff there exist a position $p \in \mathcal{P}os(s)$, a rewrite rule $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, and a ground substitution σ such that $s|_p = l\sigma$, $\langle s_i\sigma, [\,] \rangle \rightharpoonup_{\mathcal{R}}^{*} \langle t_i\sigma, \pi_i \rangle$ for all $i = 1, \ldots, n$, $t = s[r\sigma]_p$, and $\sigma' = \sigma\restriction_{(\mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(t_i) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})}$. The reverse relation, $\leftharpoondown_{\mathcal{R}}$, is then defined as follows:

$$\langle t, \beta(p, \sigma', \pi_1, \ldots, \pi_n) : \pi \rangle \leftharpoondown_{\mathcal{R}} \langle s, \pi \rangle$$

iff $\langle t, \beta(p, \sigma', \overline{\pi_n}) : \pi \rangle$ is a safe pair in $\mathcal{R}$, $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$ and there is a ground substitution θ such that $\mathcal{D}\mathsf{om}(\theta) = \mathcal{V}\mathsf{ar}(r, \overline{s_n}) \backslash \mathcal{D}\mathsf{om}(\sigma')$, $t|_p = r\theta$, $\langle t_i\, \theta \cup \sigma', \pi_i \rangle \leftharpoondown_{\mathcal{R}}^{*}$

[2] As it is common, $[\,]$ denotes the empty list and $x : xs$ is a list with head x and tail xs.

$\langle s_i\,\theta{\cup}\sigma', [\,]\rangle$ for all $i = 1, \ldots, n$, and $s = t[l\,\theta{\cup}\sigma']_p$. Here, we assume that $\cup$ is the union of substitutions and that it binds stronger than substitution application, i.e., $l\,\theta{\cup}\sigma' = l(\theta \cup \sigma')$. Note that $\theta \cup \sigma'$ is well defined since $\mathcal{D}\mathsf{om}(\theta) \cap \mathcal{D}\mathsf{om}(\sigma') = \varnothing$ (actually, $\theta \cup \sigma' = \theta\sigma' = \sigma'\theta$ since they are also ground).

We denote the union of both relations $\rightharpoonup_{\mathcal{R}} \cup \leftharpoondown_{\mathcal{R}}$ by $\rightleftharpoons_{\mathcal{R}}$.

▶ **Example 5.** Consider the DCTRS $\mathcal{R}$ from Example 1. Given the term $\mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{0})))$, we have, for instance, the following forward derivation:

$$\begin{array}{lll} \langle \mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{0}))), [\,]\rangle & \rightharpoonup_{\mathcal{R}} & \langle \mathsf{add}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{s}(\mathsf{s}(\mathsf{0}))), [\beta_3(\epsilon, id, \pi)]\rangle \\ & \rightharpoonup_{\mathcal{R}} & \cdots \\ & \rightharpoonup_{\mathcal{R}} & \langle \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{0})))), [\beta_1(1.1, id), \beta_2(1, id), \beta_2(\epsilon, id), \beta_3(\epsilon, id, \pi)]\rangle \end{array}$$

where $\pi = [\beta_4(\epsilon, id), \beta_5(\epsilon, id)]$ since we have the following reduction:

$$\langle \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{0}))), [\,]\rangle \rightharpoonup_{\mathcal{R}} \langle \mathsf{even}(\mathsf{0}), [\beta_5(\epsilon, id)]\rangle \rightharpoonup_{\mathcal{R}} \langle \mathsf{true}, [\beta_4(\epsilon, id), \beta_5(\epsilon, id)]\rangle$$

The reader can easily construct the associated backward derivation:

$$\langle \mathsf{add}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{s}(\mathsf{s}(\mathsf{0}))), [\beta_1(1.1, id), \beta_2(1, id), \beta_2(\epsilon, id), \beta_3(\epsilon, id, \pi)]\rangle \leftharpoondown^*_{\mathcal{R}} \langle \mathsf{double}(\mathsf{s}(\mathsf{s}(\mathsf{0}))), [\,]\rangle$$

Let us now explain why we need to store σ' in a step of the form $\langle s, \pi\rangle \rightharpoonup_{\mathcal{R}} \langle t, \beta(p, \sigma', \overline{\pi_n}) : \pi\rangle$. Given a DCTRS, for each rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}$, the following conditions hold:

- 3-CTRS: $\mathcal{V}\mathsf{ar}(r) \subseteq \mathcal{V}\mathsf{ar}(l, \overline{s_n}, \overline{t_n})$.
- Determinism: for all $i = 1, \ldots, n$, we have $\mathcal{V}\mathsf{ar}(s_i) \subseteq \mathcal{V}\mathsf{ar}(l, \overline{t_{i-1}})$.

Intuitively, the backward relation $\leftharpoondown_{\mathcal{R}}$ can be seen as equivalent to the forward relation $\rightharpoonup_{\mathcal{R}}$ but using a reverse rule of the form $r \to l \Leftarrow t_n \twoheadrightarrow s_n, \ldots, t_1 \twoheadrightarrow s_1$. Therefore, in order to ensure that backward reduction is deterministic, we need the same conditions as above but on the reverse rewrite rule:

- 3-CTRS: $\mathcal{V}\mathsf{ar}(l) \subseteq \mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n})$.
- Determinism: for all $i = 1, \ldots, n$, $\mathcal{V}\mathsf{ar}(t_i) \subseteq \mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})$.

Since these conditions cannot be guaranteed in general, we store

$$\sigma' = \sigma\restriction_{(\mathcal{V}\mathsf{ar}(l)\backslash\mathcal{V}\mathsf{ar}(r,\overline{s_n},\overline{t_n}))\cup\bigcup_{i=1}^{n}\mathcal{V}\mathsf{ar}(t_i)\backslash\mathcal{V}\mathsf{ar}(r,\overline{s_{i+1,n}})}$$

in the trace term so that $(r \to l \Leftarrow t_n \twoheadrightarrow s_n, \ldots, t_1 \twoheadrightarrow s_1)\sigma'$ is deterministic and fulfills the conditions of a 3-CTRS by construction, i.e., $\mathcal{V}\mathsf{ar}(l\sigma') \subseteq \mathcal{V}\mathsf{ar}(r\sigma', \overline{s_n\sigma'}, \overline{t_n\sigma'})$ and for all $i = 1, \ldots, n$, $\mathcal{V}\mathsf{ar}(t_i\sigma') \subseteq \mathcal{V}\mathsf{ar}(r\sigma', \overline{s_{i+1,n}\sigma'})$; see the proof of Theorem 11 for more details.

▶ **Example 6.** Consider, e.g., the following DCTRS:

$$\begin{array}{l} \beta_1 :\ \mathsf{f}(x, y, m) \ \to\ \mathsf{s}(w) \Leftarrow \mathsf{h}(x) \twoheadrightarrow x, \mathsf{g}(y, \mathsf{4}) \twoheadrightarrow w \\ \beta_2 :\ \mathsf{h}(\mathsf{0}) \ \to\ \mathsf{0} \qquad\quad \beta_3 :\ \mathsf{h}(\mathsf{1}) \ \to\ \mathsf{1} \qquad\quad \beta_4 :\ \mathsf{g}(x, y) \ \to\ x \end{array}$$

and the step $\langle \mathsf{f}(\mathsf{0}, \mathsf{2}, \mathsf{4}), [\,]\rangle \rightharpoonup_{\mathcal{R}} \langle \mathsf{s}(\mathsf{2}), [\beta_1(\epsilon, \sigma', \pi_1, \pi_2)]\rangle$ with $\sigma' = \{m \mapsto \mathsf{4}, x \mapsto \mathsf{0}\}$, $\pi_1 = [\beta_2(\epsilon, id)]$ and $\pi_2 = [\beta_4(\epsilon, \{y \mapsto \mathsf{4}\})]$. The binding of variable m is required to recover the value of the *erased* variable m, but the binding of variable x is also needed to perform the subderivation $\langle x, \pi_1\rangle \leftharpoondown_{\mathcal{R}} \langle \mathsf{h}(x), [\,]\rangle$ when applying a backward step from $\langle \mathsf{s}(\mathsf{2}), [\beta_1(\epsilon, \sigma', \pi_1, \pi_2)]\rangle$. If the binding for x were unknown, this step would not be deterministic. As mentioned above, an instantiated reverse rule $(\mathsf{s}(w) \to \mathsf{f}(x, y, m) \Leftarrow w \twoheadrightarrow \mathsf{g}(y, \mathsf{4}), x \twoheadrightarrow \mathsf{h}(x))\sigma' = \mathsf{s}(w) \to \mathsf{f}(\mathsf{0}, y, \mathsf{4}) \Leftarrow w \twoheadrightarrow \mathsf{g}(y, \mathsf{4}), \mathsf{0} \twoheadrightarrow \mathsf{h}(\mathsf{0})$ would be a DCTRS thanks to σ'.

We note that similar conditions could be defined for arbitrary 3-CTRSs. However, the conditions would be much more involved (e.g., one should first compute the dependencies between the equations in the conditions), so we prefer to keep the simpler conditions for DCTRSs, which is still quite a general class of CTRSs.

An easy but essential property of $\rightharpoonup_{\mathcal{R}}$ is that it is a conservative extension of standard rewriting in the following sense (we omit its proof since it is straightforward):

▶ **Theorem 7.** *Let $\mathcal{R}$ be a DCTRS. Given ground terms s, t, if $s \to^*_{\mathcal{R}} t$, then for any trace π there exists a trace π' such that $\langle s, \pi\rangle \rightharpoonup^*_{\mathcal{R}} \langle t, \pi'\rangle$.*

We note that this is not the case for the backward relation: $t \leftarrow_{\mathcal{R}} s$ does not imply $\langle t, \pi'\rangle \leftharpoondown_{\mathcal{R}} \langle s, \pi\rangle$ for an arbitrary trace π'.[3] This is actually the purpose of our notion of reversible rewriting: $\leftharpoondown_{\mathcal{R}}$ should not extend $\leftarrow_{\mathcal{R}}$ but is only aimed at performing *exactly the same steps* of the forward computation whose trace was stored but in the reverse order. Nonetheless, one can still ensure that, for any step $t \leftarrow_{\mathcal{R}} s$, there is a trace π' such that $\langle t, \pi'\rangle \leftharpoondown_{\mathcal{R}} \langle s, \pi\rangle$ for some trace π (which is an easy consequence of Theorems 7 and 10).

▶ **Example 8.** Consider the following simple TRS $\mathcal{R} = \{\beta : \ \mathsf{snd}(x, y) \to y\}$. Given the reduction $\mathsf{snd}(1, 2) \to_{\mathcal{R}} 2$, there are infinitely many reductions for 2 using $\leftarrow_{\mathcal{R}}$, e.g., $2 \leftarrow_{\mathcal{R}} \mathsf{snd}(1, 2)$, $2 \leftarrow_{\mathcal{R}} \mathsf{snd}(2, 2)$, $2 \leftarrow_{\mathcal{R}} \mathsf{snd}(3, 2)$, etc. The relation is also non-terminating: $2 \leftarrow_{\mathcal{R}} \mathsf{snd}(1, 2) \leftarrow_{\mathcal{R}} \mathsf{snd}(1, \mathsf{snd}(1, 2)) \leftarrow_{\mathcal{R}} \cdots$. In contrast, given a pair $\langle 2, \pi\rangle$, we can only perform a single deterministic and finite reduction (as proved below).

The following result states that every pair which is reachable from an initial pair with an empty trace is safe, and follows easily by induction on the length of the derivations:

▶ **Proposition 9.** *Let $\mathcal{R}$ be a DCTRS. If $\langle s, [\,]\rangle \rightleftharpoons^*_{\mathcal{R}} \langle t, \pi\rangle$, then $\langle t, \pi\rangle$ is safe in $\mathcal{R}$.*

For the following result, we need some preliminary notions (see, e.g., [26]). For every oriented CTRS $\mathcal{R}$, we inductively define the TRSs $\mathcal{R}_k$, $k \geq 0$, as follows:

$$\begin{array}{lcl} \mathcal{R}_0 & = & \varnothing \\ \mathcal{R}_{k+1} & = & \{l\sigma \to r\sigma \mid l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R},\ s_i\sigma \to^*_{\mathcal{R}_k} t_i\sigma \text{ for all } i = 1, \ldots, n\} \end{array}$$

Observe that $\mathcal{R}_k \subseteq \mathcal{R}_{k+1}$ for all $k \geq 0$. We have $\to_{\mathcal{R}} = \bigcup_{i \geq 0} \to_{\mathcal{R}_i}$. We also have $s \to_{\mathcal{R}} t$ iff $s \to_{\mathcal{R}_k} t$ for some $k \geq 0$. The minimum such k is called the *depth* of $s \to_{\mathcal{R}} t$, and the maximum depth k of $s = s_0 \to_{\mathcal{R}_{k_1}} \cdots \to_{\mathcal{R}_{k_m}} s_m = t$ (i.e., k is the maximum of depths $k_1, \ldots, k_m$) is called the *depth* of the derivation. If a derivation has depth k and length m, we write $s \to^m_{\mathcal{R}_k} t$. Analogous notions can naturally be defined for $\rightharpoonup_{\mathcal{R}}$, $\leftharpoondown_{\mathcal{R}}$, and $\rightleftharpoons_{\mathcal{R}}$.

Now, we can already state the reversibility of $\rightharpoonup_{\mathcal{R}}$:

▶ **Theorem 10.** *Let $\mathcal{R}$ be DCTRS. Given the safe pairs $\langle s, \pi\rangle$ and $\langle t, \pi'\rangle$, for all $k, m \geq 0$, $\langle s, \pi\rangle \rightharpoonup^m_{\mathcal{R}_k} \langle t, \pi'\rangle$ iff $\langle t, \pi'\rangle \leftharpoondown^m_{\mathcal{R}_k} \langle s, \pi\rangle$.*

Proof. ($\Rightarrow$) We prove the claim by induction on the lexicographic product (k, m) of the depth k and the length m of the derivation $\langle s, \pi\rangle \rightharpoonup^m_{\mathcal{R}_k} \langle t, \pi'\rangle$. Since the base case is trivial, we consider the inductive case $(k, m) > (0, 0)$. Consider a derivation $\langle s, \pi\rangle \rightharpoonup^{m-1}_{\mathcal{R}_k} \langle s_0, \pi_0\rangle \rightharpoonup_{\mathcal{R}_k} \langle t, \pi'\rangle$ whose associated product is (k, m). By Proposition 9, both $\langle s_0, \pi_0\rangle$ and $\langle t, \pi'\rangle$ are safe. By the induction hypothesis, since $(k, m-1) < (k, m)$, we have $\langle s_0, \pi_0\rangle \leftharpoondown^{m-1}_{\mathcal{R}_k} \langle s, \pi\rangle$. Consider now the step $\langle s_0, \pi_0\rangle \rightharpoonup_{\mathcal{R}_k} \langle t, \pi'\rangle$. Thus, there exist a

[3] Here, and in the following, we assume that $\leftarrow_{\mathcal{R}} = (\to_{\mathcal{R}})^{-1}$.

position $p \in \mathcal{P}os(s_0)$, a rule $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, and a ground substitution σ such that $s_0|_p = l\sigma$, $\langle s_i\sigma, [\,]\rangle \rightharpoonup^*_{\mathcal{R}_{k'_i}} \langle t_i\sigma, \pi_i\rangle$ for all $i = 1,\ldots,n$, $t = s_0[r\sigma]_p$, $\sigma' = \sigma\restriction_{(\mathcal{V}\mathsf{ar}(l)\backslash\mathcal{V}\mathsf{ar}(r,\overline{s_n},\overline{t_n}))\cup\bigcup_{i=1}^{n}\mathcal{V}\mathsf{ar}(t_i)\backslash\mathcal{V}\mathsf{ar}(r,\overline{s_{i+1,n}})}$, and $\pi' = \beta(p, \sigma', \pi_1, \ldots, \pi_n) : \pi_0$. By definition of $\rightharpoonup_{\mathcal{R}_k}$, we have that $k'_i < k$ and, thus, $(k'_i, m_1) < (k, m_2)$ for all $i = 1,\ldots,n$ and for all m_1, m_2. Hence, by the induction hypothesis, we have $\langle t_i\sigma, \pi_i\rangle \leftharpoondown^*_{\mathcal{R}_{k'_i}} \langle s_i\sigma, [\,]\rangle$ for all $i = 1,\ldots,n$. Let $\theta = \sigma\restriction_{\mathcal{V}\mathsf{ar}(r,\overline{s_n})\backslash\mathcal{D}\mathsf{om}(\sigma')}$, so that $\sigma = \theta\cup\sigma'$ is well defined. Therefore, we have $\langle t, \pi'\rangle \leftharpoondown_{\mathcal{R}_k} \langle s'_0, \pi_0\rangle$ with $t|_p = r\theta$, $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$ and $s'_0 = t[l\,\theta\cup\sigma']_p = t[l\sigma]_p = s_0$, and the claim follows.

($\Leftarrow$) This direction proceeds in a similar way. We prove the claim by induction on the lexicographic product (k, m) of the depth k and the length m of the considered derivation. Since the base case is trivial, let us consider the inductive case $(k, m) > (0, 0)$. Let us consider a derivation $\langle t, \pi'\rangle \leftharpoondown^{m-1}_{\mathcal{R}_k} \langle s_0, \pi_0\rangle \leftharpoondown_{\mathcal{R}_k} \langle s, \pi\rangle$ whose associated product is (k, m). By Proposition 9, both $\langle s_0, \pi_0\rangle$ and $\langle s, \pi\rangle$ are safe. By the induction hypothesis, since $(k, m-1) < (k, m)$, we have $\langle s_0, \pi_0\rangle \rightharpoonup^{m-1}_{\mathcal{R}_k} \langle t, \pi'\rangle$. Consider now the step $\langle s_0, \pi_0\rangle \leftharpoondown_{\mathcal{R}_k} \langle s, \pi\rangle$. Then, we have $\pi_0 = \beta(p, \sigma', \pi_1, \ldots, \pi_n) : \pi$, $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, and there exists a ground substitution θ with $\mathcal{D}\mathsf{om}(\theta) = \mathcal{V}\mathsf{ar}(r, \overline{s_n})\backslash\mathcal{D}\mathsf{om}(\sigma')$ such that $s_0|_p = r\theta$, $\langle t_i\,\theta\cup\sigma', \pi_i\rangle \leftharpoondown^*_{\mathcal{R}_{k'_i}} \langle s_i\,\theta\cup\sigma', [\,]\rangle$ for all $i = 1,\ldots,n$, and $s = s_0[l\,\theta\cup\sigma']_p$. Moreover, since $\langle s_0, \pi_0\rangle$ is safe, we have that $\mathcal{D}\mathsf{om}(\sigma') = (\mathcal{V}\mathsf{ar}(l)\backslash\mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(t_i)\backslash\mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})$ and, thus, $\theta \cup \sigma'$ is well defined. By definition of $\leftharpoondown_{\mathcal{R}_k}$, we have that $k'_i < k$ and, thus, $(k'_i, m_1) < (k, m_2)$ for all $i = 1,\ldots,n$ and for all m_1, m_2. Hence, by the induction hypothesis, we have $\langle s_i\,\theta\cup\sigma', [\,]\rangle \rightharpoonup^*_{\mathcal{R}_{k'_i}} \langle t_i\,\theta\cup\sigma', \pi_i\rangle$ for all $i = 1,\ldots,n$. Let $\sigma = \theta \cup \sigma'$, which is well defined since $\mathcal{D}\mathsf{om}(\theta) \cap \mathcal{D}\mathsf{om}(\sigma') = \varnothing$. Then, since $s|_p = l\sigma$, we can perform the step $\langle s, \pi\rangle \rightharpoonup_{\mathcal{R}_k} \langle s'_0, \beta(p, \sigma', \pi_1, \ldots, \pi_n) : \pi\rangle$ with $s'_0 = s[r\sigma]_p = s[r\,\theta\cup\sigma']_p$; moreover, $s[r\,\theta\cup\sigma']_p = s[r\theta]_p = s_0[r\theta]_p = s_0$ since $\mathcal{D}\mathsf{om}(\sigma') \cap \mathcal{V}\mathsf{ar}(r) = \varnothing$, which concludes the proof. ◀

The relevance of our backward relation $\leftharpoondown_{\mathcal{R}}$ stems from the fact that it is deterministic (thus confluent), terminating, and has a constructive definition. In the following, we say that $\langle t, \pi'\rangle \leftharpoondown_{\mathcal{R}} \langle s, \pi\rangle$ is a *deterministic* step if there is no other, different pair $\langle s'', \pi''\rangle$ with $\langle t, \pi'\rangle \leftharpoondown_{\mathcal{R}} \langle s'', \pi''\rangle$ and, moreover, the subderivations for the equations in the condition of the applied rule (if any) are deterministic, too. We say that a derivation $\langle t, \pi'\rangle \leftharpoondown^*_{\mathcal{R}} \langle s, \pi\rangle$ is deterministic if each reduction step in the derivation is deterministic.

▶ **Theorem 11.** *Let $\mathcal{R}$ be a DCTRS. Let $\langle t, \pi'\rangle$ be a safe pair with $\langle t, \pi'\rangle \leftharpoondown^*_{\mathcal{R}} \langle s, \pi\rangle$ for some term s and trace π. Then $\langle t, \pi'\rangle \leftharpoondown^*_{\mathcal{R}} \langle s, \pi\rangle$ is deterministic.*

Proof. We prove the claim by induction on the lexicographic product (k, m) of the depth k and the length m of the steps. The case that $m = 0$ is trivial, and thus we let $m > 0$. Assume $\langle t, \pi'\rangle \leftharpoondown^{m-1}_{\mathcal{R}} \langle u, \pi''\rangle$. If there is no step using $\leftharpoondown_{\mathcal{R}}$ from $\langle u, \pi''\rangle$, the claim follows trivially for all $m' \leq m$. Now, assume there is at least one step issuing from $\langle u, \pi''\rangle$, e.g., $\langle u, \pi''\rangle \leftharpoondown_{\mathcal{R}} \langle s, \pi\rangle$. For the base case $k = 1$, the applied rule is unconditional and we prove that this is the only possible step. By definition, we have $\pi'' = \beta(p, \sigma') : \pi$, $p \in \mathcal{P}os(u)$, $\beta : l \to r \in \mathcal{R}_1$, and there exists a ground substitution θ with $\mathcal{D}\mathsf{om}(\theta) = \mathcal{V}\mathsf{ar}(r)$ such that $u|_p = r\theta$ and $s = u[l\,\theta\cup\sigma']_p$. The only source of nondeterminism may come from choosing a rule labeled with β and from the computation of the substitution θ. The claim trivially follows since labels are unique in $\mathcal{R}$ and, if there is another ground substitution θ' with $\theta' = \mathcal{V}\mathsf{ar}(r)$ and $u|_p = r\theta'$, then $\theta = \theta'$.

Let us now consider $k > 1$. By definition, if $\langle u, \pi''\rangle \leftharpoondown_{\mathcal{R}_k} \langle s, \pi\rangle$, we have $\pi'' = \beta(p, \sigma', \pi_1, \ldots, \pi_n) : \pi$, $\beta : l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$ and there exists a ground substitution θ with $\mathcal{D}\mathsf{om}(\theta) = \mathcal{V}\mathsf{ar}(r)$ such that $u|_p = r\theta$, $\langle t_i\,\theta\cup\sigma', \pi_i\rangle \leftharpoondown^*_{\mathcal{R}_j} \langle s_i\,\theta\cup\sigma', [\,]\rangle$, $j < k$, for all $i = 1, \ldots, n$, and $s = t[l\,\theta \cup \sigma']_p$. By the induction hypothesis, the subderivations $\langle t_i\,\theta \cup \sigma', \pi_i\rangle \leftharpoondown^*_{\mathcal{R}_j} \langle s_i\,\theta \cup \sigma', [\,]\rangle$ are deterministic, i.e., $\langle s_i\,\theta \cup \sigma', [\,]\rangle$ is a unique resulting term obtained by reducing $\langle t_i\,\theta \cup \sigma', \pi_i\rangle$. Therefore, the only remaining source of nondeterminism can come from choosing a rule labeled with β and from the computed substitution θ. On the one hand, the labels are unique in $\mathcal{R}$. As for θ, we prove that this is indeed the only possible substitution for the reduction step. Consider the instance of rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}$ with σ': $l\sigma' \to r\sigma' \Leftarrow \overline{s_n\sigma' \twoheadrightarrow t_n\sigma'}$. Since $\langle u, \pi''\rangle$ is safe, we have that σ' is a ground substitution and $\mathcal{D}om(\sigma') = (\mathcal{V}ar(l) \backslash \mathcal{V}ar(r, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^n \mathcal{V}ar(t_i) \backslash \mathcal{V}ar(r, \overline{s_{i+1,n}})$. Then, the following properties hold:

- $\mathcal{V}ar(l\sigma') \subseteq \mathcal{V}ar(r\sigma', \overline{s_n\sigma'}, \overline{t_n\sigma'})$, since σ' is ground and it covers all the variables in $\mathcal{V}ar(l) \backslash \mathcal{V}ar(r, \overline{s_n}, \overline{t_n})$.
- $\mathcal{V}ar(t_i\sigma') \subseteq \mathcal{V}ar(r\sigma', \overline{s_{i+1,n}\sigma'})$ for all $i = 1, \ldots, n$, since σ' is ground and it covers all variables in $\bigcup_{i=1}^n \mathcal{V}ar(t_i) \backslash \mathcal{V}ar(r, \overline{s_{i+1,n}})$.

The above properties guarantee that the rule $r\sigma' \to l\sigma' \Leftarrow t_n\sigma' \twoheadrightarrow s_n\sigma', \ldots, t_1\sigma' \twoheadrightarrow s_1\sigma'$ can be seen as a rule of a DCTRS and, thus, there exists a deterministic procedure to compute θ, which completes the proof. ◀

Therefore, $\leftharpoondown_{\mathcal{R}}$ is deterministic and confluent. Termination is trivially guaranteed for pairs with a finite trace since the trace's length strictly decreases with every backward step.

4 Removing Positions from Traces

Once we have a feasible definition of reversible rewriting, there are two refinements that can be considered: i) reducing the size of the traces and ii) defining a *reversibilization* transformation so that standard rewriting becomes reversible in the transformed system. Regarding the first refinement, one could remove information from the traces by requiring certain conditions on the considered systems. For instance, requiring injective functions may help to remove rule labels from trace terms. Also, requiring *non-erasing* rules may help to remove the second component of trace terms (i.e., the substitutions). In this work, however, we deal with a more challenging topic: removing positions from traces. This is useful not only to reduce the size of the traces but it is also essential to define a reversibilization technique for DCTRSs (cf. Section 5).[4]

In the following, rather than restricting the class of considered systems, we aim at transforming a given DCTRS into one that fulfills some conditions that make storing positions unnecessary. In the following, given a CTRS $\mathcal{R}$, we say that a term t is *basic* [11] if it has the form $f(\overline{t_n})$ with $f \in \mathcal{D}_{\mathcal{R}}$ a defined function symbol and $\overline{t_n} \in \mathcal{T}(\mathcal{C}_{\mathcal{R}}, \mathcal{V})$ constructor terms. Now, we introduce the following subclass of DCTRSs:

▶ **Definition 12** (basic DCTRS). A DCTRS $\mathcal{R}$ is called *basic* if, for any rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, we have that r, $\overline{s_n}$ and $\overline{t_n}$ are either basic or constructor terms.

In principle, any DCTRS can be transformed into a basic DCTRS by applying a sequence of *flattening* transformations. Roughly speaking, flattening involves transforming a term with nested defined functions like $\mathsf{f}(\mathsf{g}(x))$ into a term $\mathsf{f}(y)$ and an (oriented) equation $\mathsf{g}(x) \twoheadrightarrow y$, where y is a fresh variable.

[4] We note that defining a transformation with traces that include positions would be a rather difficult task because positions are *dynamic* (i.e., they depend on the term being reduced) and thus would require a complex (and inefficient) program instrumentation.

▶ **Definition 13** (flattening). Let $\mathcal{R}$ be a CTRS, $R = (l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}) \in \mathcal{R}$ be a rule and R' be a new rule either of the form $l \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_i|_p \twoheadrightarrow w, s_i[w]_p \twoheadrightarrow t_i, \ldots, s_n \twoheadrightarrow t_n$, for some $p \in \mathcal{P}os(s_i)$, $1 \leqslant i \leqslant n$, or $l \to r[w]_q \Leftarrow \overline{s_n \twoheadrightarrow t_n}, r|_q \twoheadrightarrow w$, for some $q \in \mathcal{P}os(r)$, where w is a fresh variable.[5] Then, a CTRS $\mathcal{R}'$ is obtained from $\mathcal{R}$ by a *flattening* step if $\mathcal{R}' = (\mathcal{R}\backslash\{R\}) \cup \{R'\}$.

Flattening is trivially *complete* since any flattening step can be undone by binding the fresh variable again to the selected subterm and, then, proceeding as in the original system. Soundness is more subtle though. In this work, we prove the correctness of flattening for arbitrary DCTRSs w.r.t. *innermost* rewriting. As usual, the innermost rewrite relation, in symbols, $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$, is defined as the smallest binary relation satisfying the following: given ground terms $s, t \in \mathcal{T}(\mathcal{F})$, we have $s \xrightarrow{\mathsf{i}}_{\mathcal{R}} t$ iff there exist a position p in s such that no proper subterms of $s|_p$ are reducible, a rewrite rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, and a normalized ground substitution σ such that $s|_p = l\sigma$, $s_i\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t_i\sigma$, for all $i = 1, \ldots, n$, and $t = s[r\sigma]_p$.

▶ **Theorem 14.** *Let $\mathcal{R}$ be a DCTRS. If $\mathcal{R}'$ is obtained from $\mathcal{R}$ by a flattening step, then $\mathcal{R}'$ is a DCTRS and, for all ground terms s, t, we have $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$ iff $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}'} t$.*

Therefore, both a DCTRS and its basic version – obtained by applying a sequence of flattening steps – are equivalent w.r.t. innermost reduction. This justifies our use of basic DCTRSs in the remainder of this paper.

A nice property of basic DCTRSs is that one can consider reductions only at *topmost* positions. Formally, given a DCTRS $\mathcal{R}$, we say that $s \to_{p, l\to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}} t$ is a *top* reduction step if $p = \epsilon$, there is a ground substitution σ with $s = l\sigma$, $s_i\sigma \to^*_{\mathcal{R}} t_i\sigma$ for all $i = 1, \ldots, n$, $t = r\sigma$, and all the steps in $s_i\sigma \to^*_{\mathcal{R}} t_i\sigma$ for $i = 1, \ldots, n$ are also top reduction steps. We denote top reductions with $\xrightarrow{\epsilon}$ for standard rewriting, and $\overset{\epsilon}{\rightharpoonup}_{\mathcal{R}}, \overset{\epsilon}{\leftharpoondown}_{\mathcal{R}}$ for our reversible rewrite relations.

The following result basically states that $\xrightarrow{\mathsf{i}}$ and $\xrightarrow{\epsilon}$ are equivalent for basic DCTRSs:

▶ **Theorem 15.** *Let $\mathcal{R}$ be a DCTRS and $\mathcal{R}'$ be a basic DCTRS obtained from $\mathcal{R}$ by a sequence of flattening steps. Given ground terms s, t such that s is basic, we have $s \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} t$ iff $s \xrightarrow{\epsilon}{}^*_{\mathcal{R}'} t$.*

Therefore, when considering basic DCTRSs and top reductions, storing the reduced positions in the trace terms becomes redundant since they are always ϵ. Thus, in practice, one can consider simpler trace terms without positions, $\beta(\sigma, \pi_1, \ldots, \pi_n)$, that implicitly represent the trace term $\beta(\epsilon, \sigma, \pi_1, \ldots, \pi_n)$.

▶ **Example 16.** Consider the following TRS $\mathcal{R}$ defining addition and multiplication on natural numbers, and its associated basic DCTRS $\mathcal{R}'$:

$$\begin{array}{llrl} \mathcal{R} = \{ & \beta_1 : & \mathsf{a}(0, y) & \to y, \\ & \beta_2 : & \mathsf{a}(\mathsf{s}(x), y) & \to \mathsf{s}(\mathsf{a}(x, y)), \\ & \beta_3 : & \mathsf{m}(0, y) & \to 0, \\ & \beta_4 : & \mathsf{m}(\mathsf{s}(x), y) & \to \mathsf{a}(\mathsf{m}(x, y), y) \ \} \end{array} \qquad \begin{array}{llrl} \mathcal{R}' = \{ & \beta'_1 : & \mathsf{a}(0, y) & \to y, \\ & \beta'_2 : & \mathsf{a}(\mathsf{s}(x), y) & \to \mathsf{s}(z) \Leftarrow \mathsf{a}(x, y) \twoheadrightarrow z, \\ & \beta'_3 : & \mathsf{m}(0, y) & \to 0, \\ & \beta'_4 : & \mathsf{m}(\mathsf{s}(x), y) & \to \mathsf{a}(z, y) \Leftarrow \mathsf{m}(x, y) \twoheadrightarrow z \ \} \end{array}$$

For instance, given the following reduction in $\mathcal{R}$:

$$\mathsf{m}(\mathsf{s}(0), \mathsf{s}(0)) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{a}(\mathsf{m}(0, \mathsf{s}(0)), \mathsf{s}(0)) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{a}(0, \mathsf{s}(0)) \xrightarrow{\mathsf{i}}_{\mathcal{R}} \mathsf{s}(0)$$

we have the following counterpart in $\mathcal{R}'$:

$$\mathsf{m}(\mathsf{s}(0), \mathsf{s}(0)) \xrightarrow{\epsilon}_{\mathcal{R}'} \mathsf{a}(0, \mathsf{s}(0)) \xrightarrow{\epsilon}_{\mathcal{R}'} \mathsf{s}(0) \quad \text{with } \mathsf{m}(0, \mathsf{s}(0)) \xrightarrow{\epsilon}_{\mathcal{R}'} 0$$

[5] The positions p, q can be required to be different from ϵ, but this is not strictly necessary.

Trivially, all results in Section 3 hold for basic DCTRSs and top reductions starting from basic terms. The simpler trace terms without positions allow us to introduce appropriate injectivization and inversion transformations in the next section.

5 Reversibilization

In this section, we aim at *compiling* the reversible extension of rewriting into the system rules. Intuitively speaking, given a basic system $\mathcal{R}$, we aim at producing new systems $\mathcal{R}_f$ and $\mathcal{R}_b$ such that standard rewriting in $\mathcal{R}_f$, i.e., $\to_{\mathcal{R}_f}$, coincides with the forward reversible extension $\rightharpoonup_{\mathcal{R}}$ in the original system, and analogously $\to_{\mathcal{R}_b}$ is equivalent to $\leftharpoondown_{\mathcal{R}}$. Therefore, $\mathcal{R}_f$ can be seen as an *injectivization* of $\mathcal{R}$, and $\mathcal{R}_b$ can be seen as the *inversion* of $\mathcal{R}_f$.

5.1 Injectivization

Essentially, injectivization in our context amounts to add the traces to the rewrite rules, so that standard rewriting can be used:

▶ **Definition 17** (injectivization). Let $\mathcal{R}$ be a basic DCTRS. We produce a new CTRS $\mathcal{I}(\mathcal{R})$ by replacing each rule $\beta : l \to r \Leftarrow s_1 \twoheadrightarrow t_1, \ldots, s_n \twoheadrightarrow t_n$ of $\mathcal{R}$ by a new rule of the form

$$\langle l, ws\rangle \to \langle r, \beta(\overline{y}, \overline{w_n}) : ws\rangle \Leftarrow \langle s_1, [\,]\rangle \twoheadrightarrow \langle t_1, w_1\rangle, \ldots, \langle s_n, [\,]\rangle \twoheadrightarrow \langle t_n, w_n\rangle$$

in $\mathcal{I}(\mathcal{R})$, where $\{\overline{y}\} = (\mathcal{V}\mathsf{ar}(l)\backslash\mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n}) \cup \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(t_i)\backslash\mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}}))$ and both ws and $\overline{w_n}$ are fresh variables. Here, we assume that the variables of $\overline{y}$ are in lexicographic order.

Observe that there is a clear correspondence with the notion of reversible rewriting by only assuming that the reduced positions are always ϵ and, thus, they are not stored in the trace. Note also that, rather than storing a substitution, as in $\beta(\sigma, \pi_1, \ldots, \pi_n)$, we add the variables of interest to the trace term, $\beta(\overline{y}, \pi_1, \ldots, \pi_n)$, where $\overline{y}$ represent the domain of σ.

▶ **Example 18.** Consider again the DCTRS $\mathcal{R}$ from Example 6, which is already a basic DCTRS. Then, $\mathcal{R}_f = \mathcal{I}(\mathcal{R})$ is as follows:[6]

$$\begin{aligned}
\langle \mathsf{f}(x,y,m), ws\rangle &\to \langle \mathsf{s}(w), \beta_1(m,x,w_1,w_2) : ws\rangle \Leftarrow \langle \mathsf{h}(x), [\,]\rangle \twoheadrightarrow \langle x, w_1\rangle, \langle \mathsf{g}(y,\mathsf{4}), [\,]\rangle \twoheadrightarrow \langle w, w_2\rangle\\
\langle \mathsf{h}(\mathsf{0}), ws\rangle &\to \langle \mathsf{0}, \beta_2 : ws\rangle\\
\langle \mathsf{h}(\mathsf{1}), ws\rangle &\to \langle \mathsf{1}, \beta_3 : ws\rangle\\
\langle \mathsf{g}(x,y), ws\rangle &\to \langle x, \beta_4(y) : ws\rangle
\end{aligned}$$

The reversible step $\langle \mathsf{f}(\mathsf{0},\mathsf{2},\mathsf{4}), [\,]\rangle \stackrel{\epsilon}{\rightharpoonup}_{\mathcal{R}} \langle \mathsf{s}(\mathsf{2}), [\beta_1(\epsilon, \sigma', \pi_1, \pi_2)]\rangle$ with $\sigma' = \{m \mapsto \mathsf{4}, x \mapsto \mathsf{0}\}$, $\pi_1 = [\beta_2(\epsilon, id)]$ and $\pi_2 = [\beta_4(\epsilon, \{y \mapsto \mathsf{4}\})]$, has the following counterpart in $\mathcal{R}_f$:

$$\langle \mathsf{f}(\mathsf{0},\mathsf{2},\mathsf{4}), [\,]\rangle \stackrel{\epsilon}{\to}_{\mathcal{R}_f} \langle \mathsf{s}(\mathsf{2}), [\beta_1(\mathsf{4},\mathsf{0},[\beta_2],[\beta_4(\mathsf{4})])]\rangle \quad \text{with} \quad \begin{array}{l} \langle \mathsf{h}(\mathsf{0}), [\,]\rangle \stackrel{\epsilon}{\to}_{\mathcal{R}_f} \langle \mathsf{0}, [\beta_2]\rangle \;\; \text{and} \\ \langle \mathsf{g}(\mathsf{2},\mathsf{4}), [\,]\rangle \stackrel{\epsilon}{\to}_{\mathcal{R}_f} \langle \mathsf{2}, [\beta_4(\mathsf{4})]\rangle \end{array}$$

As can be seen in the example above, the trace terms that occur in a reversible rewrite derivation with a basic DCTRS $\mathcal{R}$ and those that occur in a top reduction with $\mathcal{I}(\mathcal{R})$ are similar but not exactly equal. We formalize their relation as follows:

[6] We will write just β instead of $\beta()$ when no argument is required.

▶ **Definition 19.** Given a trace π, we define $\widehat{\pi}$ recursively as follows:

$$\widehat{\pi} \;=\; \begin{cases} [\,] & \text{if } \pi = [\,] \\ \beta(\overline{t_m}, \widehat{\pi_1}, \ldots, \widehat{\pi_n}) & \text{if } \pi = \beta(\{y_1 \mapsto t_1, \ldots, y_m \mapsto t_m\}, \pi_1, \ldots, \pi_n) \end{cases}$$

where we assume that the variables $\overline{y_m}$ are in lexicographic order.

The following result states the correctness of our injectivization transformation:

▶ **Theorem 20.** *Let $\mathcal{R}$ be a basic DCTRS and $\mathcal{R}_f = \mathcal{I}(\mathcal{R})$ be its injectivization. Then $\mathcal{R}_f$ is a basic DCTRS and, given a basic ground term s, we have $\langle s, [\,]\rangle \overset{\epsilon}{\rightharpoonup}{}^*_{\mathcal{R}} \langle t, \pi\rangle$ iff $\langle s, [\,]\rangle \overset{\epsilon}{\rightarrow}{}^*_{\mathcal{R}_f} \langle t, \widehat{\pi}\rangle$.*

5.2 Inversion

In general, function inversion is a difficult and often undecidable problem (see, e.g., [24, 22, 10, 23]). For injectivized systems, though, it becomes straightforward:

▶ **Definition 21** (inversion)**.** Let $\mathcal{R}$ be a basic DCTRS and let $\mathcal{R}_f = \mathcal{I}(\mathcal{R})$ be its injectivization. Then, the inverse system, $\mathcal{R}_b = \mathcal{I}^{-1}(\mathcal{R}_f)$ is obtained from $\mathcal{R}_f$ by transforming every rule

$$\langle l, ws\rangle \to \langle r, \beta(\overline{y}, \overline{w_n}) : ws\rangle \Leftarrow \langle s_1, [\,]\rangle \twoheadrightarrow \langle t_1, w_1\rangle, \ldots, \langle s_n, [\,]\rangle \twoheadrightarrow \langle t_n, w_n\rangle$$

into a rule of the form

$$\langle r, \beta(\overline{y}, \overline{w_n}) : ws\rangle^{-1} \to \langle l, ws\rangle^{-1} \Leftarrow \langle t_n, w_n\rangle^{-1} \twoheadrightarrow \langle s_n, [\,]\rangle^{-1}, \ldots, \langle t_1, w_1\rangle^{-1} \twoheadrightarrow \langle s_1, [\,]\rangle^{-1}$$

We use a different symbol $\langle \_, \_\rangle^{-1}$ since we may want to use both the forward and the backward functions in the same system.

▶ **Example 22.** Consider the DCTRS $\mathcal{R}_f$ from Example 18. Then, its inversion $\mathcal{R}_b = \mathcal{I}^{-1}(\mathcal{R}_f)$ is defined as follows:

$$\begin{array}{rcl} \langle \mathsf{s}(w), \beta_1(m, x, w_1, w_2) : ws\rangle^{-1} & \to & \langle \mathsf{f}(x, y, m), ws\rangle^{-1} \Leftarrow \langle w, w_2\rangle^{-1} \twoheadrightarrow \langle \mathsf{g}(y, \mathsf{4}), [\,]\rangle^{-1}, \\ & & \qquad\qquad\qquad\quad\;\; \langle x, w_1\rangle^{-1} \twoheadrightarrow \langle \mathsf{h}(x), [\,]\rangle^{-1} \\ \langle \mathsf{0}, \beta_2 : ws\rangle^{-1} & \to & \langle \mathsf{h}(\mathsf{0}), ws\rangle^{-1} \\ \langle \mathsf{1}, \beta_3 : ws\rangle^{-1} & \to & \langle \mathsf{h}(\mathsf{1}), ws\rangle^{-1} \\ \langle x, \beta_4(y) : ws\rangle^{-1} & \to & \langle \mathsf{g}(x, y), ws\rangle^{-1} \end{array}$$

The correctness of the inversion transformation is then stated as follows:

▶ **Theorem 23.** *Let $\mathcal{R}$ be a basic DCTRS, $\mathcal{R}_f = \mathcal{I}(\mathcal{R})$ its injectivization, and $\mathcal{R}_b = \mathcal{I}^{-1}(\mathcal{R}_f)$ the inversion of $\mathcal{R}_f$. Then, $\mathcal{R}_b$ is a basic DCTRS and, given a basic or constructor ground term t and a trace π with $\langle t, \pi\rangle$ safe, we have $\langle t, \pi\rangle \overset{\epsilon}{\leftharpoondown}{}^*_{\mathcal{R}} \langle s, [\,]\rangle$ iff $\langle t, \widehat{\pi}\rangle^{-1} \overset{\epsilon}{\rightarrow}{}^*_{\mathcal{R}_b} \langle s, [\,]\rangle^{-1}$.*

5.3 An Improved Reversibilization Procedure

Using the transformations introduced so far, given a DCTRS $\mathcal{R}$, we can produce a basic DCTRS $\mathcal{R}'$, which can then be injectivized $\mathcal{I}(\mathcal{R}')$ and reversed $\mathcal{I}^{-1}(\mathcal{I}(\mathcal{R}'))$. Although one can find several applications for $\mathcal{I}(\mathcal{R}')$ and $\mathcal{I}^{-1}(\mathcal{I}(\mathcal{R}'))$, we note that these systems are aimed at mimicking the reversible relations $\rightharpoonup_{\mathcal{R}'}$ and $\leftharpoondown_{\mathcal{R}'}$, rather than computing injective and inverse versions of the *functions* defined in $\mathcal{R}'$. In other words, $\mathcal{I}(\mathcal{R}')$ defines a single function $\langle \_, \_\rangle$ and $\mathcal{I}^{-1}(\mathcal{I}(\mathcal{R}'))$ a single function $\langle \_, \_\rangle^{-1}$. Now, we refine these transformations so that one can actually produce injective and inverse versions of the original functions.

In principle, one could consider that the injectivization of a rule of the form[7]

$$\beta : \mathsf{f}(\overline{s_0}) \to r \Leftarrow \mathsf{f}_1(\overline{s_1}) \twoheadrightarrow t_1, \ldots, \mathsf{f}_\mathsf{n}(\overline{s_n}) \twoheadrightarrow t_n$$

will produce the following rule

$$\mathsf{f}^i(\overline{s_0}, ws) \to \langle r, \beta(\overline{y}, \overline{w_n}) : ws\rangle \Leftarrow \mathsf{f}_1^i(\overline{s_1}, [\,]) \twoheadrightarrow \langle t_1, w_1\rangle \ldots, \mathsf{f}_n^i(\overline{s_n}, [\,]) \twoheadrightarrow \langle t_n, w_n\rangle$$

where traces are now added as an additional argument of each function. The following example, though, illustrates that this is not correct in general.

▶ **Example 24.** Consider the following basic DCTRS $\mathcal{R}$:

$$\begin{array}{lrcl}
\beta_1 : & \mathsf{f}(x,y) & \to & z \Leftarrow \mathsf{h}(y) \twoheadrightarrow w,\ \mathsf{first}(x,w) \twoheadrightarrow z \\
\beta_2 : & \mathsf{h}(\mathsf{0}) & \to & \mathsf{0} \\
\beta_3 : & \mathsf{first}(x,y) & \to & x
\end{array}$$

together with the following top reduction:

$$\begin{array}{ll}
\mathsf{f}(\mathsf{2},\mathsf{1}) \xrightarrow{\epsilon}_{\mathcal{R}} \mathsf{2} & \text{with } \sigma = \{x \mapsto \mathsf{2}, y \mapsto \mathsf{1}, w \mapsto \mathsf{h}(\mathsf{1}), z \mapsto \mathsf{2}\} \\
 & \text{where } \mathsf{h}(y)\sigma = \mathsf{h}(\mathsf{1}) \xrightarrow{\epsilon}{}^{*}_{\mathcal{R}} \mathsf{h}(\mathsf{1}) = w\sigma \text{ and } \mathsf{first}(x,w)\sigma = \mathsf{first}(\mathsf{2},\mathsf{h}(\mathsf{1})) \xrightarrow{\epsilon}{}^{*}_{\mathcal{R}} \mathsf{2} = z\sigma
\end{array}$$

The improved injectivization above would return the following basic DCTRS:

$$\begin{array}{rcl}
\mathsf{f}^i(x,y,ws) & \to & \langle z, \beta_1(w_1,w_2) : ws\rangle \Leftarrow \mathsf{h}^i(y,[\,]) \twoheadrightarrow \langle w, w_1\rangle,\ \mathsf{first}^i(x,w,[\,]) \twoheadrightarrow \langle z, w_2\rangle \\
\mathsf{h}^i(\mathsf{0}, ws) & \to & \langle \mathsf{0}, \beta_2 : ws\rangle \\
\mathsf{first}^i(x,y,ws) & \to & \langle x, \beta_3(y) : ws\rangle
\end{array}$$

Unfortunately, the corresponding reduction for $\mathsf{f}^i(\mathsf{2},\mathsf{1},[\,])$ above cannot be done in this system since $\mathsf{h}^i(\mathsf{1},[\,])$ cannot be reduced to $\langle \mathsf{h}^i(\mathsf{1}), [\,]\rangle$.

In order to solve the above drawback, one could *complete* the function definitions with rules that reduce each irreducible term t to a tuple of the form $\langle t, [\,]\rangle$. Although we find it a promising idea for future work, in this paper we propose a simpler approach. In the following, we consider a refinement of innermost reduction where only constructor substitutions are computed. Formally, the constructor reduction relation, $\xrightarrow{\mathsf{c}}$, is defined as follows: given ground terms $s, t \in \mathcal{T}(\mathcal{F})$, we have $s \xrightarrow{\mathsf{c}}_{\mathcal{R}} t$ iff there exist a position p in s such that no proper subterms of $s|_p$ are reducible, a rewrite rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n} \in \mathcal{R}$, and a ground *constructor* substitution σ such that $s|_p = l\sigma$, $s_i\sigma \xrightarrow{\mathsf{c}}{}^{*}_{\mathcal{R}} t_i\sigma$ for all $i = 1, \ldots, n$, and $t = s[r\sigma]_p$.

Furthermore, we also require a further requirement on DCTRSs: we say that $\mathcal{R}$ is a c-DCTRS (a pure-constructor system [20]) if $\mathcal{R}$ is a DCTRS and, for any rule $l \to r \Leftarrow \overline{s_n \twoheadrightarrow t_n}$, we have that $l, \overline{s_n}$ are basic terms and $r, \overline{t_n}$ are constructor terms. Note that requiring $\overline{s_n}$ to be basic terms (thus excluding constructor terms) is not a real restriction since any equation of the form $s \twoheadrightarrow t$, with s (and t) a constructor term, can be removed by matching s and t, removing the equation, and applying the matching substitution to the rule (cf. [23]).

▶ **Definition 25** (refined injectivization)**.** Let $\mathcal{R}$ be a basic c-DCTRS. We produce a new CTRS $\mathbf{I}(\mathcal{R})$ by replacing each rule $\beta : \mathsf{f}(\overline{s_0}) \to r \Leftarrow \mathsf{f}_1(\overline{s_1}) \twoheadrightarrow t_1, \ldots, \mathsf{f}_\mathsf{n}(\overline{s_n}) \twoheadrightarrow t_n$ of $\mathcal{R}$ by a new rule of the form

$$\mathsf{f}^i(\overline{s_0}) \to \langle r, \beta(\overline{y}, \overline{w_n})\rangle \Leftarrow \mathsf{f}_1^i(\overline{s_1}) \twoheadrightarrow \langle t_1, w_1\rangle, \ldots, \mathsf{f}_n^i(\overline{s_n}) \twoheadrightarrow \langle t_n, w_n\rangle$$

in $\mathbf{I}(\mathcal{R})$, where $\{\overline{y}\} = (\mathcal{V}\mathsf{ar}(l) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_n}, \overline{t_n})) \cup \bigcup_{i=1}^{n} \mathcal{V}\mathsf{ar}(t_i) \backslash \mathcal{V}\mathsf{ar}(r, \overline{s_{i+1,n}})$ and $\overline{w_n}$ are fresh variables. Here, we assume that the variables of $\overline{y}$ are in lexicographic order.

[7] By abuse, here we let $\overline{s_0}, \ldots, \overline{s_n}$ denote sequences of terms of arbitrary length.

Observe that now we do not need to keep a trace in each term, but only a single trace term since all reductions finish in one step in a basic c-DCTRS. By abuse of notation, we still use the notation $\widehat{\pi}$ when π is a trace term instead of a trace.

▶ **Theorem 26.** *Let $\mathcal{R}$ be a basic c-DCTRS and $\mathcal{R}_f = \mathbf{I}(\mathcal{R})$ be its injectivization. Then $\mathcal{R}_f$ is a basic c-DCTRS and, given a basic ground term $\mathsf{f}(\overline{s})$ and a constructor ground term t, we have $\langle \mathsf{f}(\overline{s}), [\,] \rangle \overset{\mathsf{c}}{\rightharpoondown}_{\mathcal{R}} \langle t, \pi \rangle$ iff $\mathsf{f}^i(\overline{s}) \overset{\mathsf{c}}{\rightarrow}_{\mathcal{R}_f} \langle t, \widehat{\pi} \rangle$.*

Now, the refined version of the inversion transformation proceeds as follows:

▶ **Definition 27** (refined inversion)**.** Let $\mathcal{R}$ be a basic c-DCTRS and $\mathcal{R}_f = \mathbf{I}(\mathcal{R})$ be its injectivization. The inverse system $\mathcal{R}_b = \mathbf{I}^{-1}(\mathcal{R}_f)$ is obtained from $\mathcal{R}_f$ by replacing each rule

$$\mathsf{f}^i(\overline{s_0}) \to \langle r, \beta(\overline{y}, \overline{w_n}) \rangle \Leftarrow \mathsf{f}^i_1(\overline{s_1}) \twoheadrightarrow \langle t_1, w_1 \rangle, \ldots, \mathsf{f}^i_n(\overline{s_n}) \twoheadrightarrow \langle t_n, w_n \rangle$$

of $\mathcal{R}_f$ by a new rule of the form

$$\mathsf{f}^{-1}(r, \beta(\overline{y}, \overline{w_n})) \to \langle \overline{s_0} \rangle \Leftarrow \mathsf{f}^{-1}_n(t_n, w_n) \twoheadrightarrow \langle \overline{s_n} \rangle, \ldots, \mathsf{f}^{-1}_1(t_1, w_1) \twoheadrightarrow \langle \overline{s_1} \rangle$$

in $\mathbf{I}^{-1}(\mathcal{R}_f)$. Here, we assume that the variables of $\overline{y}$ are in lexicographic order.

▶ **Example 28.** Consider again the basic DCTRS of Example 6 which is a c-DCTRS. The injectivization transformation $\mathbf{I}$, returns the following c-DCTRS $\mathcal{R}_f$:

$$\begin{array}{l}
\mathsf{f}^i(x, y, m) \to \langle \mathsf{s}(w), \beta_1(m, x, w_1, w_2) \rangle \Leftarrow \mathsf{h}^i(x) \twoheadrightarrow \langle x, w_1 \rangle, \mathsf{g}^i(y, \mathsf{4}) \twoheadrightarrow \langle w, w_2 \rangle \\
\qquad \mathsf{h}^i(\mathsf{0}) \to \langle \mathsf{0}, \beta_2 \rangle \qquad \mathsf{h}^i(\mathsf{1}) \to \langle \mathsf{1}, \beta_3 \rangle \qquad \mathsf{g}^i(x, y) \to \langle x, \beta_4(y) \rangle
\end{array}$$

Then, inversion with $\mathbf{I}^{-1}$ produces the following c-DCTRS $\mathcal{R}_b$:

$$\begin{array}{l}
\mathsf{f}^{-1}(\mathsf{s}(w), \beta_1(m, x, w_1, w_2)) \to \langle x, y, m \rangle \Leftarrow \mathsf{g}^{-1}(w, w_2) \twoheadrightarrow \langle y, \mathsf{4} \rangle, \mathsf{h}^{-1}(x, w_1) \twoheadrightarrow \langle x \rangle \\
\qquad \mathsf{h}^{-1}(\mathsf{0}, \beta_2) \to \langle \mathsf{0} \rangle \qquad \mathsf{h}^{-1}(\mathsf{1}, \beta_3) \to \langle \mathsf{1} \rangle \qquad \mathsf{g}^{-1}(x, \beta_4(y)) \to \langle x, y \rangle
\end{array}$$

Finally, the correctness of the refined inversion transformation is stated as follows:

▶ **Theorem 29.** *Let $\mathcal{R}$ be a basic c-DCTRS, $\mathcal{R}_f = \mathbf{I}(\mathcal{R})$ its injectivization, and $\mathcal{R}_b = \mathbf{I}^{-1}(\mathcal{R}_f)$ the inversion of $\mathcal{R}_f$. Then, $\mathcal{R}_b$ is a basic c-DCTRS and, given a basic ground term $\mathsf{f}(\overline{s})$ and a constructor ground term t with $\langle t, \pi \rangle$ a safe pair, we have $\langle t, \pi \rangle \overset{\mathsf{c}}{\leftharpoondown}_{\mathcal{R}} \langle \mathsf{f}(\overline{s}), [\,] \rangle$ iff $\mathsf{f}^{-1}(t, \widehat{\pi}) \overset{\mathsf{c}}{\rightarrow}_{\mathcal{R}_b} \langle \overline{s} \rangle$.*

5.4 Applications

A potential application of our reversibilization technique is in the context of *bidirectional program transformation* (see, e.g., [8, 15] and references therein). This technique applies when we have a data structure, called the *source*, which is transformed to another data structure, called the *view*. Typically, we have a *view function* that takes the source and returns the corresponding view. Here, the bidirectionalization transformation aims at defining a *backward transformation* that takes a modified view, and returns the corresponding *modified* source. Defining a view function and a backward transformation that form a bidirectional transformation is not easy, and therefore our reversibilization technique can be useful in this context. For instance, let us assume that we have a view function, view, that takes a source and returns the corresponding view, and is defined by means of a c-DCTRS. Then, following our approach, we can produce an injective version, say view^i, and an inverse version view^{-1}. Now, one could solve the view update problem with the following function:

$$\mathsf{upd}(s, v') \to s' \Leftarrow \mathsf{view}^i(s) \twoheadrightarrow \langle v, \pi\rangle,\ \mathsf{view}^{-1}(v', \pi) \twoheadrightarrow \langle s'\rangle$$

where s is the original source, v' is the updated view, and s' – the returned value – is the corresponding updated source.

A more challenging application is the *reversibilization of cellular automata* [19]. In a cellular automaton, evolution is determined by some fixed rule (generally, a mathematical function) that determines the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood. If we consider a rewrite system for defining this rule, our approach can help us to produce *reversible* cellular automata from irreversible ones, an important feature in this field.

As in [18], we can consider a one-dimensional irreversible cellular automaton where each cell has two neighbours. The cellular automaton can be represented as a (potentially infinite) array of cells. Consider, e.g., the following function to control the evolution of a cellular automaton whose cells can only take value $\square$ or $\blacksquare$ (it is known as *rule 150*):

$$\begin{array}{llll} \beta_1 : \mathsf{f}(\square,\square,\square) \to \square & \beta_2 : \mathsf{f}(\square,\square,\blacksquare) \to \blacksquare & \beta_3 : \mathsf{f}(\square,\blacksquare,\square) \to \blacksquare & \beta_4 : \mathsf{f}(\square,\blacksquare,\blacksquare) \to \square \\ \beta_5 : \mathsf{f}(\blacksquare,\square,\square) \to \blacksquare & \beta_6 : \mathsf{f}(\blacksquare,\square,\blacksquare) \to \square & \beta_7 : \mathsf{f}(\blacksquare,\blacksquare,\square) \to \square & \beta_8 : \mathsf{f}(\blacksquare,\blacksquare,\blacksquare) \to \blacksquare \end{array}$$

Evolution takes place then by applying *simultaneously* the above function to every cell and its neighbours. Following our approach, we can injectivize function f as follows:

$$\begin{array}{llll} \mathsf{f}^i(\square,\square,\square) \to \langle\square,\beta_1\rangle & \mathsf{f}^i(\square,\square,\blacksquare) \to \langle\blacksquare,\beta_2\rangle & \mathsf{f}^i(\square,\blacksquare,\square) \to \langle\blacksquare,\beta_3\rangle & \mathsf{f}^i(\square,\blacksquare,\blacksquare) \to \langle\square,\beta_4\rangle \\ \mathsf{f}^i(\blacksquare,\square,\square) \to \langle\blacksquare,\beta_5\rangle & \mathsf{f}^i(\blacksquare,\square,\blacksquare) \to \langle\square,\beta_6\rangle & \mathsf{f}^i(\blacksquare,\blacksquare,\square) \to \langle\square,\beta_7\rangle & \mathsf{f}^i(\blacksquare,\blacksquare,\blacksquare) \to \langle\blacksquare,\beta_8\rangle \end{array}$$

We can then consider that cells include a list of rule labels, so that the following evolution:

$$[\ \square,\ \square,\ \blacksquare,\ \square,\ \square\ ] \Rightarrow [\ \square,\ \blacksquare,\ \blacksquare,\ \blacksquare,\ \square\ ] \Rightarrow [\ \blacksquare,\ \square,\ \blacksquare,\ \square,\ \blacksquare\ ] \Rightarrow \cdots$$

with the original cellular automaton, are now represented as follows:

$$\begin{array}{ll} & [\ \langle\square,\ [\,]\rangle,\ \langle\square,\ [\,]\rangle,\ \langle\blacksquare,\ [\,]\rangle,\ \langle\square,\ [\,]\rangle,\ \langle\square,\ [\,]\rangle\ ] \\ \Rightarrow & [\ \langle\square,\ [\beta_1]\rangle,\ \langle\blacksquare,\ [\beta_2]\rangle,\ \langle\blacksquare,\ [\beta_3]\rangle,\ \langle\blacksquare,\ [\beta_5]\rangle,\ \langle\square,\ [\beta_1]\rangle\ ] \\ \Rightarrow & [\ \langle\blacksquare,\ [\beta_2,\beta_1]\rangle,\ \langle\square,\ [\beta_4,\beta_2]\rangle,\ \langle\blacksquare,\ [\beta_8,\beta_3]\rangle,\ \langle\square,\ [\beta_7,\beta_5]\rangle,\ \langle\blacksquare,\ [\beta_5,\beta_1]\rangle\ ] \\ \Rightarrow & \cdots \end{array}$$

Thus, the new cellular automaton is clearly reversible. With respect to the previous approach in [18], our reversibilization process is more intuitive (the one proposed in [18] is rather *ad-hoc*), does not increase the number of steps (while in [18] $2n + k$ steps are required for each step of the original cellular automaton, where k is a constant and n is the number of non-empty cells in the cellular automaton), and its correctness is trivial by construction (correctness is only sketched in [18]). On the other hand, our approach increases the size of the cellular automaton by a factor that depends in principle on the length of the computation (but not on the size of the cellular automaton).

A more detailed description of the above applications can be found in [21].

6 Related Work

Regarding reversible computing, one can already find a number of references in the literature (e.g., [4, 9, 30]). Our work starts with the well-known approach of Landauer [14] which proposes that saving the history of a computation makes it reversible. This approach to reversibilization has already been considered in the past and has been applied in different contexts and computational models, e.g., a probabilistic guarded command language [32], a low level virtual machine [25], the call-by-name lambda calculus [12, 13], cellular automata [28, 18], combinatory logic [7], a flowchart language [31], or a functional language [15, 27].

However, to the best of our knowledge, this is the first work that considers a reversible extension of (conditional) term rewriting. We note, though, that Abramsky [1] introduced an approach to reversible computation with *pattern matching automata*, which could also be represented in terms of standard notions of term rewriting. His approach, though, requires a condition called *biorthogonality* (which, in particular, implies injectivity), a condition that would be overly restrictive in our setting. Roughly speaking, in our approach we achieve a similar class of systems through injectivization from more general systems.

Another related work are the papers by Matsuda et al [15, 16] which focus on *bidirectional program transformation* for functional programs. In [15], functional programs corresponding to linear and *right-treeless*[8] constructor TRSs are considered. In [16], the previous class is extended to those corresponding to left-linear right-treeless TRSs. The methods in [15, 16] for injectivization and inversion consider a more restricted class of systems than those considered in this paper; on the other hand, they apply a number of analyses to improve the result, which explains the smaller traces in their approach. Besides being more general, we consider that our approach gives better insights to understand the need for the requirements of the program transformations. Finally, [27] introduces a transformation for functional programs which has some similarities with both the approach of [15] and our improved transformation in Section 5.3; in contrast, though, [27] also applies the Bennett *trick* [3] in order to avoid some unnecessary information.

7 Discussion and Future Work

In this paper, we have introduced a reversible extension of term rewriting. In order to keep our approach as general as possible, we have initially considered DCTRSs as input systems, and proved the soundness and reversibility of our extension of rewriting. Then, in order to introduce a reversibilization transformation for these systems, we have also presented a transformation from DCTRSs to basic DCTRSs which is correct for innermost reduction. Finally, for constructor reduction, we are able to further refine our reversibilization transformations. We have successfully applied our approach in the context of bidirectional program transformation and the reversibilization of cellular automata.

As for future work, we plan to investigate restricted classes of CTRSs so that we can further reduce the size of the traces. In particular, we will look for conditions under which we can remove the variable bindings, the rule label, or even the complete trace. For this purpose, we will consider non-erasing rules and injective functions, since we think that there are different contexts where these conditions arise quite naturally.

Acknowledgements. We thank the anonymous reviewers for their useful comments and suggestions to improve this paper.

[8] There are no nested defined symbols in the right-hand sides, and, moreover, any term rooted by a defined function in the right-hand sides can only take different variables as its proper subterms.

References

1 S. Abramsky. A structural approach to reversible computation. *Theor. Comput. Sci.*, 347(3):441–464, 2005.

2 F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

3 C. H. Bennet. Logical reversibility of computation. *IBM Journal of Research and Development*, 17:525–532, 1973.

4 C. H. Bennett. Notes on the history of reversible computation. *IBM Journal of Research and Development*, 44(1):270–278, 2000.

5 A. Bérut, A. Arakelyan, A. Petrosyan, S. Ciliberto, R. Dillenschneider, and E. Lutz. Experimental verification of Landauer's principle linking information and thermodynamics. *Nature*, 483:187–189, 2012.

6 P. Crescenzi and C. H. Papadimitriou. Reversible simulation of space-bounded computations. *Theor. Comput. Sci.*, 143(1):159–165, 1995.

7 A. Di Pierro, C. Hankin, and H. Wiklicky. Reversible combinatory logic. *Mathematical Structures in Computer Science*, 16(4):621–637, 2006.

8 N. Foster, K. Matsuda, and J. Voigtländer. Three complementary approaches to bidirectional programming. In Jeremy Gibbons, editor, *Generic and Indexed Programming – International Spring School, SSGIP 2010, Oxford, UK, March 22-26, 2010, Revised Lectures*, volume 7470 of *Lecture Notes in Computer Science*, pages 1–46. Springer, 2012.

9 M. P. Frank. Introduction to reversible computing: motivation, progress, and challenges. In Nader Bagherzadeh, Mateo Valero, and Alex Ramírez, editors, *Proceedings of the Second Conference on Computing Frontiers*, pages 385–390. ACM, 2005.

10 R. Glück and M. Kawabe. A method for automatic program inversion based on LR(0) parsing. *Fundam. Inform.*, 66(4):367–395, 2005.

11 N. Hirokawa and G. Moser. Automated Complexity Analysis Based on the Dependency Pair Method. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Proc. of IJCAR 2008*, volume 5195 of *Lecture Notes in Computer Science*, pages 364–379. Springer, 2008.

12 L. Huelsbergen. A logically reversible evaluator for the call-by-name lambda calculus. In T. Toffoli and M. Biafore, editors, *Proc. of PhysComp96*, pages 159–167. New England Complex Systems Institute, 1996.

13 W. E. Kluge. A reversible SE(M)CD machine. In Pieter W. M. Koopman and Chris Clack, editors, *Proc. of the 11th International Workshop on the Implementation of Functional Languages, IFL'99. Selected Papers*, volume 1868 of *Lecture Notes in Computer Science*, pages 95–113. Springer, 2000.

14 R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5:183–191, 1961.

15 K. Matsuda, Z. Hu, K. Nakano, M. Hamana, and M. Takeichi. Bidirectionalization transformation based on automatic derivation of view complement functions. In R. Hinze and N. Ramsey, editors, *Proc. of the 12th ACM SIGPLAN International Conference on Functional Programming, ICFP 2007*, pages 47–58. ACM, 2007.

16 K. Matsuda, Z. Hu, K. Nakano, M. Hamana, and M. Takeichi. Bidirectionalizing programs with duplication through complementary function derivation. *Computer Software*, 26(2):56–75, 2009. In Japanese.

17 A. Middeldorp and E. Hamoen. Completeness results for basic narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5:213–253, 1994.

18 K. Morita. Reversible simulation of one-dimensional irreversible cellular automata. *Theor. Comput. Sci.*, 148(1):157–163, 1995.

19 K. Morita. Computation in reversible cellular automata. *Int. J. General Systems*, 41(6):569–581, 2012.

20 M. Nagashima, M. Sakai, and T. Sakabe. Determinization of conditional term rewriting systems. *Theor. Comput. Sci.*, 464:72–89, 2012.

21 N. Nishida, A. Palacios, and G. Vidal. Reversible term rewriting: foundations and applications. Technical report, DSIC, UPV, 2016. Available from the following URL: `http://users.dsic.upv.es/~gvidal/german/rr16/`.

22 N. Nishida, M. Sakai, and T. Sakabe. Partial inversion of constructor term rewriting systems. In Jürgen Giesl, editor, *Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA 2005)*, volume 3467 of *Lecture Notes in Computer Science*, pages 264–278. Springer, 2005.

23 N. Nishida and G. Vidal. Program inversion for tail recursive functions. In Manfred Schmidt-Schauß, editor, *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications (RTA 2011)*, volume 10 of *LIPIcs*, pages 283–298. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2011.

24 A. Romanenko. The generation of inverse functions in Refal. In *Proc. of IFIP TC2 Workshop on Partial Evaluation and Mixed Computation*, pp. 427–444, North-Holland, 1988.

25 B. Stoddart, R. Lynas, and F. Zeyda. A virtual machine for supporting reversible probabilistic guarded command languages. *Electr. Notes Theor. Comput. Sci.*, 253(6):33–56, 2010.

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

27 M. K. Thomsen and H. B. Axelsen. Interpretation and programming of the reversible functional language RFUN. In *Proc. of the 27th International Symposium on Implementation and Application of Functional Languages (IFL 2015)*. Springer, 2016. To appear.

28 T. Toffoli. Computation and construction universality of reversible cellular automata. *J. Comput. Syst. Sci.*, 15(2):213–231, 1977.

29 T. Yamakami. One-way reversible and quantum finite automata with advice. *Inf. Comput.*, 239:122–148, 2014.

30 T. Yokoyama. Reversible computation and reversible programming languages. *Electr. Notes Theor. Comput. Sci.*, 253(6):71–81, 2010.

31 T. Yokoyama, H.B. Axelsen, and R. Glück. Fundamentals of reversible flowchart languages. *Theor. Comput. Sci.*, 611:87–115, 2016.

32 P. Zuliani. Logical reversibility. *IBM Journal of Research and Development*, 45(6):807–818, 2001.

Certifying Confluence of Almost Orthogonal CTRSs via Exact Tree Automata Completion[*]

Christian Sternagel[1] and Thomas Sternagel[2]

1 University of Innsbruck, Innsbruck, Austria
christian.sternagel@uibk.ac.at

2 University of Innsbruck, Innsbruck, Austria
thomas.sternagel@uibk.ac.at

Abstract

Suzuki et al. showed that properly oriented, right-stable, orthogonal, and oriented conditional term rewrite systems with extra variables in right-hand sides are confluent. We present our Isabelle/HOL formalization of this result, including two generalizations. On the one hand, we relax proper orientedness and orthogonality to extended proper orientedness and almost orthogonality modulo infeasibility, as suggested by Suzuki et al. On the other hand, we further loosen the requirements of the latter, enabling more powerful methods for proving infeasibility of conditional critical pairs. Furthermore, we formalized a construction by Jacquemard that employs exact tree automata completion for non-reachability analysis and apply it to certify infeasibility of conditional critical pairs. Combining these two results and extending the conditional confluence checker ConCon accordingly, we are able to automatically prove and certify confluence of an important class of conditional term rewrite systems.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.1 Mathematical Logic, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases certification, conditional term rewriting, confluence, infeasible critical pairs, non-reachability

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.29

1 Introduction

Today there are a number of tools in existence which allow us to conveniently check various properties of standard term rewrite systems (TRSs). To not just rely on the trustworthiness and programming-prowess of the tool-authors, these tools are progressively accompanied by certifiers, that is, computer-verified programs which rigorously assure correctness of a tool's output with respect to a given input. The prevalent procedure for the development of certifiers comprises the following two phases: First, employ a proof assistant (in our case Isabelle/HOL [9]) in order to formalize the underlying theory, resulting in a *formal library* (in our case IsaFoR,[1] an *Isabelle/HOL Formalization of Rewriting*). Then, verify a program using this library, resulting in the actual certifier (in our case CeTA [17]).

Just for clarification, by *formalizing the underlying theory*, we mean that we take known proofs and definitions from the literature as well as our own results, scrutinize them, fill in the gaps, and provide such a level of detail that we arrive at a mechanized proof that is accepted by a proof assistant. Against common belief, such mechanized proofs are not necessarily

[*] The research described in this paper is supported by FWF (Austrian Science Fund) project P27502.

[1] http://cl-informatik.uibk.ac.at/software/ceta/

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 29; pp. 29:1–29:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

inscrutable to humans. On the contrary, especially Isabelle/HOL's *proof documents* are highly structured and, provided some practice, are sometimes easier to follow than their paper-originals (for the best reading experience we recommend Isabelle/jEdit [19] for browsing). More often than not, at least some minor inaccuracies and sometimes even proper errors in published proofs are exposed during the process of formalization (e.g., in previous work [14] one of the authors detected a missing left-linearity assumption in some of his yet earlier work [11]). So the benefits of formalization are threefold:

1. By scrutinizing known results, we gain a better understanding and clarify inaccuracies and sometimes even correct errors.
2. We obtain computer-checkable theories which may be used to generate certifiers for the underlying results as well as to build new results on top of them.
3. Using such certifiers we are able to increase the reliability of tools that build on the formalized theory and also expose errors in the tools themselves.

As mentioned earlier the formalization of standard term rewriting is ongoing work since almost a decade, with many widely used results.

Ultimately we strife to establish the same state of the art for conditional term rewrite systems (CTRSs). We already embarked on this journey in [12] and further this enterprise by generalizing and extending our previous work.

The developments we describe in this article are part of the IsaFoR library and are freely available for inspection, see theories `Conditional_Rewriting/Level_Confluence.thy` and `Tree_Automata/Exact_Tree_Automata_Completion.thy` and their `*_Impl.thy` variants.

Contribution. The following three tasks are the original contributions of this work. We already formalized the result that right-stable, properly oriented, almost orthogonal, oriented 3-CTRSs are confluent by Suzuki et al. [16] in previous work [12]. (1) Here, we extend the syntactical part of the criterion to be applicable to *extended properly oriented* CTRSs. Moreover, we revisit the definition of *almost orthogonality* and relax it in a way that we now may employ new infeasibility criteria (Sections 3 and 4.3). Moreover, we shortly revisit non-reachability and non-joinability via `tcap` (Section 4.1) with an eye towards certification. (2) Additionally, we formalized the known result that reachability is decidable for linear and growing TRSs by Jacquemard [8] (Section 4.2). We use this to check for infeasibility of conditional critical pairs. (3) Finally, we incorporated the above findings in the certifier CeTA (Section 4.3) as well as the conditional confluence checker ConCon [15] (Section 5), so we are able to certify a large portion of the confluence proofs which are generated by ConCon.

Related Work. Felgenhauer and Thiemann [3] formalize so called state-compatible automata and thereby also show that it is decidable if a regular tree language is closed under rewriting. This is also part of IsaFoR and loosely related to our work. In the yearly confluence competition[2] confluence checkers for various flavors of term rewriting (like CO3, CoScart, and ConCon for conditional rewriting [18]) compete in different categories. However, at the time of writing none of the other conditional confluence tools supports certification.

2 Preliminaries

We assume familiarity with the basic notions of (conditional) term rewriting [2, 10], but shortly recapitulate terminology and notation that we use in the remainder.

[2] `http://coco.nue.riec.tohoku.ac.jp`

Given two arbitrary binary relations $\to_\alpha$ and $\to_\beta$, we write ${}_\alpha\!\leftarrow$, $\to_\alpha^+$, $\to_\alpha^*$ for the *inverse*, the *transitive closure*, and the *reflexive transitive closure* of $\to_\alpha$, respectively. Moreover, the relations ${}_\alpha^*\!\leftarrow \cdot \to_\beta^*$ and $\to_\beta^* \cdot {}_\alpha^*\!\leftarrow$ are called *meetability* and *joinability*. We say that $\to_\alpha$ and $\to_\beta$ *commute* whenever ${}_\alpha^*\!\leftarrow \cdot \to_\beta^* \subseteq \to_\beta^* \cdot {}_\alpha^*\!\leftarrow$ holds. The same property is called *confluence*, in case α and β coincide. Given a set B we define the set of ancestors with respect to $\to_\alpha$ by $(\to_\alpha)[B] = \{a \mid \exists b \in B.\, a \to_\alpha b\}$.

We use $\mathcal{V}(\cdot)$ to denote the set of variables occurring in a given syntactic object, like a term, a pair of terms, a list of terms, etc. The set of terms $\mathcal{T}(\mathcal{F},\mathcal{V})$ over a given signature of function symbols $\mathcal{F}$ and set of variables $\mathcal{V}$ is defined inductively: $x \in \mathcal{T}(\mathcal{F},\mathcal{V})$ for all variables $x \in \mathcal{V}$, and for every n-ary function symbol $f \in \mathcal{F}$ and terms $t_1,\ldots,t_n \in \mathcal{T}(\mathcal{F},\mathcal{V})$ also $f(t_1,\ldots,t_n) \in \mathcal{T}(\mathcal{F},\mathcal{V})$. A term t is called *ground* if $\mathcal{V}(t) = \varnothing$. The set of ground terms over $\mathcal{F}$ is denoted by $\mathcal{T}(\mathcal{F})$ and the set of ground instances of a term t over a fixed signature is denoted by $\Sigma(t)$. We say that terms s and t *unify*, written $s \sim t$, if $s\sigma = t\sigma$ for some substitution σ. For brevity, we speak about non-reachability, non-meetability, and non-joinability of two terms s and t, when we actually mean that the respective property holds for arbitrary substitution instances $s\sigma$ and $t\tau$. The $\mathtt{tcap}$ function [6] approximates the topmost part of a term, its "cap," that does not change under rewriting (we defer a formal definition until Section 4.1). It is well known that $\mathtt{tcap}(s) \not\sim t$ implies non-reachability of t from s.

For the purposes of this paper a *rewrite rule* (or just *rule*) is a pair of terms, written $\ell \to r$, whose left-hand side is not a variable (meaning that extra variables in right-hand sides are explicitly allowed). A *conditional rewrite rule* is additionally equipped with a list of pairs of terms, written $c = s_1 \approx t_1, \ldots, s_k \approx t_k$, and called its *conditions*. Let c_i denote the first i conditions of c and $c_{i,j}$ the list of conditions $s_i \approx t_i, \ldots, s_j \approx t_j$. A *(conditional) term rewrite system* $\mathcal{R}$ is a set of (conditional) rewrite rules. A CTRS which allows for extra variables in right-hand sides of rules is called a *3-CTRS* (formally, $\mathcal{V}(r) \subseteq \mathcal{V}(\ell, c)$ for all $\ell \to r \Leftarrow c \in \mathcal{R}$). We restrict our attention to *oriented* CTRSs, i.e., where conditions are interpreted as reachability requirements. The rewrite relation induced by an oriented CTRS $\mathcal{R}$ is structured into *levels*. For each level i, a TRS $\mathcal{R}_i$ is defined recursively as follows: $\mathcal{R}_0 = \varnothing$, and $\mathcal{R}_{i+1} = \{\ell\sigma \to r\sigma \mid \ell \to r \Leftarrow c \in \mathcal{R}, \forall s \approx t \in c.\, s\sigma \to_{\mathcal{R}_i}^* t\sigma\}$. For brevity, we write $\to_n$ for the rewrite relation of $\mathcal{R}_n$ whenever $\mathcal{R}$ is clear from the context. Furthermore, we write $\sigma, n \vdash c$, whenever $s\sigma \to_n^* t\sigma$ for all $s \approx t$ in c. By dropping all conditions from a CTRS $\mathcal{R}$ we obtain its *underlying TRS*, denoted $\mathcal{R}_u$. Note that $\to_{\mathcal{R}} \subseteq \to_{\mathcal{R}_u}$.

Two variable-disjoint variants of rules $\ell_1 \to r_1 \Leftarrow c_1$ and $\ell_2 \to r_2 \Leftarrow c_2$ in $\mathcal{R}$ such that $\ell_1|_p \notin \mathcal{V}$ and $\ell_1|_p\mu = \ell_2\mu$ with most general unifier (mgu) μ, constitute a *conditional overlap*. A conditional overlap that does not result from overlapping two variants of the same rule at the root, gives rise to a *conditional critical pair* (CCP) $r_1\mu \approx r_1[r_2]_p\mu \Leftarrow c_1\mu, c_2\mu$. A CCP $s \approx t \Leftarrow c$ is said to be *infeasible* if its conditions cannot be satisfied by any substitution σ. We sometimes use rules, overlaps, critical pairs, etc. without the addendum "conditional."

We consider bottom-up non-deterministic finite tree automata (TA) $\mathcal{A} = \langle \mathcal{F}, Q, Q_f, \Delta \rangle$ where $\mathcal{F}$ is a signature, Q a set of states disjoint from the signature, $Q_f \subseteq Q$ the set of final states, and Δ a set of transitions of the shape $f(q_1,\ldots,q_n) \to q$ with $f \in \mathcal{F}$ and $q_1,\ldots,q_n,q \in Q$ or $q \to p$ with $q,p \in Q$. The language of a TA $\mathcal{A}$ is given by $L(\mathcal{A}) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists q \in Q_f.\, t \to_{\mathcal{A}}^* q\}$. We say that a set of ground terms E is *regular* if there is a TA $\mathcal{A}$ such that $L(\mathcal{A}) = E$. A substitution from variables to states is called a *state substitution*. A TRS $\mathcal{R}$ is called *growing* if for all $\ell \to r \in \mathcal{R}$ the variables in $\mathcal{V}(\ell) \cap \mathcal{V}(r)$ occur at depth at most 1 in ℓ (cf. [8]). Given a TRS $\mathcal{R}$ the *linear growing approximation* [8] is defined as any linear growing TRS obtained from $\mathcal{R}$ by linearizing the left-hand sides, renaming the variables in the right-hand sides that occur at a depth greater than one in the corresponding left-hand side, and finally also linearizing the right-hand sides.

3 An Enhanced Criterion for Level-Confluence of CTRSs

Level-confluence of a CTRS $\mathcal{R}$ is the property that $\mathcal{R}_n$ is confluent for each level n. Clearly, level-confluence implies confluence (take the maximum of the two levels employed in a peak).

The following purely syntactic criterion for level-confluence of (possibly nonterminating) oriented CTRSs with extra variables in right-hand sides was given by Suzuki et al.

▶ **Lemma 1** (Suzuki et al. [16, Corollary 4.7]). *Orthogonal, properly oriented, right-stable, and oriented 3-CTRSs are level-confluent.* ◀

In earlier work [12] we formalized Lemma 1 in Isabelle/HOL and extended it from orthogonal to almost orthogonal CTRSs with infeasible critical pairs. In the following we present a relaxation of *almost orthogonality modulo infeasibility* that allows us to conclude non-meetability from non-joinability when showing infeasibility of conditional overlaps.

▶ **Definition 2** (Almost Orthogonality modulo Infeasibility). A left-linear CTRS $\mathcal{R}$ is *almost orthogonal (modulo infeasibility)* if each overlap between rules $\ell_1 \to r_1 \Leftarrow c_1$ and $\ell_2 \to r_2 \Leftarrow c_2$ with mgu μ at position p either

1. results from overlapping two variants of the same rule at the root, or
2. is trivial (i.e., $p = \epsilon$ and $r_1\mu = r_2\mu$), or
3. is infeasible in the following sense: for arbitrary m and n, whenever levels m and n commute, then it is impossible to satisfy the conditions stemming from the first rule on level m and at the same time the conditions stemming from the second rule on level n. More formally: $\forall m\, n.\, ({}_{m}^{*}\!\leftarrow \cdot \rightarrow_{n}^{*} \subseteq \rightarrow_{n}^{*} \cdot {}_{m}^{*}\!\leftarrow \implies \nexists \sigma.\, m, \sigma \vdash c_1\mu \wedge n, \sigma \vdash c_2\mu)$.

Note that without **2** and **3**, Definition 2 corresponds to plain orthogonality. Also note that by dropping **3**, Definition 2 reduces to the definition of *almost orthogonality* given by Hanus [7]. In our original definition of *almost orthogonality modulo infeasibility* [12], **3** is the stronger requirement that the conditions of the resulting critical pair are infeasible (i.e., $\nexists \sigma\, n.\, n, \sigma \vdash c_1\mu, c_2\mu$). In the following, whenever we talk about *almost orthogonality* we mean Definition 2.

Observe that the level-commutation[3] assumption of the third alternative in Definition 2 allows us to reduce non-meetability to non-joinability. That this is useful in practice is shown by the following example.

▶ **Example 3** (Non-Meetability via tcap). Consider the CTRS consisting of the two rules $\{\mathsf{f}(x) \to \mathsf{a} \Leftarrow x \approx \mathsf{a}, \mathsf{f}(x) \to \mathsf{b} \Leftarrow x \approx \mathsf{b}\}$ which has the critical pair $\mathsf{a} \approx \mathsf{b} \Leftarrow x \approx \mathsf{a}, x \approx \mathsf{b}$. Since $\mathsf{tcap}(\mathsf{cs}(x, x)) = \mathsf{cs}(y, z) \sim \mathsf{cs}(\mathsf{a}, \mathsf{b})$, where cs is an auxiliary function symbol, we cannot conclude infeasibility via non-reachability analysis using tcap. However, $\mathsf{tcap}(\mathsf{a}) = \mathsf{a} \not\sim \mathsf{b} = \mathsf{tcap}(\mathsf{b})$ shows non-joinability of a and b. By **3** this shows non-meetability of a and b and thereby infeasibility of the critical pair.

In general it is beneficial to test for non-meetability via non-joinability of conditions with identical left-hand sides, see also Lemma 21.

Before we can state the main result of this section we have to define two syntactic properties of conditional rewrite rules.

[3] While this is called *shallow confluence* in the literature, we believe that *level-commutation* is a better, since more descriptive, name.

▶ **Definition 4** (Right-Stable, Extended Properly Oriented). A conditional rule $\ell \to r \Leftarrow c$ with k conditions $c = s_1 \approx t_1, \dots, s_k \approx t_k$ is called

1. *right-stable* whenever we have $\mathcal{V}(t_i) \cap \mathcal{V}(\ell, c_{i-1}, s_i) = \varnothing$ and t_i is either a linear constructor term or a ground $\mathcal{R}_u$-normal form, for all $1 \leqslant i \leqslant k$; and
2. *extended properly oriented* when either $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$ or there is some $0 \leqslant m \leqslant k$ such that $\mathcal{V}(s_i) \subseteq \mathcal{V}(\ell, t_1, \dots, t_{i-1})$ for all $1 \leqslant i \leqslant m$ and $\mathcal{V}(r) \cap \mathcal{V}(s_i \approx t_i) \subseteq \mathcal{V}(\ell, t_1, \dots, t_m)$ for all $m < i \leqslant k$.

Observe the following property of extended properly oriented rules of 3-CTRSs

$$\mathcal{V}(r) \subseteq \mathcal{V}(\ell, c_m) \cup (\mathcal{V}(r) \cap \mathcal{V}(c_{m+1,k})) \qquad (\star)$$

which we will use later and which can be shown by induction on $k - m$.

▶ **Theorem 5.** *Almost orthogonal, extended properly oriented, right-stable, and oriented 3-CTRSs are confluent.*

Before proving this statement we need an auxiliary definition, where we adopt the convention that the number of holes of a multihole context is denoted by the corresponding lower-case letter, e.g., c for C, d for D, e for E etc.

▶ **Definition 6.** We say that there is an *extended parallel rewrite step at level* n from s to t, written $s \mathrel{\Vert\!\!\!\twoheadrightarrow}_n t$, whenever we have a multihole context C, and sequences of terms $s_1, \dots, s_c$ and $t_1, \dots, t_c$, such that $s = C[s_1, \dots, s_c]$, $t = C[t_1, \dots, t_c]$, and for all $1 \leqslant i \leqslant k$ we have one of $(s_i, t_i) \in \mathcal{R}_n$ (that is, a root-step at level n) and $s_i \to_{n-1}^* t_i$.

Proof of Theorem 5. Let $\mathcal{R}$ be a CTRS satisfying all required properties. Instead of directly proving the above statement, we prove the commuting diamond property, ${}_m\mathrel{\twoheadleftarrow\!\!\!\Vert} \cdot \mathrel{\Vert\!\!\!\twoheadrightarrow}_n \subseteq \mathrel{\Vert\!\!\!\twoheadrightarrow}_n \cdot {}_m\mathrel{\twoheadleftarrow\!\!\!\Vert}$, for all m and n and a suitable relation $\to_n \subseteq \mathrel{\Vert\!\!\!\twoheadrightarrow}_n \subseteq \to_n^*$ which is called *extended parallel rewriting*. This yields commutation of $\to_m^*$ and $\to_n^*$ for all m and n, and thereby level-confluence, which in turn ensures confluence.

We proceed by complete induction on $m + n$. By induction hypothesis (IH) we may assume the result for all $m' + n' < m + n$. Now consider the peak $t \;{}_m\mathrel{\twoheadleftarrow\!\!\!\Vert}\; s \mathrel{\Vert\!\!\!\twoheadrightarrow}_n u$. If any of m and n equals 0, we are done (since $\mathrel{\Vert\!\!\!\twoheadrightarrow}_0$ is the identity relation). Thus we may assume $m = m' + 1$ and $n = n' + 1$ for some m' and n'. By the definition of extended parallel rewriting, we obtain multihole contexts C and D, and sequences of terms $s_1, \dots, s_c$, $t_1, \dots, t_c$, $u_1, \dots, u_d$, $v_1, \dots, v_d$, such that $s = C[s_1, \dots, s_c]$ and $t = C[t_1, \dots, t_c]$, as well as $s = D[u_1, \dots, u_d]$ and $u = D[v_1, \dots, v_d]$; and $(s_i, t_i) \in \mathcal{R}_m$ or $s_i \to_{m'}^* t_i$ for all $1 \leqslant i \leqslant c$, as well as $(u_i, v_i) \in \mathcal{R}_n$ or $u_i \to_{n'}^* v_i$ for all $1 \leqslant i \leqslant d$.

It is relatively easy to define the greatest lower bound $C \sqcap D$ of two contexts C and D by a recursive function (that simultaneously traverses the two contexts in a top-down manner and replaces subcontexts that differ by a hole) and prove that we obtain a lower semilattice. Now we identify the common part E of C and D, employing the semilattice properties of multihole contexts, that is, $E = C \sqcap D$. Then $C = E[C_1, \dots, C_e]$ and $D = E[D_1, \dots, D_e]$ for some multihole contexts $C_1, \dots, C_e$ and $D_1, \dots, D_e$ such that for each $1 \leqslant i \leqslant e$ we have $C_i = \square$ or $D_i = \square$. This also means that there is a sequence of terms $s'_1, \dots, s'_e$ such that $s = E[s'_1, \dots, s'_e]$ and for all $1 \leqslant i \leqslant e$, we have $s'_i = C_i[s_{k_i}, \dots, s_{k_i + c_i - 1}]$ for some subsequence $s_{k_i}, \dots, s_{k_i + c_i - 1}$ of $s_1, \dots, s_c$ (we denote similar terms for t, u, and v by t'_i, u'_i, and v'_i, respectively). Moreover, note that by construction $s'_i = u'_i$ for all $1 \leqslant i \leqslant e$. Since extended parallel rewriting is closed under multihole contexts, it suffices to show that for each $1 \leqslant i \leqslant e$ there is a term v such that $t'_i \mathrel{\Vert\!\!\!\twoheadrightarrow}_n v \;{}_m\mathrel{\twoheadleftarrow\!\!\!\Vert}\; v'_i$, in order to conclude the proof. We concentrate on the case $C_i = \square$ (the case $D_i = \square$ is completely symmetric). Moreover, note that when we have $s_i' \to_{m'}^* t_i'$, the proof concludes by IH (together with some basic properties of the involved relations), and thus we remain with $(s_i', t_i') \in \mathcal{R}_m$. At this point we distinguish the following cases:

1. ($D_i = \Box$). Also here, the non-root case $u_i' \to_{n'}^* v_i'$ is covered by the IH. Thus, we may restrict to $(u_i', v_i') \in \mathcal{R}_n$, giving rise to a root overlap. Since $\mathcal{R}$ is almost orthogonal, this means that either the resulting conditions are not satisfiable or the resulting terms are the same (in both of these cases we are done), or two variable disjoint variants of the same rule $\ell \to r \Leftarrow c$ with conditions $c = s_1 \approx t_1, \dots, s_j \approx t_j$ were involved, i.e., $u_i' = \ell\sigma_1 = \ell\sigma_2$ for some substitutions σ_1 and σ_2 that both satisfy all conditions in c. Without extra variables in r, this is the end of the story (since then $r\sigma_1 = r\sigma_2$); but we also want to cover the case where $\mathcal{V}(r) \nsubseteq \mathcal{V}(\ell)$, and thus have to reason why this does not cause any trouble. Together with the fact that $\ell \to r \Leftarrow c$ is extended properly oriented we obtain a $0 \leqslant k \leqslant j$ such that (1) $\mathcal{V}(s_i) \subseteq \mathcal{V}(\ell, t_1, \dots, t_{i-1})$ for all $1 \leqslant i \leqslant k$ and (2) $\mathcal{V}(r) \cap \mathcal{V}(s_i \approx t_i) \subseteq \mathcal{V}(\ell, t_1, \dots, t_k)$ for all $k < i \leqslant j$ by Definition 4.2. Then we prove by an inner induction on $i \leqslant j$ that there is a substitution σ such that
   - **a.** $\sigma(x) = \sigma_1(x) = \sigma_2(x)$ for all x in $\mathcal{V}(\ell)$, and
   - **b.** $\sigma_1(x) \mathrel{⇼}_{n'}^* \sigma(x)$ and $\sigma_2(x) \mathrel{⇼}_{m'}^* \sigma(x)$ for all x in $\mathcal{V}(\ell, c_{\min\{i,k\}}) \cup (\mathcal{V}(r) \cap \mathcal{V}(c_{k+1,i}))$.

   In the base case σ_1 satisfies the requirements. So suppose $i > 0$ and assume by IH that both properties hold for $i - 1$ and some substitution σ. If $i > k$ we are done by (2). Otherwise $i \leqslant k$. Now consider the condition $s_i \approx t_i$. By (1) we have $\mathcal{V}(s_i) \subseteq \mathcal{V}(\ell, c_{i-1})$. Using the IH for **1b** we obtain $s_i\sigma_1 \mathrel{⇼}_{n'}^* s_i\sigma$ and $s_i\sigma_2 \mathrel{⇼}_{m'}^* s_i\sigma$. Moreover $s_i\sigma_1 \mathrel{⇼}_{m'}^* t_i\sigma_1$ and $s_i\sigma_2 \mathrel{⇼}_{n'}^* t_i\sigma_2$ since σ_1 and σ_2 satisfy c, and thus by the outer IH we obtain s' such that $t_i\sigma_1 \mathrel{⇼}_{n'}^* s'$ and $t_i\sigma_2 \mathrel{⇼}_{m'}^* s'$. Recall that by right-stability t_i is either a ground $\mathcal{R}_u$-normal form or a linear constructor term. In the former case $t_i\sigma_1 = t_i\sigma_2 = s'$ and hence σ satisfies **1a** and **1b**. In the latter case right-stability allows us to combine the restriction of σ_1 to $\mathcal{V}(t_i)$ and the restriction of σ to $\mathcal{V}(\ell, c_{i-1})$ into a substitution satisfying **1a** and **1b**. This concludes the inner induction. Since $\mathcal{R}$ is a 3-CTRS, using ($\star$) together with **1b** shows $r\sigma_1 \mathrel{⇼}_{n'}^* r\sigma$ and $r\sigma_2 \mathrel{⇼}_{m'}^* r\sigma$. Since $\mathrel{⇼}_{n'}^* \subseteq \mathrel{⇼}_n$ and $\mathrel{⇼}_{m'}^* \subseteq \mathrel{⇼}_m$ we can take $v = r\sigma$ to conclude this case.
2. ($D_i \neq \Box$). Then for some $1 \leqslant k \leqslant d$, we have $(u_j, v_j) \in \mathcal{R}_n$ or $u_j \to_{n'}^* v_j$ for all $k \leqslant j \leqslant k + d_i - 1$, that is, an extended parallel rewrite step of level n from $s_i' = u_i' = D_i[u_{k_i}, \dots, u_{k_i+d_i-1}]$ to $D_i[v_{k_i}, \dots, v_{k_i+d_i-1}] = v_i'$. Since $\mathcal{R}$ is almost orthogonal and, by $D_i \neq \Box$, root overlaps are excluded, the constituent parts of the extended parallel step from s_i' to v_i' take place exclusively inside the substitution of the root-step to the left (which is somewhat obvious – as also stated by Suzuki et al. [16] – but surprisingly hard to formalize, even more so when having to deal with infeasibility). We again close this case by induction on the number of conditions making use of right-stability of $\mathcal{R}$. ◀

Clearly, applicability of Theorem 5 relies on having powerful techniques for proving infeasibility at our disposal. Those are the topic of the next section.

4 Infeasibility

In the context of oriented conditional term rewriting we may employ non-reachability criteria in order to conclude infeasibility of conditions. The two prevalent methods to check for non-reachability use unification and tree automata techniques, respectively. In the following two sections we describe both of these.

4.1 Non-reachability by unification

Probably the fastest available method of checking for non-reachability is to try to unify the tcap of the source term with the target term; which is the de facto standard for approximating dependency graphs for termination proofs [6]. Typical "pen and paper" definitions rely on replacing subterms by "fresh variables" making them somewhat hard to formalize (as already remarked in [17]). Instead of inventing fresh variables out of thin air, the IsaFoR-version of tcap replaces every variable occurrence by the symbol $\square$. The resulting terms behave like ground multihole contexts – we call them *ground contexts* – and they are intended to represent the set of all terms resulting from replacing all "holes" by arbitrary terms. This is made formal by the *substitution instance class* $[\![t]\!]$ of a ground context t: $[\![\square]\!] = \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $[\![f(t_1, \ldots, t_n)]\!] = \{f(s_1, \ldots, s_n) \mid s_i \in [\![t_i]\!]\}$. Note that for variable-disjoint terms s and t, unifiability coincides with $s\sigma = t\tau$ for some substitutions σ and τ. Thus asking whether a term t unifies with a variable-disjoint term represented by the ground context s is equivalent to checking whether $t\sigma \in [\![s]\!]$ for some substitution σ. The latter is called *ground context matching* and shown to be decidable by an efficient algorithm by Thiemann and Sternagel [17]. Thus we can define an efficient executable version of tcap by

$$\mathsf{tcap}_{\mathcal{R}}(t) = \begin{cases} \square & \text{if } t \text{ is a variable} \\ \square & \text{if } t = f(t_1, \ldots, t_n) \text{ and } \ell\sigma \in [\![u]\!] \text{ for some } \sigma \text{ and } \ell \to r \in \mathcal{R} \\ u & \text{otherwise} \end{cases}$$

where $u = f(\mathsf{tcap}_{\mathcal{R}}(t_1), \ldots, \mathsf{tcap}_{\mathcal{R}}(t_n))$.

▶ **Lemma 7** (tcap is sound). *If $s\sigma \to_{\mathcal{R}}^* t$ then $t \in [\![\mathsf{tcap}(s)]\!]$.* ◀

Then checking non-reachability of t from s amounts to deciding whether $\nexists\tau.\, t\tau \in [\![\mathsf{tcap}(s)]\!]$, for which we use the more succinct notation $\mathsf{tcap}(s) \not\sim t$ almost everywhere else in this paper.

While the above definition of tcap and the corresponding soundness lemma were already present in IsaFoR, the following easy extension also allows us to test for non-joinability.

▶ **Lemma 8.** *If $s\sigma \to_{\mathcal{R}}^* \cdot {}_{\mathcal{R}}^*\!\!\leftarrow t\tau$ then $[\![\mathsf{tcap}(s)]\!] \cap [\![\mathsf{tcap}(t)]\!] \neq \varnothing$.*

Proof. We have $s\sigma \to_{\mathcal{R}}^* u$ and $t\tau \to_{\mathcal{R}}^* u$ for some u. By Lemma 7 we have $u \in \mathsf{tcap}(s)$ and $u \in \mathsf{tcap}(t)$. ◀

Fortunately the same techniques that are used to obtain an algorithm for ground context matching can be reused for *ground context unifiability*, i.e., checking $[\![\mathsf{tcap}(s)]\!] \cap [\![\mathsf{tcap}(t)]\!] \neq \varnothing$ (elsewhere in this paper we use the notation $\mathsf{tcap}(s) \not\sim \mathsf{tcap}(t)$).

4.2 Non-reachability by exact tree automata completion

What is generally known as tree automata completion today was introduced by Genet in 1998 [4, 5]. But already in 1996 Jacquemard [8] used a similar concept to show decidability of reachability for linear and growing TRSs. His proof was based on the construction of a tree automaton that accepts the set of ground terms which are normalizable with respect to a given linear and growing TRS $\mathcal{R}$. If we replace the automaton recognizing $\mathcal{R}$-normal forms in Jacquemard's construction by an arbitrary automaton $\mathcal{A}$ we arrive at a tree automaton that accepts the $\mathcal{R}$-ancestors of the language of $\mathcal{A}$.

We need some basic definitions and auxiliary lemmas before we present the construction of this *ancestor automaton* in detail.

▶ **Definition 9** (Ground-Instance Transitions, Δ_t). Let $[t]$ denote a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ where all variable-occurrences have been replaced by a fresh symbol $\Box$. Using such terms as states we define the set Δ_t that contains all transitions which are needed to recognize all ground-instances of a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ in state $[t]$.

$$\Delta_t = \begin{cases} \{f([t_1], \ldots, [t_n]) \to [t]\} \cup \bigcup_{1 \leqslant i \leqslant n} \Delta_{t_i} & \text{if } t = f(t_1, \ldots, t_n) \\ \{f(\Box, \ldots, \Box) \to \Box \mid f \in \mathcal{F}\} & \text{otherwise} \end{cases}$$

Note that if t is not linear this actually gives an overapproximation.

The next lemma holds by definition of Δ_t.

▶ **Lemma 10.** *For any subterm s of any term t if there is a sequence $u \to_{\Delta_t}^{+} [s]$ then u is a ground-instance of s, and vice versa if t is linear.* ◀

We now use Δ_t to define an automaton for the ground-instances of t.

▶ **Definition 11** (Ground-Instance Automaton, $\mathcal{A}_{\Sigma(t)}$). Let Q_t denote the set of states occurring in Δ_t then we call the tree automaton $\mathcal{A}_{\Sigma(t)} = \langle \mathcal{F}, Q_t, \{[t]\}, \Delta_t \rangle$ the *ground-instance automaton for t*.

▶ **Lemma 12.** *The language of $\mathcal{A}_{\Sigma(t)}$ is an overapproximation of the set of ground-instances of t in general and an exact characterization if t is linear.* ◀

Using the concept of ground-instance automaton we are now able to define a tree automaton which accepts all $\mathcal{R}$-ancestors of a given regular set of ground terms using exact tree automata completion (ETAC).

▶ **Definition 13** (Ancestor Automaton, $\mathsf{anc}_{\mathcal{R}}(\mathcal{A})$). Given a tree automaton $\mathcal{A} = \langle \mathcal{F}, Q_{\mathcal{A}}, Q_f, \Delta \rangle$ whose states are all accessible, and a linear and growing TRS $\mathcal{R}$ the construction proceeds as follows.

First we extend the set of transitions of $\mathcal{A}$ in such a way that we can match left-hand sides of rules in $\mathcal{R}$. This yields the set of transitions $\Delta_0 = \Delta \cup \bigcup_{\ell \to r \in \mathcal{R}} \Delta_\ell$. Let $\mathcal{A}_0 = \langle \mathcal{F}, Q, Q_f, \Delta_0 \rangle$ where Q denotes the set of states in Δ_0. We have to ensure (for example by using the disjoint union of states) that for any state q which is used in both Δ and some Δ_ℓ, the terms which can reach it are the same ($\{t \mid t \to_{\Delta}^{+} q\} = \{t \mid t \to_{\Delta_\ell}^{+} q\}$). Then the language does not change, that is, $L(\mathcal{A}_0) = L(\mathcal{A})$.

Finally, we saturate Δ_0 by inference rule (†) in order to extend the language by $\mathcal{R}$-ancestors, that is, if we can reach a state q from an instance of a right-hand side of a rule in $\mathcal{R}$ we add a transition which ensures q is reachable from the corresponding left-hand side.[4]

$$\frac{f(\ell_1, \ldots, \ell_n) \to r \in \mathcal{R} \quad r\theta \to_{\Delta_k}^{*} q}{f(q_1, \ldots, q_n) \to q \in \Delta_{k+1}} \tag{†}$$

Here $\theta\colon \mathcal{V}(r) \to Q$ is a state substitution and $q_i = \ell_i\theta$ if ℓ_i is a variable in r and $q_i = [\ell_i]$ otherwise. Note that this inductive definition possibly adds many new transitions from Δ_k to Δ_{k+1}.

Since $\mathcal{R}$ is finite, the number of states is finite, and we do not introduce new states using (†), this process terminates after finitely many steps resulting in the set of transitions Δ_m. Also note that Δ_k is monotone with respect to k, i.e., $\Delta_k \subseteq \Delta_{k+1}$ for all $k \geqslant 0$. We call $\mathsf{anc}_{\mathcal{R}}(\mathcal{A}) = \langle \mathcal{F}, Q, Q_f, \Delta_m \rangle$ the $\mathcal{R}$-ancestors automaton for $\mathcal{A}$. It is easy to show that $L(\mathcal{A}_0) \subseteq L(\mathsf{anc}_{\mathcal{R}}(\mathcal{A}))$.

[4] This is symmetric to resolving compatibility violations in the tree automata completion by Genet [4, 5].

$$\begin{tikzcd}
s = D[f(s_1,\dots,s_n)] \arrow[r, "*", "\Delta \cup \Delta_{k'}"'] \arrow[d, dotted, "\mathcal{R}"', "*"] & D[f(q_1,\dots,q_n)] \arrow[r, "*", "\Delta \cup \Delta_{k'}"'] & C[f(q_1,\dots,q_n)] \arrow[r, "*", "\rho"'] & C[q'] \arrow[r, "*", "\Delta_{k'+1}"'] & {[t]} \\
D[\ell\tau] \arrow[r, dotted, "*", "\mathcal{R}"'] & D[r\tau] \arrow[r, dotted, "*", "\Delta \cup \Delta_{k'}"'] & C[r\tau] \arrow[r, dotted, "*", "\Delta \cup \Delta_{k'}"'] & C[r\theta] \arrow[u, dotted, "*", "\Delta_{k'}"']
\end{tikzcd}$$

Figure 1 Bypassing ρ to close the induction step.

▶ **Theorem 14.** *Given a tree automaton $\mathcal{A}$ as well as a linear and growing TRS $\mathcal{R}$ the language of $\mathsf{anc}_{\mathcal{R}}(\mathcal{A})$ is exactly the set of $\mathcal{R}$-ancestors of $L(\mathcal{A})$.*

Proof. First we prove that $(\to_{\mathcal{R}}^*)[L(\mathcal{A})] \subseteq L(\mathsf{anc}_{\mathcal{R}}(\mathcal{A}))$. Pick a term $s \in (\to_{\mathcal{R}}^*)[L(\mathcal{A})]$. But that means that there is a rewrite sequence $s \to_{\mathcal{R}}^k t$ of length $k \geqslant 0$ for some $t \in L(\mathcal{A})$. We proceed by induction on k. If $k = 0$ then $s = t$ and hence $s \in L(\mathsf{anc}_{\mathcal{R}}(\mathcal{A}))$. Now assume $k = k' + 1$ for some $k' \geqslant 0$ then there is a rewrite sequence $s = C[f(\ell_1,\dots,\ell_n)\sigma] \to_{\mathcal{R}} C[r\sigma] \to_{\mathcal{R}}^{k'} t$ for some context C, rewrite rule $f(\ell_1,\dots,\ell_n) \to r \in \mathcal{R}$, and substitution σ. By induction hypothesis $C[r\sigma] \in L(\mathsf{anc}_{\mathcal{R}}(\mathcal{A}))$. But that means that there is a state substitution $\theta\colon \mathcal{V}(r) \to Q$, a state $q \in Q$, and a final state $q_f \in Q_f$ such that $C[r\sigma] \to_{\Delta_m}^* C[r\theta] \to_{\Delta_m}^* C[q] \to_{\Delta_m}^* q_f$. From the construction using rule (†) we know that there is a transition $f(q_1,\dots,q_n) \to q \in \Delta_m$ such that $q_i = \ell_i\theta$ if $\ell_i \in \mathcal{V}(r)$ and $q_i = [\ell_i]$ otherwise. If $\ell_i \in \mathcal{V}(r)$ then $\ell_i\sigma \to_{\Delta_m}^+ \ell_i\theta$ and otherwise $\ell_i\sigma \to_{\Delta_m}^+ [\ell_i]$ for all $1 \leqslant i \leqslant n$. Hence in both cases $\ell_i\sigma \to_{\Delta_m}^+ q_i$. But then we can construct the sequence $s = C[f(\ell_1\sigma,\dots,\ell_n\sigma)] \to_{\Delta_m}^* C[f(q_1,\dots,q_n)] \to_{\Delta_m} C[q] \to_{\Delta_m}^* q_f$ and hence $s \in L(\mathsf{anc}_{\mathcal{R}}(\mathcal{A}))$.

For the other direction we prove the following two properties for all sequences $s \to_{\Delta_m}^+ q$:

1. If $q = [t]$ for some subterm of a left-hand side of a rule in $\mathcal{R}$ then $s \in (\to_{\mathcal{R}}^*)[\Sigma(t)]$.
2. If $q \in Q_f$ then $s \in (\to_{\mathcal{R}}^*)[L(\mathcal{A})]$.

The proof for both properties works along the same lines. We sketch the one for the first property here. From the construction using rule (†) we know that $s \to_{\Delta_k}^+ [t]$ for some $k \geqslant 0$. We proceed by induction on k. If $k = 0$ then $s \to_{\Delta_0}^+ [t]$. By construction of $\mathcal{A}_0$ and Lemma 10 we have $s \in \Sigma(t)$ and hence also $s \in (\to_{\mathcal{R}}^*)[\Sigma(t)]$. Now assume that $k = k' + 1$ for some $k' \geqslant 0$. By induction hypothesis (IH$_0$) $s \to_{\Delta_{k'}}^+ [t]$ implies $s \in (\to_{\mathcal{R}}^*)[\Sigma(t)]$ for all terms s and t. Consider the set $\Delta_{k'+1} \setminus \Delta_{k'}$ of transitions which were newly added in $\Delta_{k'+1}$. We continue by a second induction on the size of $\Delta_{k'+1} \setminus \Delta_{k'}$. If it is empty we have a $\Delta_{k'}$-sequence and may simply close the proof with an application of IH$_0$. Otherwise we have some set Δ and transition $\rho\colon f(q_1,\dots,q_n) \to q'$ that was created from some rule $\ell \to r \in \mathcal{R}$ with $\ell = f(\ell_1,\dots,\ell_n)$ and the sequence $r\theta \to_{\Delta'_k}^* q'$ by an application of rule (†) such that $\{\rho\} \uplus \Delta \subseteq \Delta_{k'+1} \setminus \Delta_{k'}$. The second induction hypothesis (IH$_1$) is if $s \to_{\Delta \cup \Delta_{k'}}^+ [t]$ then $s \in (\to_{\mathcal{R}}^*)[\Sigma(t)]$. Let m denote the number of steps that use ρ. We continue by a third induction on m. If $m = 0$ the sequence from s to $[t]$ only used transitions in $\Delta \cup \Delta_{k'}$ and using IH$_1$ we are done. Otherwise there is some $m' \geqslant 0$ such that $m = m' + 1$ and the induction hypothesis (IH$_2$) is that for all terms s, t if $s \to_{\Delta \cup \Delta_{k'}}^+ [t]$ using ρ only m' times then $s \in (\to_{\mathcal{R}}^*)[\Sigma(t)]$. Now we look at the first step using ρ in the sequence, i.e., $s = D[f(s_1,\dots,s_n)] \to_{\Delta \cup \Delta_{k'}}^* C[f(q_1,\dots,q_n)] \to_\rho C[q'] \to_{\Delta_{k'+1}}^* [t]$. Note that from this we get $D[u] \to_{\Delta \cup \Delta_{k'}}^* C[u]$ for all terms u.

Next we define a substitution τ such that $s \to_{\mathcal{R}}^* D[\ell\tau] \to_{\mathcal{R}} D[r\tau] \to_{\Delta \cup \Delta_{k'}}^* C[r\tau] \to_{\Delta \cup \Delta_{k'}}^* C[q']$. This allows us to bypass the ρ-step and so we arrive at a $\Delta_{k'+1}$-sequence from $D[r\tau]$ to $[t]$ containing one less ρ-step as shown in Figure 1. The construction of τ proceeds as follows: We fix $1 \leqslant i \leqslant n$. If ℓ_i is a variable in r define τ_i to be $\{\ell_i \mapsto s_i\}$. Otherwise we know from the definition of inference rule (†) that $q_i = [\ell_i]$ and $s_i \to^{+}_{\Delta \cup \Delta_{k'}} [\ell_i]$. From that we have that $s_i \in (\to^*_{\mathcal{R}})[\Sigma(\ell_i)]$ but that means that there is some substitution τ_i such that $s_i \to^*_{\mathcal{R}} \ell_i\tau_i$. Moreover let $\tau' = \{x \mapsto u_x \mid x \in \mathcal{V}(r) \setminus \mathcal{V}(\ell)\}$ where u_x is an arbitrary but fixed ground term such that $u_x \to^*_{\Delta_0} x\theta$.[5] Now let τ be the disjoint union of $\tau_1, \ldots, \tau_n, \tau'$. This substitution is well-defined because ℓ is linear. By construction of τ we get $s \to^*_{\mathcal{R}} D[\ell\tau]$.

Consider a variable x occurring in r. If x also occurs in ℓ we have $x = \ell_i$ for some unique $1 \leqslant i \leqslant n$ because $\mathcal{R}$ is growing. But then by construction of τ_i we get $x\tau = \ell_i\tau_i = s_i$. Moreover from the definition of q_i in inference rule (†) we have $q_i = \ell_i\theta = x\theta$. But then $x\tau \to^{+}_{\Delta \cup \Delta_{k'}} x\theta$ from $s_i \to^{+}_{\Delta \cup \Delta_{k'}} q_i$. On the other hand, if x does not occur in ℓ then $x\tau = x\tau'$ and $x\tau' \to^*_{\Delta_0} x\theta$ by construction of τ'. So in both cases $r\tau \to^*_{\Delta \cup \Delta_{k'}} r\theta$. Together with $r\theta \to^*_{\Delta_{k'}} q'$ and $C[q'] \to^*_{\Delta_{k'+1}} [t]$ we may construct the sequence $D[r\tau] \to^{+}_{\Delta_{k'+1}} q_f$ which uses ρ only m' times. Using IH$_2$ we arrive at $D[r\tau] \in (\to^*_{\mathcal{R}})[\Sigma(t)]$. Together with $s \to^*_{\mathcal{R}} D[\ell\tau] \to^*_{\mathcal{R}} D[r\tau]$ this means that $s \in (\to^*_{\mathcal{R}})[\Sigma(t)]$ and we are done. ◀

▶ **Lemma 15** (Non-Reachability via anc). *Let $\mathcal{R}$ be a linear and growing TRS over signature $\mathcal{F}$. We may conclude non-reachability of t from s if the following language is empty:*

$$L(\mathcal{A}_{\Sigma(s)} \cap \mathit{anc}_{\mathcal{R}}(\mathcal{A}_{\Sigma(t)})) \qquad ◀$$

▶ **Example 16** (Infeasibility via anc). Consider the CTRS $\mathcal{R} = \{\mathsf{f}(\mathsf{a}, x) \to \mathsf{a}, \mathsf{f}(\mathsf{b}, x) \to \mathsf{b}, \mathsf{g}(\mathsf{a}, x) \to \mathsf{c} \Leftarrow \mathsf{f}(x, \mathsf{a}) \approx \mathsf{a}, \mathsf{g}(x, \mathsf{a}) \to \mathsf{d} \Leftarrow \mathsf{f}(x, \mathsf{b}) \approx \mathsf{b}, \mathsf{c} \to \mathsf{c}\}$. It has two critical pairs $\mathsf{c} \approx \mathsf{d} \Leftarrow \mathsf{f}(\mathsf{a}, \mathsf{b}) \approx \mathsf{b}, \mathsf{f}(\mathsf{a}, \mathsf{a}) = \mathsf{a}$ and the symmetric one. Since $\mathsf{tcap}(\mathsf{f}(\mathsf{a}, \mathsf{b})) = x \sim \mathsf{b}$ and $\mathsf{tcap}(\mathsf{f}(\mathsf{a}, \mathsf{a})) = x \sim \mathsf{a}$ unification is not sufficient to show infeasibility of these critical pairs. On the other hand, since the underlying TRS $\mathcal{R}_u$ is linear and growing, we may construct the tree automata $\mathcal{A}_{\Sigma(\mathsf{f}(\mathsf{a},\mathsf{b}))}$ and $\mathsf{anc}_{\mathcal{R}_u}(\mathcal{A}_{\Sigma(\mathsf{b})})$. Because the language of the intersection automaton is empty we have shown infeasibility of the condition $\mathsf{f}(\mathsf{a}, \mathsf{b}) \approx \mathsf{b}$ by Lemma 15. So both critical pairs are infeasible.

Moreover, as shown in the following example, anc may be employed to show infeasibility of conditions of a conditional rewrite rule, directly. Such rules can never be used to rewrite and so might as well be removed from a CTRS.

▶ **Example 17** (Infeasible Rules via anc). Consider the CTRS $\mathcal{R} = \{\mathsf{h}(x) \to \mathsf{a}, \mathsf{g}(x) \to x, \mathsf{g}(x) \to \mathsf{a} \Leftarrow \mathsf{h}(x) \approx \mathsf{b}, \mathsf{c} \to \mathsf{c}\}$. The condition of the third rule is infeasible because $\mathsf{h}(x)$ only rewrites to a and not to b. This cannot be shown by unification because $\mathsf{tcap}(\mathsf{h}(x)) = y \sim \mathsf{b} = \mathsf{tcap}(\mathsf{b})$. Fortunately the underlying TRS $\mathcal{R}_u$ is linear and growing and hence we can construct the tree automata $\mathcal{A}_{\Sigma(\mathsf{h}(x))}$ and $\mathsf{anc}_{\mathcal{R}_u}(\mathcal{A}_{\Sigma(\mathsf{b})})$. The language of the intersection automaton is empty and we have shown infeasibility of the condition $\mathsf{h}(x) \approx \mathsf{b}$ by Lemma 15.

Remember that in our setting the right-hand sides of conditions are always linear terms. Hence it is beneficial to start with the ground-instance automaton for the right-hand side of a condition (which in this case is exact) and compute the set of ancestors rather than taking the possibly non-linear left-hand side of a condition, overapproximating the ground-instances and only then computing the descendants of this set.

4.3 Certification

In this section we give an overview of all techniques that are newly supported by our certifier CeTA and what kind of information it requires from a certificate in CPF [14] (short

[5] Since all states in $\mathcal{A}_0$ are accessible we can always find such a term u_x.

for *certification problem format*). Before we come to the special infeasibility condition of Definition 2, we handle the common case where, given a list of conditions c, we are interested in proving $\sigma, n \nvdash c$ for every substitution σ and level n.

▶ **Lemma 18** (Infeasibility Certificates). *Given $(\mathcal{R}, c)$ consisting of a CTRS $\mathcal{R}$ and a list of conditions $c = s_1 \approx t_1, \ldots, s_k \approx t_k$, infeasibility of c with respect to $\mathcal{R}$ can be certified in one of the following ways:*

1. *Provide two terms s and t with $s \approx t \in c$, and a non-reachability certificate for $(\mathcal{R}_u, s, t)$.*
2. *Provide a function symbol cs of arity n (called a* compound symbol*) together with a non-reachability certificate for $(\mathcal{R}_u, \mathsf{cs}(s_1, \ldots, s_k), \mathsf{cs}(t_1, \ldots, t_k))$.*
3. *For an arbitrary subset c' of c, provide an infeasibility certificate for $(\mathcal{R}, c')$.*
4. *Provide three terms s, t, and u such that $s \approx u$ and $t \approx u$ are equations in c together with a non-joinability certificate for $(\mathcal{R}_u, s, t)$.*

Proof. Note that **3** is obvious and **1** only exists for tool-author convenience but is subsumed by the combination of **2** and **3**. Moreover, **2** follows from the fact that $\mathsf{cs}(s_1, \ldots, s_k)\sigma \not\to^*_{\mathcal{R}_u} \mathsf{cs}(t_1, \ldots, t_k)\tau$ for all σ and τ, implies the existence of at least one $1 \leqslant i \leqslant k$ such that $s_i\sigma \not\to^*_{\mathcal{R}_u} t_i\tau$ for all σ and τ. Finally, for **4**, whenever $s\sigma$ and $t\tau$ are not joinable for arbitrary σ and τ, the existence of μ and n such that $\mu, n \vdash s \approx u, t \approx u$ is impossible. ◀

Note that in **2** we check for non-reachability between left-hand sides and their corresponding right-hand sides, while in **4** we check for non-joinability between two left-hand sides. Thus, while in general non-joinability is more difficult to show than non-reachability, **4** is not directly subsumed by **2**.

▶ **Lemma 19** (Non-Reachability Certificates). *Given $(\mathcal{R}, s, t)$ consisting of a TRS $\mathcal{R}$ and two terms s and t, $\mathcal{R}$-non-reachability of t from s can be certified in one of the following ways:*

1. *Indicate that $\mathsf{tcap}(s)$ does not unify with t.*
2. *Provide a TRS $\mathcal{R}'$ such that for each $\ell \to r \in \mathcal{R}$ there is $\ell' \to r' \in \mathcal{R}'$ and a substitution σ with $\ell = \ell'\sigma$ and $r = r'\sigma$, together with a non-reachability certificate for $(\mathcal{R}', s, t)$.*
3. *Provide a non-reachability certificate for $(\mathcal{R}^{-1}, t, s)$.*
4. *Make sure that $\mathcal{R}$ is linear and growing and provide a finite signature $\mathcal{F}$ and two constants a and $\square$ such that $\mathsf{a} \in \mathcal{F}$ but $\square \notin \mathcal{F}$, together with a tree automaton $\mathcal{A}$ that is an overapproximation of $\mathsf{anc}_{\mathcal{R}}(\mathcal{A}_{\Sigma(t)})$ and satisfies $L(\mathcal{A}_{\Sigma(s)} \cap \mathcal{A}) = \varnothing$.*

Proof. If $\mathsf{tcap}_{\mathcal{R}}(s) \not\sim t$, then **1** holds by Lemma 7. Further note that $\to_{\mathcal{R}} \subseteq \to_{\mathcal{R}'}$ and thus **2** is immediate. Moreover, **3** is obvious, leaving us with **4**. From a certification perspective this is the most interesting case. To begin with, there are two reasons why we do not want to repeat the full construction of anc inside CeTA. Firstly, we would unnecessarily repeat an operation with at least exponential worst-case complexity. Secondly, a fully-verified executable algorithm is not even part of our formalization, instead we heavily rely on inductive definitions.[6] In CeTA we check that $\mathcal{A}$ is an overapproximation of $\mathsf{anc}_{\mathcal{R}}(\mathcal{A}_{\Sigma(t)})$ as follows: firstly, we ensure that $\mathcal{A}$ does not contain epsilon transitions, that $[t]$ is included in the final states of $\mathcal{A}$, and that Δ_t as well as the matching rules with respect to the signature $\mathcal{F}$ are part of the transitions of $\mathcal{A}$; secondly, we check that $\mathcal{A}$ is closed with respect to inference rule (†). Since $\mathcal{A}_{\Sigma(s)}$ is an overapproximation of $\Sigma(s)$ and by the required conditions together with

[6] While turning the existing inductive definitions into executable recursive functions would definitely be possible, we stress that this is not necessary.

Theorem 14, $L(\mathcal{A})$ overapproximates $[\rightarrow^*_{\mathcal{R}}](\Sigma(t))$, we can conclude $\Sigma(s) \cap [\rightarrow^*_{\mathcal{R}}](\Sigma(t)) = \varnothing$. Thus there are no *ground* substitutions σ and τ such that $s\sigma, t\tau \in \mathcal{T}(\mathcal{F})$ and $s\sigma \rightarrow^*_{\mathcal{R}} t\tau$. In order to conclude that the same holds true for *arbitrary* substitutions (not necessarily restricted to $\mathcal{F}$), we rely on an earlier result [13] that implies that whenever $s\sigma \rightarrow^*_{\mathcal{R}} t\tau$ for arbitrary σ and τ and $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ then there are σ' and τ' such that $s\sigma', t\tau' \in \mathcal{T}(\mathcal{F})$ and $s\sigma' \rightarrow^*_{\mathcal{R}} t\tau'$. ◀

Note that **2** allows us to certify the linear growing approximation of a TRS without actually having to formalize it in Isabelle/HOL. More specifically, whenever $\mathcal{R}'$ is the result of applying the linear growing approximation to $\mathcal{R}$, then the corresponding certificate will pass **2** and $\mathcal{R}'$ will be linear and growing in the check for **4**; otherwise **4** will fail.

▶ **Lemma 20** (Non-Joinability Certificates). *Given $(\mathcal{R}, s, t)$ consisting of a TRS $\mathcal{R}$ and two term s and t, $\mathcal{R}$-non-joinability of s and t can be certified in one of the following ways.*

1. *Indicate that $\mathsf{tcap}(s)$ does not unify with $\mathsf{tcap}(t)$.*
2. *If at least one of the terms, say t, is a ground $\mathcal{R}$-normal form provide a non-reachability certificate for $(\mathcal{R}, s, t)$.*

Proof. We prove **1** by Lemma 8 and **2** by Lemma 7 since non-joinability reduces to non-reachability when one of the terms is an $\mathcal{R}$-normal form. ◀

▶ **Lemma 21** (Ao-Infeasibility Certificates). *Given $(\mathcal{R}, c_1, c_2)$ consisting of a CTRS $\mathcal{R}$ fulfilling all syntactic requirements of Theorem 5 and two lists of conditions c_1 and c_2, infeasibility with respect to almost orthogonality can be certified in one of the following ways:*

1. *Provide an infeasibility certificate for $(\mathcal{R}, c)$ where c is the concatenation of c_1 and c_2.*
2. *Provide three terms s, t and u such that $s \approx t$ is an equation in c_1 and $s \approx u$ an equation in c_2, together with a non-joinability certificate for $(\mathcal{R}_u, t, u)$.*

Proof. While **1** follows from Lemma 18, in **2** we make use of the level-commutation assumption of Definition 2 to deduce non-meetability of t and u from non-joinability of t and u. ◀

4.4 Comparison

For our main use case, Theorem 5, we are restricted to left-linear CTRSs (via almost orthogonality) and linear right-hand sides of conditions (via right-stability). The latter also holds for right-hand sides that are combined by a compound symbol (again by right-stability). We show that in this setting anc subsumes tcap (at least in theory and for the forward direction).

▶ **Lemma 22.** *Let $\mathcal{R}$ be a left-linear CTRS and t a linear term. If tcap can show non-reachability of t from s, then so can anc.*

Proof. We proof the contrapositive and assume that anc cannot show non-reachability. Moreover, let $\mathcal{R}'$ denote the result of applying the linear growing approximation to $\mathcal{R}_u$. Then there is some term u such that $u \in L(\mathcal{A}_{\Sigma(s)})$ and $u \in L(\mathsf{anc}_{\mathcal{R}'}(\mathcal{A}_{\Sigma(t)}))$. Since t is linear and $\mathcal{R}'$ is linear and growing the latter implies that $u \in (\rightarrow^*_{\mathcal{R}'})[\Sigma(t)]$ and thus $u \rightarrow^*_{\mathcal{R}'} t\tau$ for some substitution τ. By Lemma 7, this means that $t\tau \in [\![\mathsf{tcap}_{\mathcal{R}'}(u)]\!]$. Since $u \in L(\mathcal{A}_{\Sigma(s)})$, it is clearly the case that $u \in \Sigma(\mathsf{ren}(s))$ and thus $u = \mathsf{ren}(s)\sigma$ for some substitution σ, where ren denotes an arbitrary linearization of s. Moreover $[\![\mathsf{tcap}_{\mathcal{R}'}(u)]\!] \subseteq [\![\mathsf{tcap}_{\mathcal{R}'}(\mathsf{ren}(s))]\!] = [\![\mathsf{tcap}_{\mathcal{R}'}(s)]\!]$. Together with $t\tau \in [\![\mathsf{tcap}_{\mathcal{R}'}(u)]\!]$ from above, we obtain $t\tau \in [\![\mathsf{tcap}_{\mathcal{R}'}(s)]\!]$. However, tcap does only consider the left-hand sides of rules, which are the same in $\mathcal{R}'$ and $\mathcal{R}_u$, thus also $t\tau \in [\![\mathsf{tcap}_{\mathcal{R}_u}(s)]\!]$ which implies $\mathsf{tcap}_{\mathcal{R}_u}(s) \sim t$. ◀

Table 1 82 right-stable, extended properly oriented, and oriented 3-CTRSs.

	ConCon			
	uncertified	certified 2	certified 2+3	certified+
confluent	47	23	32	35
non-confluent	15	-	-	-
don't know	20	59	50	47

If we also consider the reverse direction, that is, checking if $t \to^*_{\mathcal{R}_u^{-1}} s$ for some condition $s \approx t$ in Theorem 5, then tcap may well succeed where anc fails, as shown by the next example.

▶ **Example 23** (anc vs. tcap)**.** The oriented 3-CTRS $\mathcal{R} = \{\mathsf{g}(x) \to \mathsf{f}(x,x), \mathsf{g}(x) \to \mathsf{g}(x) \Leftarrow \mathsf{g}(x) \approx \mathsf{f}(\mathsf{a},\mathsf{b})\}$ is right-stable and extended properly oriented. It has two symmetric CCPs of the form $\mathsf{f}(x,x) \approx \mathsf{g}(x) \Leftarrow \mathsf{g}(x) \approx \mathsf{f}(\mathsf{a},\mathsf{b})$. The underlying TRS $\mathcal{R}_u$ is not linear and growing, so if we want to apply anc we have to apply the linear growing approximation, resulting in $\mathcal{R}' = \{\mathsf{g}(x) \to \mathsf{f}(x,y), \mathsf{g}(x) \to \mathsf{g}(x)\}$. But then anc is not able to show infeasibility since the language of $\mathcal{A}_{\Sigma(\mathsf{g}(x))} \cap \mathsf{anc}_{\mathcal{R}'}(\mathcal{A}_{\Sigma(\mathsf{f}(\mathsf{a},\mathsf{b}))})$ is not empty and also for the reverse direction $\mathcal{A}_{\Sigma(\mathsf{f}(\mathsf{a},\mathsf{b}))} \cap \mathsf{anc}_{\mathcal{R}'^{-1}}(\mathcal{A}_{\Sigma(\mathsf{g}(x))})$ we get a non-empty language. On the other hand using the reversed underlying system $\mathcal{R}_u^{-1} = \{\mathsf{f}(x,x) \to \mathsf{g}(x), \mathsf{g}(x) \to \mathsf{g}(x)\}$ we have that $\mathsf{tcap}_{\mathcal{R}_u^{-1}}(\mathsf{f}(\mathsf{a},\mathsf{b})) = \mathsf{f}(\mathsf{a},\mathsf{b}) \not\sim \mathsf{g}(x)$. So in this case tcap succeeds where anc fails.

5 Conclusion and Future Work

We have not only produced several thousand (∼6600) lines of proof documents, but also refined and extended an earlier result which allows us to certify confluence proofs for a larger class of CTRSs. Moreover, a new method to check for non-reachability between terms has been added, which (at least theoretically) further expands this class. The certifier CeTA has been updated to handle all new certificates and the confluence tool ConCon has been extended to use the new results and generate certifiable output for them.

Our formalization exposed an error in the exact tree automata completion procedure implemented in ConCon which is now corrected.

Experiments We shortly examine ConCon's ability to provide certifiable proofs before and after the implementation of the presented results. Our testbed comprises 82 right-stable, extended properly oriented, and oriented 3-CTRSs taken from the confluence problems database (Cops).[7] Note that only 52 of these 82 systems have at least one CCP and hence are amenable to the improved infeasibility methods.

All in all ConCon implements three criteria for checking confluence of oriented CTRSs.

1. Strongly deterministic, quasi-decreasing 3-CTRSs are confluent if all CPs are joinable [1].
2. Theorem 5 from above.
3. A deterministic 3-CTRS is confluent if its unraveling is left-linear and confluent [20].

These are accompanied by some infeasibility-methods (including the ones presented). So far criteria **2** and **3** have been formalized.

[7] http://cops.uibk.ac.at

In Table 1 we summarize our findings. The column labeled *uncertified* contains the results of unleashing ConCon using all (possibly not yet certifiable) criteria for confluence and infeasibility it implements. The next two columns labeled *certified 2* and *certified 2+3*, respectively, show the numbers when only using certifiable methods already present in ConCon before the modification. In *certified 2* we only used the syntactic method from Suzuki et al. Column *certified 2+3* combines the latter with method **3** employing unravelings. Finally, the column labeled *certified+* gives the results for the current version of ConCon including the newly implemented certifiable methods. One interesting point is that method **2** completely subsumes **3** in *certified+*. What we can see from the table is the following: We were able to increase the applicability of **2** by 12 systems. But if we also take into account method **3** we only gain 3 certified proofs. All together we can certify 35 out of 47 confluence proofs. So far we have not worked on formalizing the used non-confluence criteria, hence none of the non-confluence proofs can be certified.

Concerning Examples 16 and 17 it is interesting to note that criterion **1** does not apply, because both systems are non-terminating and also criterion **3** does not apply, because the unraveled system is non-confluent in both cases. So **2** is the only of the three methods that can handle both examples.

Finally, we believe that the small gain of only 3 certified proofs in total, has to be taken with a grain of salt. On the one hand, since the number of CTRSs in Cops is rather low. On the other hand, and more importantly, since the class of CTRSs to which Theorem 5 potentially applies, closely corresponds to what is actually allowed in functional and logic programs.

Future Work For the moment we have restricted our attention to confluence of CTRSs in general and Theorem 5 in particular. However, we provide general techniques for the certification of non-reachability and non-joinability; those should be applicable also in other areas like certification of dependency graph approximations for termination of TRSs, non-confluence of TRSs, and more hypothetically the correctness of protocols.

As for the applicability of our method to the certification of dependency graph approximations, we conducted some preliminary experiments. Here, a *potential edge* consists of two pairs of terms (s, t) and (u, v) where the goal is to prove non-reachability of u from t (since then it does not turn into an actual edge, which might lead to an easier termination proof). Since at the time of writing the only certifiable way of handling dependency graphs in current termination tools is a tcap-based estimation, we started with all the 542428 potential edges obtained from the *termination problem database*[8] that cannot already be handled by tcap. Of those, 129599 are trivially shown to be actual edges via unification (i.e., no rewrite steps are necessary). Out of the remaining 412829 potential edges, we were able to show non-reachability for 10217/24291/43364 when using a timeout of 1/3/10 seconds per edge.

Another direction of future work will be to formalize criterion **1** and also extend our formalization to non-confluence proofs.

Acknowledgments. We thank Bertram Felgenhauer for producing at least 1.5 of our examples. We also thank the anonymous reviewers for their constructive and helpful comments. Further thanks goes to the Austrian Science Fund (FWF project P27502) for supporting our work. Note that the authors are given in alphabetical order.

[8] `http://termination-portal.org/wiki/TPDB`

References

1 Jürgen Avenhaus and Carlos Loría-Sáenz. On conditional rewrite systems with extra variables and deterministic logic programs. In *Proceedings of the 5th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 822 of *Lecture Notes in Computer Science*, pages 215–229. Springer, 1994. 10.1007/3-540-58216-9_40.

2 Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

3 Bertram Felgenhauer and René Thiemann. Reachability analysis with state-compatible automata. In *Proceedings of the 8th International Conference on Language and Automata Theory and Applications*, volume 8370 of *Lecture Notes in Computer Science*, pages 347–359. Springer, 2014. 10.1007/978-3-319-04921-2_28.

4 Thomas Genet. Decidable approximations of sets of descendants and sets of normal forms. In *Proceedings of the 9th International Conference on Rewriting Techniques and Applications*, volume 1379 of *Lecture Notes in Computer Science*, pages 151–165. Springer, 1998. 10.1007/BFb0052368.

5 Thomas Genet and Valerié Viet Triem Tong. Reachability analysis of term rewriting systems with timbuk. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, volume 2250 of *Lecture Notes in Computer Science*, pages 695–706. Springer, 2001. 10.1007/3-540-45653-8_48.

6 Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp. Proving and disproving termination of higher-order functions. In *Proceedings of the 5th International Workshop on Frontiers of Combining Systems*, volume 3717 of *Lecture Notes in Computer Science*, pages 216–231. Springer, 2005. 10.1007/11559306_12.

7 Michael Hanus. On extra variables in (equational) logic programming. In *Proceedings of the 12th International Conference on Logic Programming*, pages 665–679. MIT Press, 1995.

8 Florent Jacquemard. Decidable approximations of term rewriting systems. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, volume 1103 of *Lecture Notes in Computer Science*, pages 362–376. Springer, 1996. 10.1007/3-540-61464-8_65.

9 Tobias Nipkow, Lawrence Charles Paulson, and Makarius Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002. 10.1007/3-540-45949-9.

10 Enno Ohlebusch. *Advanced Topics in Term Rewriting*. Springer, 2002.

11 Christian Sternagel and Aart Middeldorp. Root-labeling. In *Proceedings of the 19th International Conference on Rewriting Techniques and Applications*, volume 5117 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 2008. 10.1007/978-3-540-70590-1_23.

12 Christian Sternagel and Thomas Sternagel. Level-confluence of 3-CTRSs in Isabelle/HOL. In Tiwari and Aoto [18]. `http://www.csl.sri.com/users/tiwari/iwc2015/iwc2015.pdf`.

13 Christian Sternagel and René Thiemann. Signature extensions preserve termination - an alternative proof via dependency pairs. In *Proceedings of the 19h EACSL Annual Conference on Computer Science Logic*, volume 6247 of *Lecture Notes in Computer Science*, pages 514–528. Springer, 2010. 10.1007/978-3-642-15205-4_39.

14 Christian Sternagel and René Thiemann. The Certification Problem Format. In *Proceedings of the 11th Workshop on User Interfaces for Theorem Provers*, pages 61–72, 2014. 10.4204/EPTCS.167.8.

15 Thomas Sternagel and Aart Middeldorp. Conditional confluence (system description). In *Proceedings of the Joint 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 456–465. Springer, 2014. 10.1007/978-3-319-08918-8_31.

16 Taro Suzuki, Aart Middeldorp, and Tetsuo Ida. Level-confluence of conditional rewrite systems with extra variables in right-hand sides. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications*, volume 914 of *Lecture Notes in Computer Science*, pages 179–193. Springer, 1995. 10.1007/3-540-59200-8_56.

17 René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2009. 10.1007/978-3-642-03359-9_31.

18 Ashish Tiwari and Takahito Aoto, editors. *Proceedings of the 4th International Workshop on Confluence*, 2015. `http://www.csl.sri.com/users/tiwari/iwc2015/iwc2015.pdf`.

19 Makarius Wenzel. System description: Isabelle/jEdit in 2014. In *Proceedings of the 11th Workshop on User Interfaces for Theorem Provers*, pages 84–94, 2014. 10.4204/EPTCS.167.10.

20 Sarah Winkler and René Thiemann. Formalizing soundness and completeness of unravelings. In *Proceedings of the 10th International Workshop on Frontiers of Combining Systems*, volume 9322 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 2015. 10.1007/978-3-319-24246-0_15.

Category Theory in Coq 8.5

Amin Timany[1] **and Bart Jacobs**[2]

1 **iMinds-DistriNet – KU Leuven, Dept. of Computer Science, Heverlee, Belgium**
amin.timany@cs.kuleuven.be

2 **iMinds-DistriNet – KU Leuven, Dept. of Computer Science, Heverlee, Belgium**
bart.jacobs@cs.kuleuven.be

Abstract

We report on our experience implementing category theory in Coq 8.5. Our work formalizes most of basic category theory, including concepts not covered by existing formalizations, in a library that is fit to be used as a general-purpose category-theoretical foundation.

Our development particularly takes advantage of two features new to Coq 8.5: primitive projections for records and universe polymorphism. Primitive projections allow for well-behaved dualities while universe polymorphism provides a relative notion of largeness and smallness. The latter is one of the main contributions of this paper. It pushes the limits of the new universe polymorphism and constraint inference algorithm of Coq 8.5.

In this paper we present in detail smallness and largeness in categories and the foundation they are built on top of. We furthermore explain how we have used the universe polymorphism of Coq 8.5 to represent smallness and largeness arguments by simply ignoring them and entrusting them to the universe inference algorithm of Coq 8.5. We also briefly discuss our experience throughout this implementation, discuss concepts formalized in this development and give a comparison with a few other developments of similar extent.

1998 ACM Subject Classification I.2.3 Deduction and Theorem Proving, F.4.1 Mathematical Logic

Keywords and phrases Category Theory, Coq 8.5, Universe Polymorphism, Homotopy Type Theory

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.30

1 Introduction

A category [11, 2] consists of a collection of objects and for each pair of objects A and B a collection of morphisms (aka arrows or homomorphisms) from A to B. Moreover, for each object A we have a distinguished morphism $id_A : A \to A$. Morphisms are composable, i.e., given two morphisms $f : A \to B$ and $g : B \to C$, we can compose them to form: $g \circ f : A \to C$. Composition must satisfy the following additional conditions: $\forall f : A \to B.\ f \circ id_A = f = id_B \circ f$ and $\forall f, g, h.\ (h \circ g) \circ f = h \circ (g \circ f)$.

The notion of a category can be seen as a generalization of sets. In fact sets as objects together with functions as morphisms form the important category **Set**. On the other hand, it can be seen as a generalization of the mathematical concept of a preorder. In this regard, a category can be thought of as a preorder where objects form the elements of the preorder and morphisms from A to B can be thought of as "witnesses" of the fact that $A \preceq B$. Thus, identity morphisms are witnesses of reflexivity whereas composition of morphisms forms witnesses for transitivity and the additional axioms simply spell out coherence conditions for witnesses. Put concisely, categories are preorders where the quality and nature of the relation holding between two elements is important. In this light, categories are to preorders what intuitionistic logic is to classical logic. A combination of these two interpretations of categories can provide an essential and useful intuition for understanding most, if not all, of the theory.

This generality and abstractness is what led some mathematicians to call this mathematical theory "general abstract nonsense" in its early days. However category theory, starting from this simple yet vastly abstract and general definition, encompasses most mathematical concepts and has found applications not only in mathematics but also in other disciplines, e.g, computer science.

In computer science it has been used extensively, especially in the study of semantics of programming languages [15], in particular constructing the first (non-trivial) model of the untyped lambda calculus by Dana Scott [17], type systems [10], and program verification [5, 3, 4].

Given the applications of category theory and its fundamentality on the one hand and the arising trend of formalizing mathematics in proof assistants on the other, it is natural to have category theory formalized in one; in particular, a formalization that is practically useful as a category-theoretical foundation for other works. This paper is a report of our experience developing such a library. There already exist a relatively large number of formalizations of category theory in proof assistants [14, 16, 9, 1, 7]. However, most of these implementations are not general purpose and rather focus on parts of the theory which are relevant to the specific application of the authors. See the bibliography of Gross et al. [8] for an extensive list of such developments.

Features of Coq 8.5 used: η for records and universe polymorphism

This development makes use of two features new to Coq 8.5. Namely, primitive projection for records (i.e., the η rule for records) and universe polymorphism.

Following Gross et al. [7], we use primitive projections for records which allow for well behaved-dualities in category theory. The dual (aka opposite) of a category $\mathcal{C}$ is a category $\mathcal{C}^{op}$ which has the same objects as $\mathcal{C}$ where the collection of morphisms from A to B is swapped with that from B to A. Drawing intuition from the similarity of categories and preorders, the opposite of a category (seen as a preorder) is simply a category where the order of objects is reversed. Use of duality arguments in proofs and definitions in category theory are plentiful, e.g., sums and products, limits and co-limits, etc. One particular property of duality is that it is involutive. That is, for any category $\mathcal{C}$, $(\mathcal{C}^{op})^{op} = \mathcal{C}$. The primitive projection for records simply states that two instances of a record type are definitionally equal if and only if their projections are. In terms of categories, two categories are definitionally equal if and only if their object collections are, morphism collections are and so forth. This means that we get that the equality $(\mathcal{C}^{op})^{op} = \mathcal{C}$ is definitional. Similar results hold for the duality and composition of functors, for natural transformations, etc. That is we get definitional equalities such as $(\mathcal{F}^{op})^{op} = \mathcal{F}$, $(\mathcal{N}^{op})^{op} = \mathcal{N}$ and $(\mathcal{F} \circ \mathcal{G})^{op} = \mathcal{F}^{op} \circ \mathcal{G}^{op}$ where $\mathcal{F}$ and $\mathcal{G}$ are functors and $\mathcal{N}$ is a natural transformation.

To achieve well behaved dualities, in addition to primitive projections one needs to slightly adjust the definition of a category itself. More precisely, the definition of the category must carry a symmetric form of associativity of composition. The reason being the fact that for the dual category we can simply swap the proof of associativity with its symmetric form and thus after taking the opposite twice get back the proof we started with.

In this development we have used universe polymorphism, a feature new to Coq 8.5, to represent relative smallness/largeness. In short, universe polymorphism allows for a definition to be polymorphic in its universe variables. This allows us, for instance, to construct the category of (relatively small) categories directly. That is, the category constructed is at a universe level (again polymorphic) while its objects are categories at a lower universe level. We will elaborate the use of universe polymorphism to represent relative largeness and smallness below in Section 2.

Contributions

The main contributions of this development are its extent of coverage of basic concepts in category theory and its use of the universe polymorphism of Coq 8.5 and its universe inference algorithm to represent relative smallness/largeness. The latter, as explained below, allows us to represent smallness and largeness using universe levels by simply forgetting about them and letting Coq's universe inference algorithm take care of smallness and largeness requirements as necessary.

The structure of the rest of this paper

The rest of this paper is organized as follows. Section 2 gives an explanation of smallness and largeness in category theory based on the foundation used. This is followed by a detailed explanation of our use of the new universe polymorphism and universe constraint inference algorithm of Coq 8.5 to represent relative smallness/largeness of categories. There, we also give a short comparison of the way other developments represent (relatively) large concepts.

In Section 3, we give a high-level explanation of the concepts formalized and some notable features in this work. We furthermore provide a comparison of our work with a number of other works of similar extent. We also briefly discuss the axioms that we have used throughout this development.

Section 4 describes the work that we have done or plan to do which is based on the current work as category-theoretical foundation. Finally, in Section 5 we conclude with a short summary of the paper.

Development source code. The repository of our development can be found in GitHub [21].

2 Universes, Smallness and Largeness

A category is usually called small if its objects and morphisms form sets and large otherwise. It is called locally small if the morphisms between any two objects form a set but objects fail to. For instance, the category **Set** of sets and functions is a locally small category as the collection of all sets does not form a set while for any two sets, there is a set of functions between them. These distinctions are important when working with categories. For instance, a category is said to be complete if it has the limit of all *small* diagrams ($\mathcal{F} : \mathcal{C} \to \mathcal{D}$ is a small diagram if $\mathcal{C}$ is a small category). For instance, **Set** is complete but does not have the cartesian product of all large families of sets.

These terminology and considerations are due to the fact that the original foundations of category theory by Eilenberg and Mac Lane were laid on top of NGB (von Neumann-Gödel-Bernays) set theory. In NBG, in addition to sets, the notion of a class (a collection of sets which itself is not *necessarily* a set) is also formalized. For any property φ, there is a class C_φ of all sets that have property φ. If the collection of sets satisfying φ forms a set then C_φ is just that set. Otherwise, C_φ is said to be a proper class. In this formalism, one can formalize large categories but cannot use them. For instance, the functor category $\mathbf{Set}^{\mathbf{Set}}$ is not defined as its objects are already proper classes and there is no class of proper classes in NBG.

The other foundation that is probably the most popular among mathematicians is that of ZF with Grothendieck's axiom of universe. Roughly speaking, a Grothendieck universe V is a set that satisfies ZF axioms, e.g., if $A \in V$ and $B \in V$ then $\{A, B\} \in V$ (axiom of pairing), if $A \in V$ then $2^A \in V$ (axiom of power set), etc. We also have if $A \in B$ and $B \in V$ then $A \in V$. *Grothendieck's axiom* says that for any set x there is a Grothendieck universe V such that $x \in V$. This also implies that for any Grothendieck universe V, there is a Grothendieck universe V' such that $V \in V'$.

Working on top of this foundation, one can talk about V-small categories and use all the set-theoretic power of ZF. The notion of completeness for a V-small category can be defined as having all V-small limits. The category of all V-small sets will be a V'-small category where $V \in V'$. It is also a V-locally-small category as its set of morphisms are V-small but its set of objects fail to be. For more details on foundations for category theory see chapter 12 of McLarty's book [13].

The type hierarchy of Coq (also known as universes), as explained below, bears a striking resemblance to Grothendieck universes just explained. In the rest of this section we discuss how Coq's new universe polymorphism feature allows us to use Coq universes instead of Grothendieck universes in a completely transparent way. That is, we never mention any universes in the whole of the development and Coq's universe inference algorithm (part of the universe polymorphism feature) infers them for us.

2.1 Coq's Universes

In higher-order dependent type theories such as that of Coq, types are also terms and themselves have types. As expected, allowing existence of a type of all types results in self-referential paradoxes, such as Girard's paradox [6]. Thus, to avoid such paradoxes type theories like Coq use a countably infinite hierarchy of types of types (also known as universes): $\texttt{Type}_0 : \texttt{Type}_1 : \texttt{Type}_2 : \ldots$ The type system of Coq additionally has the cumulativity property, i.e., for any term $\texttt{T} : \texttt{Type}_n$ we also have $\texttt{T} : \texttt{Type}_{n+1}$.

The type system of Coq has the property of *typical ambiguity*. That is, in writing definitions, we don't have to specify universe levels and/or constraints on them. The system automatically infers the constraints necessary for the definitions to be valid. In case, the definition is such that no (consistent) set of constraints can be inferred, the system rejects it issuing a "universe inconsistency" error. It is due to this feature that throughout this development we have not had the need to specify any universe levels and/or constraints by hand.

To better understand typical ambiguity in Coq, let's consider the following definition.

```
Definition Tp := Type.
```

In this case, Coq introduces a new universe variable for the level of the type `Type`. That is, internally, the definition looks like[1]:

```
Definition Tp : Type@{i+1} := Type@{i}.
```

[1] `Type@{i}` is Coq's syntax for $\texttt{Type}_i$.

Note that in older version of Coq and when universe polymorphism is not enabled in Coq 8.5 the universe level `i` above is a global universe level, i.e., it is fixed. Hence, the following definition is rejected with a universe inconsistency error.

```
Definition TpTp : Tp := Tp.
```

The problem here is that this definition requires (`Type@{i} : Type@{i}`) which requires the system to add the constraint `i < i` which makes the set of constraints inconsistent. Without universe polymorphism, one way to solve this problem would be to duplicate the definition of `Tp` as `Tp'` which would be internally represented as:

```
Definition Tp' : Type@{j+1} := Type@{j}.
```

Now we can define `TpTp'`:

```
Definition TpTp' : Tp' := Tp.
```

which Coq accepts and consequently adds the constraint `i < j` to the global set of universe constraints. As these constraints are global however, after defining `TpTp'` we can't define `Tp'Tp`

```
Definition Tp'Tp : Tp := Tp'.
```

This is rejected with a universe inconsistency error as it requires `j < i` to be added to the global set of constraints which makes it inconsistent as it already contains `i < j` from `TpTp'`.

2.2 Universe Polymorphism

Coq has recently been extended [18] to support universe polymorphism. This feature is now included in the recently released Coq 8.5. When enabled, universe levels of a definition are bound at the level of that definition. Also, any universe constraints needed for the definition to be well-defined are local to that definition. That is the definition of `Tp` defined above is represented internally as:

```
Definition Tp@{i} : Type@{i+1} := Type@{i}. (* Constraints: *)
```

Note that the universe level `i` here is local to the definition. Hence, `Tp` can be instantiated at different universe levels. As a result, the definition of `TpTp` above is no longer rejected and is represented internally as:

```
Definition TpTp@{i j} : Tp@{j} := Tp@{i}. (* Constraints: i < j *)
```

That is, the two times `Tp` is mentioned, two different instances of it are considered at two different universe levels `i` and `j` resulting in the constraint `i < j` for the definition to be well-defined.

Note the resemblance between universes in Coq and Grothendieck universes. E.g., the fact that if `A : Type@{i}` and `B : Type@{i}` then `{x : Type@{i} | x = A ∨ x = B} : Type@{i}`, cumulativity, etc.

In the sequel, in some cases, we only show the internal representation of concepts formalized in Coq.

2.3 Smallness and Largeness

In this implementation, we use universe levels as the underlying notion of smallness/largeness. In other words, we simply ignore smallness and largeness of constructions and simply allow

Coq to infer the necessary conditions for definitions to be well-defined. We define categories without mentioning universe levels. They are internally represented as:

```
Record Category@{i j} :=
   {
      Obj : Type@{i};
      Hom : Obj → Obj → Type@{j};
      ...
   } : Type@{max(i+1, j+1)} (* Constraints: *)
```

The category of (small) categories is internally represented as:

```
Definition Cat@{i j k l} :=
   {|
      Obj := Category@{k l};
      Hom := fun (𝒞 𝒟 : Category@{k l}) ⇒ Functor@{k l k l} 𝒞 𝒟;
      ...
   |} : Category@{i j} (* Constraints: k < i, l < i, k ≤ j, l ≤ j *)
```

That is, **Cat** has as objects categories that are small compared to itself.

Having a universe-polymorphic **Cat** means for any category $\mathcal{C}$ there is a version of **Cat** that has $\mathcal{C}$ as an object. Therefore, for example, to express the fact that two categories are isomorphic, we simply use the general definition of isomorphism in the specific category **Cat**. This means we can use all facts and lemmas proven for isomorphisms, for isomorphisms of categories with no further effort required.

The category of types (representation of **Set** in Coq) is internally represented as:

```
Definition Set@{i j} :=
   {|
      Obj := Type@{j};
      Hom := fun (A B : Type@{j}) ⇒ A → B;
      ...
   |} : Category@{i j} (* Constraints: j < i *)
```

The constraint j < i above is exactly what we expect as **Set** is locally small. The reason that Coq's universe inference algorithm produces this constraint is that the type of objects of **Set** is Type@{j} which itself has type Type@{i}. But, the homomorphisms of this category are functions between two types whose type is Type@{j}. Thus, the type of homomorphisms themselves is Type@{j}. For details of typing rules for function types see the manual of Coq [12].

Complete Small Categories are Mere Preorder Relations! Perhaps the best showcase of using the new universe polymorphism of Coq to represent smallness/largeness can be seen in the theorem below which simply implies that any complete category is a preorder category, i.e., there is at most one morphism between any two objects.

```
Theorem Complete_Preorder (𝒞 : Category) (CC : Complete 𝒞) :
      forall x y : Obj 𝒞, Hom x y' ≃ ((Arrow 𝒞) → Hom x y)
```

where y' is the limit of the constant functor from the discrete category **Discr**(Arrow $\mathcal{C}$) that maps every object to y, (Arrow $\mathcal{C}$) is the type of all homomorphisms of category $\mathcal{C}$ and $\simeq$ denotes isomorphism. In other words, for any pair of objects x and y the set of functions from the set of all morphisms in $\mathcal{C}$ to the set of morphisms from x to y is isomorphic to the set of morphisms from x to some constant object y'. This though, would result in a contradiction as soon as we have two objects A and B in $\mathcal{C}$ for which the collection of morphisms from A to B has more than one element. Hence, we have effectively shown that *any* complete category is a preorder category.

This is indeed absurd as the category **Set** is complete and there are types in Coq that have more than one function between them! However, this theorem holds for small (in the conventional sense) categories. That is, any *small and complete* category is a preorder category[2].

As expected, the constraints on the universe levels of this theorem that are inferred by Coq do indeed confirm this fact. That is, this theorem is in fact only applicable to a category $\mathcal{C}$ for which the level of the type of objects is less than or equal to the level of the type of arrows. This is in direct conflict with the constraints inferred for **Set** as explained above. Hence, Coq will refuse to apply this theorem to the category **Set** with a universe inconsistency error.

2.4 Limitations Imposed by Using Universe Levels for Smallness and Largeness

The universe polymorphism of Coq, as explained in Sozeau et al. [18], treats inductive types by considering copies of them at different levels. Furthermore, if a term of a universe polymorphic inductive type is assumed to be of two instances of that inductive type with two different sets of universe level variables, additional constraints are imposed so that the corresponding universe level variables in those two sets are required to be equal. As records are a special kind of inductive types, the same holds for them. For us, this implies that if we have $\mathcal{C}$: `Category@{i j}` and we additionally have that $\mathcal{C}$: `Category@{i’ j’}`, Coq enforces `i` = `i’` and `j` = `j’`. This means, **Cat**`@{i j k l}` is in fact *not* the category of *all* smaller categories. Rather it is the category of smaller categories that are at level `k` and `l` and not any lower level.

Apart from the fact that **Cat** defined this way is not the category of all relatively small categories, these constraints on universe levels impose practical restrictions as well. For instance, looking at the fact that **Cat**`@{i j k l}` has exponentials (functor categories), we can see the constraints that `j` = `k` = `l`. Consequently, only those copies have exponentials for which this constraints holds. Looking back at **Set**, we had the constraint that the level of the type of morphisms is strictly less than that of objects. This means, there is no version of **Cat** that both has exponentials and a version of **Set** in its objects.

Moreover, we can use the Yoneda lemma to show that in any cartesian closed category, for any objects a, b and c:

$$\left(a^b\right)^c \simeq a^{b\times c} \tag{1}$$

Yet, this theorem can’t be applied to **Cat**, even though it holds for **Cat**.

It is worth noting that although the category **Cat**`@{i j k l}` is the category of all categories `Category@{k l}` and not lower, for any lower category it contains an “isomorphic copy” of that category. That is any category $\mathcal{C}$: `Category@{k’ l’}` such that $\mathtt{k}' \leq \mathtt{k}$ and $\mathtt{l}' \leq \mathtt{l}$ can be “lifted” to `Category@{k l}`. Such a lifting function can be simply written as:

```
Definition Lift (𝒞 : Category@{k’ l’}) : Category@{k l} :=
  {| Obj := Obj 𝒞; Hom := Hom 𝒞; . . . |}.
```

[2] This theorem and its proof are taken from Awodey’s book [2].

and the appropriate constraints, i.e., $\mathtt{k}' \leq \mathtt{k}$ and $\mathtt{l}' \leq \mathtt{l}$ are inferred by Coq. However, working with such liftings is in practice cumbersome as in interesting cases where $\mathtt{k}' < \mathtt{k}$ and/or $\mathtt{l}' < \mathtt{l}$, we can't prove or even specify `Lift` $\mathcal{C} = \mathcal{C}$ as it is ill-typed. This means, any statement regarding $\mathcal{C}$ must be proven separately for `Lift` $\mathcal{C}$ in order for them to be useful for the lifted version.

It is possible to alleviate these problems if we have support for cumulative inductive types in Coq, as proposed in Timany et al. [24]. In such a system, any category $\mathcal{C}$: `Category@{i j}` will also have type `Category@{k l}` so long as the constraints $\mathtt{i} \leq \mathtt{k}$ and $\mathtt{j} \leq \mathtt{l}$ are satisfied.

However, these limitations are not much more than a small inconvenience and in practice we can work in their presence with very little extra effort. At least as far as basic category theory goes. Our development is an attestation to that.

2.5 Smallness and Largeness in Other Developments

In homotopy type theory (HoTT) [20] a category $\mathcal{C}$ has a further constraint that for any two objects A and B the set of morphisms from A to B must form an hSet (a homotopy type-theoretical concept). On the other hand, for two categories $\mathcal{C}$ and $\mathcal{D}$, the set of functors from $\mathcal{C}$ to $\mathcal{D}$ does not necessarily form an hSet. It does however when the set of objects of $\mathcal{D}$ forms an hSet. Therefore, in HoTT settings one can construct the category of small *strict* categories, i.e., small categories whose type of objects forms an hSet, and not the category of all small categories. However, the category of small strict categories itself is not strict. Hence, contrary to the category **Cat** in our development, there is no category (in the HoTT sense, i.e., one whose objects form an hSet) that has the category of small strict categories as one of its objects. In this regard, working in HoTT is similar to working in NBG rather than ZF with Grothendieck universes.

The situation regarding the category of small strict categories discussed above is due to the fact that homotopy type-theoretical levels for types (e.g., hSet) concern a notion of (homotopy theoretical) *complexity* rather than cardinality. In fact, in other situations, e.g., in defining limits of functors, where cardinality is concerned universe levels can be used to express smallness and largeness. In other words, in HoTT settings, when defining limits, one can simply not mention universe levels and let Coq infer that the definition of limit for a functor $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ is well-defined whenever, $\mathcal{C}$ is relatively small compared to $\mathcal{D}$. This also means that the restrictions mentioned above are also present in HoTT settings when universe levels are used to represent smallness and largeness. For instance isomorphism 1 above can't be proven in **Cat** using the Yoneda lemma even if a, b and c are strict categories.

This is how smallness and largeness works in both Gross et al. [7] and Ahrens et al. [1]. This is also the case for our development when ported on top of the HoTT library [19]. As one consequence, contrary to what was explained above, in migrating to the HoTT library settings we can't simply consider the isomorphism of categories as the general notion of isomorphism in the specific case of **Cat**.

In Huet et al. [9], working in Coq 8.4, the authors define a duplicate definition of categories, `Category`', tailored to represent large categories. This way, they form the `Category`' of categories (`Category`) – much like we used `Tp'` above.

Peebles et al. [16] however use universe levels to represent smallness and largeness. But working in Agda which provides no typical ambiguity or cumulativity, they have to hand code all universe levels everywhere; whereas we rely on Coq's inference of constraints to do the hard work. Noteworthy is also the fact that their categories have three universe variables instead of our two. One for the level of the type of objects, one for the level of the type of morphisms and one for the level of the type of the setoid equality for their setoids of morphisms.

3 Concepts Formalized, Features and Comparison

In this development we have formalize most of the basic category theory. Here, by basic we mean not involving higher (e.g., 2-categories), monoidal or enriched categories. This spans over the simple yet important and useful basic concepts like terminal/initial objects, products/sums, equalizers/coequalizers, pullbacks/pushouts and exponentials on the one hand and adjunctions, Kan extensions, (co)limits (as (left)right local Kan extensions) and toposes on the other.

The well-behaved dualities (in the sense discussed above) allow us to simply define dual notions, just as duals of their counterparts, e.g., initial objects as terminal objects of the dual category or the local left Kan extension of $\mathcal{F}$ along $\mathcal{G}$ as the local right Kan extension of $\mathcal{F}^{op}$ along $\mathcal{G}^{op}$.

3.1 Concepts Formalized: Generality and Diversity

Throughout this development we have tried to formalize concepts in as general a way as possible so long as they are comfortably usable. For instance, we define (co)limits as (left)right local Kan extensions along the unique functor to the terminal category. By doing so, we can extend facts about them to (co)limits. As an example, consider (left)right adjoints preserving (co)limits and (co)limit functors being adjoint to Δ explained below.

Different versions of adjunction and Kan extensions. In this formalization, we have multiple versions of the definition of adjunctions and Kan extensions. In particular, we define unit-universal morphism property adjunction, unit-co-unit adjunction, universal morphism adjunction and hom-functor adjunction. For these different versions, we provide conversions to and from the unit-universal morphism property definition which is taken to be the main definition. This definition is also taken to be the main definition of adjunction in Awodey's book [2]. For local Kan extensions, we define them as (initial)terminal (co)cones along a functor as well as through the hom-functor. Global Kan extensions are simply defined through adjunctions.

The main reason for this diversity, aside from providing a versatile category theory library, is the fact that each of these definitions is most suitable for some specific purpose.

For instance, using the hom-functor definition of adjunctions makes it very easy to prove that isomorphic functors have the same adjoints: $\mathcal{F} \simeq \mathcal{F}' \Rightarrow \mathcal{F} \dashv \mathcal{G} \Rightarrow \mathcal{F}' \dashv \mathcal{G}$, duality of adjunction: $\mathcal{F} \dashv \mathcal{G} \Rightarrow \mathcal{G}^{op} \dashv \mathcal{F}^{op}$, and uniqueness of adjoint functors: $\mathcal{F} \dashv \mathcal{G} \Rightarrow \mathcal{F}' \dashv \mathcal{G} \Rightarrow \mathcal{F} \simeq \mathcal{F}'$. The last case simply follows from the Yoneda lemma. On the other hand, the unit-universal morphism property definition of adjunctions together with the definition of Kan extensions as cones along a functor provide an easy way to convert from local to global Kan extensions.

Universal morphism adjoints in practice express sufficient conditions for a functor to have a (left)right adjoint. That is, a functor $\mathcal{G} : \mathcal{C} \to \mathcal{D}$ is a right adjoint (has a left adjoint functor) if the comma category $(x \downarrow \mathcal{G})$ has a terminal object for any $x : \mathcal{D}$. As we will briefly discuss below, (left)right adjoint functors preserve (co)limits. Freyd's adjoint functor theorem gives an answer to the question "when is a functor that preserves all limits a right adjoint (has a left adjoint functor)". Universal morphism adjoints appear in this theorem and that's why we have included them in our formalization.

(Left)right adjoints preserve (co)limits. Awodey [2] devotes a whole section to this fact with the title "RAPL" (Right Adjoints Preserve Limits). For a better understanding of this fact and perhaps the concept of adjunctions, let us draw intuition from categorical interpretations of logic. In categorical interpretations of logic, the existential and universal quantifiers are interpreted as left and right adjoints to some functor while conjunctions and disjunctions are defined as products and sums respectively which respectively are in turn limits and co-limits (see Jacobs' book [10] for details). In this particular case, RAPL and its dual boil down to: $\forall x.\ P(x) \wedge Q(x) \Leftrightarrow \forall x.\ P(x) \wedge \forall x.\ Q(x)$ and $\exists x.\ P(x) \vee Q(x) \Leftrightarrow \exists x.\ P(x) \vee \exists x.\ Q(x)$. We prove this fact in general for (left)right local Kan extensions. To this end, the unit-co-unit definition of adjunctions is the easiest to use to prove the main lemma which along with hom-functor definition of Kan extensions proves that (left)right adjunctions preserve (left)right Kan extensions. That is for an adjunction $\mathcal{L} \dashv \mathcal{R}$ where $\mathcal{R} : \mathcal{D} \to \mathcal{E}$ and $\mathcal{L} : \mathcal{E} \to \mathcal{D}$ if in the diagram on the left $\mathcal{H}$ is the local right Kan extension of $\mathcal{F}$ along $\mathcal{P}$ then in the right diagram $\mathcal{R} \circ \mathcal{H}$ is the local right Kan extension of $\mathcal{R} \circ \mathcal{F}$ along $\mathcal{P}$:

$$\begin{tikzcd}
\mathcal{C} \arrow[r, "\mathcal{F}"] \arrow[d, "\mathcal{P}"'] & \mathcal{D} \\
\mathcal{C}' \arrow[ur, "\mathcal{H}"'] &
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\mathcal{C} \arrow[r, "\mathcal{R}\circ\mathcal{F}"] \arrow[d, "\mathcal{P}"'] & \mathcal{E} \\
\mathcal{C}' \arrow[ur, "\mathcal{R}\circ\mathcal{H}"'] &
\end{tikzcd}$$

The case of (co)limits follows immediately. In Coq we show this by constructing a local right Kan extension (using the hom-functor definition) of $\mathcal{R} \circ \mathcal{F}$ along $\mathcal{P}$ where the Kan extension functor (`HLRKE`) is $\mathcal{R}$ composed with the Kan extension functor of $\mathcal{F}$ along $\mathcal{P}$:

```
Definition Right_Adjoint_Preserves_Hom_Local_Right_KanExt
   {𝒞 𝒞' : Category} (𝒫 : Functor 𝒞 𝒞') {𝒟 : Category} (ℱ : Functor 𝒞 𝒟)
   (hlrke : Hom_Local_Right_KanExt 𝒫 ℱ)
   {ℰ : Category} {ℒ : Functor ℰ 𝒟} {ℛ : Functor 𝒟 ℰ} (adj : UCU_Adjunct ℒ ℛ)
: Hom_Local_Right_KanExt 𝒫 (ℛ ∘ ℱ) :=
    {|
       HLRKE := (ℛ ∘ (HLRKE hlrke));
       HLRKE_Iso := ...
    |}.
```

(Co)limit functors are adjoint to Δ. In order to show that (co)limits are adjoint to the diagonal functor (Δ) we simply use the fact that local (left)right Kan extensions assemble together to form (left)right global Kan extensions. As global Kan extensions are defined as (left)right adjoints to the pre-composition functor, putting these two facts together, we effortlessly obtain that (co)limits form functors which are (left)right adjoint to Δ.

Cardinality restrictions. We introduce the notion of cardinality restriction in the category **Set**. A cardinality restriction is a property over types (objects of **Set**) such that if it holds for some type, it must hold for any other type isomorphic (in **Set**) to it. That is, if a cardinality restriction holds for a type, it must hold for any other type with the same cardinality.

```
Record Card_Restriction : Type :=
  { Card_Rest : Type → Prop;
    Card_Rest_Respect : forall (A B : Type),
      (A ≃≃ B ::> Set) → Card_Rest A → Card_Rest B }.
```

The type (`A ≃≃ B ::>` **Set**) is the type of isomorphisms $A \simeq B$ in **Set**. As an example, the cardinality restriction corresponding to finiteness is defined as follows.

```
Definition Finite : Card_Restriction :=
  {| Card_Rest := fun A ⇒ inhabited {n : nat & (A ≃≃ {x : nat | x < n} ::> Set)}; . . . |}.
```

The definition above basically says that a type A is finite if there exists some n such that A is isomorphic to the type $\{x : \texttt{nat} \mid x < n\}$ of natural numbers less than n.

(Co)limits restricted by cardinality. We use the notion of cardinality restrictions above to define (co)limits restricted by cardinality. For a cardinality restriction P, we say a category $\mathcal{C}$ has (co)limits of cardinality P ($\mathcal{C}$ is P-(co)complete) if for all functors $\mathcal{F} : \mathcal{D} \to \mathcal{C}$ such that $P(Obj_{\mathcal{D}})$ and $\forall AB \in Obj_{\mathcal{D}}, P(Hom(A, B))$, $\mathcal{C}$ has the (co)limit of $\mathcal{F}$.

```
Definition Has_Restr_Limits (𝒞 : Category) (P : Card_Restriction) :=
  forall {𝒥 : Category} (ℱ : Functor 𝒥 𝒞), P 𝒥 → P (Arrow 𝒥) → Limit ℱ.
```

We state several lemmas about cardinality restricted (co)completeness, e.g., if a category has all limits of a specific cardinality its dual has all co-limits of that cardinality.

```
Definition Has_Restr_Limits_to_Has_Restr_CoLimits_Op
        {𝒞 : Category} {P : Card_Restriction}
        (HRL : Has_Restr_Limits 𝒞 P) : Has_Restr_CoLimits (𝒞^op) P := . . .
```

This also allows us to define a topos, simply as a category that is cartesian closed, has all finite limits and a subobject classifier where finiteness is represented as a cardinality restriction.

```
Class Topos : Type :=
  { Topos_Cat : Category;
    Topos_Cat_CCC : CCC Topos_Cat;
    Topos_Cat_Fin_Limit : Has_Restr_Limits Topos_Cat Finite;
    Topos_Cat_SOC : SubObject_Classifier Topos_Cat }.
```

(Co)Limits by (Sums)Products and (Co)Equalizers. A discrete category is a category where the only morphisms are identities. That is, any set can induce a discrete category by simply considering the category which has as objects members of that set and the only morphisms are identity morphisms. We define the discrete category of a type A as a category, $\mathbf{Discr}(A)$ with terms of type A as objects and the collection of morphisms from an object x to an object y are proofs of equality of $x = y$.

```
Definition Discr_Cat (A : Type) : Category := {|Obj := A; Hom := fun a b ⇒ a = b; . . . |}.
```

Similarly, a discrete functor is a functor that is induced from a mapping f from a type A to objects of a category $\mathcal{C}$:

```
Definition Discr_Func {𝒞 : Category} {A : Type} (f : A → 𝒞) : Functor (Discr_Cat A) 𝒞 :=
     {| FO := f; . . . |}.
```

We define the notion of generalized (sums)products to be that of (co)limits of functors from a discrete category.

```
Definition GenProd {A : Type} {𝒞 : Category} (f : A → 𝒞) := Limit (Discr_Func f).
```

We use these generalized (sums)products to show that any category that has all generalized (sums)products and (co)equalizers has all (co)limits. We also prove the special case of cardinality restricted (co)limits. Using the notions explained above, we show that given a cardinality restriction P if a category has (co)equalizers as well as all generalized (sums)products that satisfy P, then that category is P-(co)complete.

```
Definition Restr_GenProd_Eq_Restr_Limits
   {C : Category} (P : Card_Restriction)
   {CHRP : forall (A : Type) (f : A → 𝒞), (P A) → (GenProd f)}
   {HE : Has_Equalizers C}
: Has_Restr_Limits C P := ...
```

Categories of Presheaves. To the best of our knowledge, ours is the only category theory development featuring facts about categories of presheaves such as their (co)completeness, and being a topos. The category of presheaves on $\mathcal{C}$, ($\mathbf{PSh}(\mathcal{C})$), is a category whose objects are functors of the form $\mathcal{C}^{op} \to \mathbf{Set}$ and whose morphisms are natural transformations. In other words, a presheaf $P : \mathcal{C}^{op} \to \mathbf{Set}$ on $\mathcal{C}$ is a collection of sets indexed by objects of $\mathcal{C}$ such that for a morphism $f : A \to B$ in $\mathcal{C}$, there is a function (a conversion if you will) $P(f) : P(B) \to P(A)$ in $\mathbf{Set}$. Presheaves being toposes, each come with their own logic. As an example, Birkedal et al. [4] show that the logic of the category of presheaves on ω (the preorder of natural numbers considered as a category) corresponds to the step-indexing technique used in the field of programming languages and program verification. For more details about elementary properties of categories of presheaves see Awodey's book [2]. There categories of presheaves are called categories of diagrams.

3.2 Comparison

Tables 1 and 2 give an overall comparison of our development with select other implementations of category theory of comparable extent. These tables mention only the most notable features and concepts formalized and do not contain many notions and lemmas in these developments. Notice also that the list of concepts and features appearing in these tables is by no means exhaustive and is not the union of all formalized concepts and features of these developments. In these tables, our development is the first column.

3.3 Axioms

One axiom that is used ubiquitously throughout the development is the uniqueness of proofs of equality.

```
forall (A : Type) (x y : A) (p q : x = y), p = q
```

We in practice enforce this axiom using proof-irrelevance (as `p` and `q` are proofs). To facilitate the use of this axiom, we prove a number of lemmas, e.g.:[3]

```
Lemma Functor_eq_simplify (𝒞 𝒟 : Category) (ℱ 𝒢 : Functor 𝒞 𝒟) :
      (FO ℱ = FO 𝒢) → (FA ℱ = FA 𝒢) → ℱ = 𝒢
```

which says two functors are equal if their object and arrow maps are. If so, the proofs that the arrow maps preserve identity and composition are just assumed equal using proof-irrelevance (uniqueness of equality proofs).

[3] This is an over-simplification: in practice types of `FA` $\mathcal{F}$ and `FA` $\mathcal{G}$ don't match and therefore their equality as stated here is ill-typed. In practice, we adjust the type of `FA` $\mathcal{F}$ using the equality of object maps.

Table 1 Comparison of features and concepts formalized with a few other implementations of comparable extent.

Concept / Feature	[21]	[7]	[9]	[1]	[16]
Automation	partial	✓			
Based on HoTT	in [22]$^\sharp$	✓		✓	
Setoid for Morphisms			✓		✓
Assumes UIP or equivalent	few restricted cases				✓
Basic constructions:					
Terminal/Initial object	✓	✓	✓	✓	✓
Products/Sums	✓		✓	✓	✓
Equalizers/Coequalizers	✓		✓		
Pullbacks/Pushouts	✓		✓	✓	✓
Basic constructions above are (co)limits	✓		✓		
exponentials	✓		✓		✓
Subobject classifier	✓			✓	✓
External constructions:					
Comma categories	✓	✓	✓	✓	✓
Product category	✓	✓	✓	✓	✓
Sum category		✓			
Cat. of categories (**Cat**):	✓	✓	✓		✓
Cartesian closure	✓		✓		
Initial/terminal object	✓	✓	✓		✓
Category of sets (**Set**):	✓	✓	✓	✓	✓
Basic (co)limits	✓	init./term.	partial		
(Local†)Cartesian closure	✓		CCC		
(Co†)Completeness	✓		comp.		✓
Sub-object classifier	(`Prop : Type`)†				
Topos	✓†				
Hom functor	✓	✓	✓	✓	✓
Fully-faithful functors	✓	✓		✓	✓
Essentially (inj)sur-jective functors	✓	✓		✓	✓
The Yoneda lemma	✓	✓	✓	✓	✓
Monoidal Categories		partial			✓
Enriched Categories		partial			partial
2-categories					✓
Pseudo-functors		✓			✓
(Co)monads and algebras :					
(Co)Monad				✓	✓
T-(co)algebras (T: an endofunctor)	✓			✓	✓
Eilenberg Moore cat.					✓
Kleisli cat.					✓

†Uses the axioms: propositional extensionality and constructive indefinite description (choice).
$^\sharp$The version of our development we are migrating to HoTT settings, on top of HoTT library.

Table 2 Comparison of features and concepts formalized with a few other implementations of comparable extent (cont.).

Concept / Feature	[21]	[7]	[9]	[1]	[16]
Adjunction	✓	✓	✓		✓
Unit-universal morphism adjunction	✓	✓			
Hom-functor adjunction	✓	✓	✓		
Unit-counit adjunction	✓	✓	✓	✓	✓
Universal morphism adjunction	✓	✓	✓		
Uniqueness up to natural isomorphism	✓				
Naturally isomorphic functors have	✓				
the same left/right adjoints					
Adjoint composition laws	✓	✓			✓
Category of adjunctions	✓				
(objects: categories; morphisms: adjunctions)					
Partial adjunctions		✓			
Adjoint Functor Theorem	✓				✓
Kan extensions	✓	✓			✓
Global definition	✓	✓		✓	
Local definition	✓	✓			
Through hom-functor	✓				
Through cones (along a functor)	✓				✓
Through partial adjoints		✓			
Uniqueness	✓				
Preservation by adjoint functors	✓				
Naturally isomorphic functors form	✓				
the same left/right Kan extension					
Pointwise kan extensions	✓	✓			
(preserved by representable functors)					
(Co)Limits	✓	✓	✓	✓	✓
As (left)right kan extensions	✓	✓			
As (initial)terminal (co)cones			✓	✓	✓
(Sum)Product-(co)equalizer (co)limits	✓				
(Co)Limit functor	✓	✓			
(Co)Limits functor adjoint to Δ	✓	✓			
(Co)limits restricted by cardinality	✓				
Pointwise (as kan extensions), i.e.,	✓		✓		
preserved by Hom functor					
Category of presheaves over $\mathcal{C}$ ($\mathbf{PSh}_{\mathcal{C}}$):	✓				✓
Terminal/Initial object	✓				
Products/Sums	✓				
Equalizers/Coequalizers	✓†				
Pullbacks	✓				
Cartesian closure	✓				
Completeness/Co-completeness	✓†				
Sub-object classifier (Sieves)	✓†				
Topos	✓†				

† Uses the axioms: propositional extensionality and constructive indefinite description (choice).

Using uniqueness of equality proofs in the definition of categories is an essential necessity. As otherwise, as explained in the HoTT book [20], the category defined is not a category but a form of higher category. That's why in any formalization of category theory this axiom is assumed or enforced in one way or another.

In homotopy type theory (HoTT) settings, assuming uniqueness of proofs of equality in general is in direct contradiction with the univalence axiom which sits at the heart of HoTT. Therefore in developments of category theory on top of HoTT, e.g., Gross et al. [7] and Ahrens et al. [1], they include the fact that proofs of equalities of morphisms are unique as part of the definition of a category. This is precisely the requirement that collections of morphisms should form hSets discussed above.

In developments using setoids, e.g. Huet et al. [9] and Peebles et al. [16], the authors customize the setoid equalities so that proofs are never considered. For instance, they *define* the setoid equality for functors so that two functors are equal whenever their object and morphism maps are.

We are currently in the process of porting a version of our development on top of the HoTT library[4]. There we also stop using this axiom and change the definition of categories. As expected almost all of the cases where we use uniqueness of proofs of equality (in a direct or indirect way) are not problematic in HoTT settings, i.e., they are applied to equality of morphisms. However, there are a few limited cases were they are not. Some of these cases are no longer relevant in the HoTT settings and some others are very easily surmountable. For more details of our ongoing effort of porting this development on top of the HoTT library see the extended version of this paper [25].

Apart from the axiom of uniqueness of proofs of equality, we have made frequent use of the axiom of functional extensionality. However, this axiom is a consequence of the univalence axiom and is in fact provided in the HoTT library and frequently used therein.

We have in particular taken advantage of two other axioms, propositional extensionality and axiom of choice (constructive indefinite description in the library of Coq) which we have used, e.g., to construct co-limits in **Set** and presheaf categories. Along with using setoids, using these axioms to represent quotient types in type theory is standard practice. We plan to use higher inductive types, as explained in the HoTT book [20], to construct such co-limits in the version ported on top of the HoTT library.

4 Future Work: Building on Categories

We believe that this development is one that provides a foundation for other works based on category-theoretical foundations. We have plans to make use of the foundation of category theory that has been laid in this work. In particular, we plan to make use of this foundation for mechanization of categorical logic (see Jacobs' book [10]) and higher order separation logic (see Biering et al. [3]) for the purpose of using them as foundations for mechanization of program verification. In particular, the theory of presheaves developed provides a basis for formalization of the internal logic of presheaf categories with a particular interest in the topos of trees [4].

In this regard, we have already used this development as a foundation to formalize the theory of Birkedal et al. [5] to solve category theoretical recursive ultra-metric space equations [23]. In Birkedal et al. [5], the authors use the theory of ultra-metric spaces to build unique (up to isomorphism) fixed-points of particular category-theoretical recursive domain-theoretic

[4] The version being ported on top of the HoTT library can be found at GitHub [22].

equations. More precisely, they construct fixed-points of a particular class of mixed variance functors, i.e., functors of the form $\mathcal{F} : (\mathcal{C}^{op} \times \mathcal{C}) \to \mathcal{C}$. Solutions to such mixed-variance functors can for example be used to construct models for imperative programming languages. Successful implementation of this theory [23] on top of our general foundation of categories, although arguably not huge, is evidence that this development is fit for being used as a general-purpose foundation.

In Birkedal et al. [5], the authors define the notion of an M-category to be a category in which the set of morphisms between any two objects form a non-empty ultra-metric space. In our formalization, based on a general theory of ultra-metric spaces, we define M-categories as categories in which the type of morphisms between any two objects forms an ultra-metric space, dropping the rather strong non-emptiness requirement. We instead require some weaker conditions which still allow us to form fixed-points.

An interesting instance of M-categories is the presheaf topos of the preorder category of natural numbers, i.e., the topos of trees. In our development, just showing that this category qualifies as an M-category is sufficient to immediately be able to construct desired fixed-points. This is due to the fact that in the foundations provided, all necessary conditions for an M-category to allow formation of solutions, e.g., existence of limits of a particular class of functors is already established.

5 Conclusion

The most important conclusion of this paper is that Coq 8.5 with its new features: η for records and universe polymorphism, is next to ideal for formalization of category theory and related parts of mathematics. We believe that Coq 8.5 is the first version of Coq that makes it possible to lay a truly useful and versatile general purpose category theoretical foundation as we have demonstrated.

In summary, we surveyed our development of the foundations of category theory. This development features most of the category-theoretical concepts that are formalized in most other such developments and some more. We pushed the limits of the new feature of universe polymorphism and the constraint inference algorithm of Coq 8.5 by using them to represent relative smallness/largeness. As discussed, it gives very encouraging results despite the restrictions imposed by not having cumulative inductive types.

We have successfully used this implementation as the categorical foundation to build categorical ultra-metric space theoretic fixed-points of recursive domain equations. This seems an encouraging initial indication that this work is fit to perform the important role of a general purpose category theoretical foundation for other developments to build upon.

Acknowledgements. We would like to thank Jason Gross, Matthieu Sozeau and Georges Gonthier for helpful discussions. We are also grateful to the anonymous reviewers for their invaluable comments and suggestions.

This work was funded by EU FP7 FET-Open project ADVENT under grant number 308830.

References

1 Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. Univalent categories and the rezk completion. *Mathematical Structures in Computer Science*, 25(05):1010–1039, jan 2015. URL: https://github.com/benediktahrens/rezk_completion, doi:10.1017/s0960129514000486.

2 Steve Awodey. *Category theory*. Oxford University Press, 2010.

3 Bodil Biering, Lars Birkedal, and Noah Torp-Smith. Bi-hyperdoctrines, higher-order separation logic, and abstraction. *ACM Trans. Program. Lang. Syst.*, 29(5), August 2007. doi:10.1145/1275497.1275499.

4 Lars Birkedal, Rasmus Ejlers Mogelberg, Jan Schwinghammer, and Kristian Stovring. First steps in synthetic guarded domain theory: Step-indexing in the topos of trees. In *2011 IEEE 26th Annual Symposium on Logic in Computer Science*. Institute of Electrical & Electronics Engineers (IEEE), jun 2011. doi:10.1109/lics.2011.16.

5 Lars Birkedal, Kristian Støvring, and Jacob Thamsborg. The category-theoretic solution of recursive metric-space equations. *Theoretical Computer Science*, 411(47):4102–4122, oct 2010. doi:10.1016/j.tcs.2010.07.010.

6 T. Coquand. An analysis of Girard's paradox. Technical Report RR-0531, INRIA, May 1986. URL: https://hal.inria.fr/inria-00076023.

7 Jason Gross, Adam Chlipala, and David I. Spivak. Experience Implementing a Performant Category-Theory Library in Coq. In *Interactive Theorem Proving – 5th International Conference, ITP 2014. Proceedings*, pages 275–291, July 2014. doi:10.1007/978-3-319-08970-6_18.

8 Jason Gross, Adam Chlipala, and David I. Spivak. Experience implementing a performant category-theory library in coq, 2014. arXiv:arXiv:1401.7694.

9 Gérard P. Huet and Amokrane Saïbi. Constructive category theory. In *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 239–276, 2000. URL: http://www.lix.polytechnique.fr/coq/pylons/coq/pylons/contribs/view/ConCaT/v8.4.

10 Bart Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam, 1999.

11 Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 1978.

12 The Coq development team. *Coq 8.5 Reference Manual*. Inria, 2015.

13 Colin McLarty. *Elementary Categories, Elementary Toposes*. Oxford University Press, Oxford, UK, 1996.

14 Adam Megacz. Category Theory in Coq. URL: http://www.megacz.com/berkeley/coq-categories/.

15 John C. Mitchell. *Foundations of Programming Languages*. MIT Press, Cambridge, MA, USA, 1996.

16 Daniel Peebles, James Deikun, Ulf Norell, Dan Doel, Andrea Vezzosi, Darius Jahandarie, and James Cook. copumpkin/categories. URL: https://github.com/copumpkin/categories.

17 Antonino Salibra. Scott is always simple. In *Proceedings of the 37th International Conference on Mathematical Foundations of Computer Science*, MFCS'12, pages 31–45, Berlin, Heidelberg, 2012. Springer-Verlag. doi:10.1007/978-3-642-32589-2_3.

18 Matthieu Sozeau and Nicolas Tabareau. Universe Polymorphism in Coq. In *Interactive Theorem Proving, ITP 2014, Proceedings*, pages 499–514, July 2014. doi:10.1007/978-3-319-08970-6_32.

19 The Univalent Foundations Program. HoTT Version of Coq and Library. URL: https://github.com/HoTT/HoTT.

20 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. http://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

21 Amin Timany. Categories. URL: https://github.com/amintimany/Categories/tree/FSCD16, doi:10.5281/zenodo.50689.

22 Amin Timany. Categories-HoTT. URL: `https://github.com/amintimany/Categories-HoTT`.

23 Amin Timany and Bart Jacobs. The Category-theoretic Solution of Recursive Ultra-metric Space Equations. Presented at CoqPL'16: The Second International Workshop on Coq for PL. URL: `https://github.com/amintimany/CTDT`.

24 Amin Timany and Bart Jacobs. First Steps Towards Cumulative Inductive Types in CIC. In *12th International Colloquium on Theoretical Aspects of Computing (ICTAC 2015) 2015. Proceedings*, pages 608–617, 2015. `doi:10.1007/978-3-319-25150-9_36`.

25 Amin Timany and Bart Jacobs. Category Theory in Coq 8.5: Extended Version. Technical Report CW697, iMinds-Distrinet, KU Leuven, April 2016. URL: `http://www2.cs.kuleuven.be/publicaties/rapporten/cw/CW697.abs.html`.

Formal Languages, Formally and Coinductively

Dmitriy Traytel

Department of Computer Science, ETH Zürich, Zürich, Switzerland
traytel@inf.ethz.ch

Abstract

Traditionally, formal languages are defined as sets of words. More recently, the alternative coalgebraic or coinductive representation as infinite tries, i.e., prefix trees branching over the alphabet, has been used to obtain compact and elegant proofs of classic results in language theory. In this paper, we study this representation in the Isabelle proof assistant. We define regular operations on infinite tries and prove the axioms of Kleene algebra for those operations. Thereby, we exercise corecursion and coinduction and confirm the coinductive view being profitable in formalizations, as it improves over the set-of-words view with respect to proof automation.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Formal languages, codatatypes, corecursion, coinduction, interactive theorem proving, Isabelle/HOL

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.31

1 Introduction

If we ask a computer scientist what a formal language is, the answer will most certainly be: a set of words. Here, we advocate another valid answer: an infinite trie. This is the coalgebraic approach to languages [24], viewed through the lens of a lazy functional programmer.

This paper shows how to formalize the coalgebraic or coinductive approach to formal languages in the Isabelle/HOL proof assistant in the form of a gentle introduction to corecursion and coinduction. Our interest in the coalgebraic approach to formal languages arose in the context of a larger formalization effort of coalgebraic decision procedures for regular languages [28, 30]. Indeed, we present here a reusable library modeling languages, which lies at the core of those formalized decision procedures. A lesson we have learned from this exercise and hope to convey here is that often it is worthwhile to look at well-understood objects from a different (in this case coinductive) perspective.

When programming with infinite structures in the total setting of a proof assistant, productivity must be ensured. Intuitively, a corecursive function is productive if it always eventually yields observable output, e.g., in form of a constructor. Functions that output exactly one constructor before proceeding corecursively or stopping with a fixed (non-corecursive) value are called primitively corecursive – a fragment dual to well-understood primitively recursive functions on inductive datatypes. Primitively corecursive functions are productive. Currently, the only form of corecursion supported by Isabelle is primitive corecursion. While sophisticated methods involving domain, measure, and category theory for handling more complex corecursive specifications have been proposed [16, 4], we explore here how far primitive corecursion can get us. Restricting ourselves to this fragment is beneficial in several ways. First, our constructions become mostly Isabelle independent, since primitive corecursion is supported by all coinduction-friendly proof assistants. Second, when working in the restricted setting, we quickly hit and learn to understand the limits. In fact, we will face some non-primitively corecursive specifications on infinite tries, which we reduce to a

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 31; pp. 31:1–31:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

composition of primitively corecursive specifications. Those reductions are insightful and hint at a general pattern for handling certain non-primitively corecursive specifications.

Infinite data structures are often characterized in terms of observations. For infinite tries, which we define as a coinductive datatype or short codatatype (Section 2), we can observe the root, which in our case is labeled by a Boolean value. This label determines if the empty word is *accepted* by the trie. Moreover, we can observe the immediate subtrees of a trie, of which we have one for each alphabet letter. This observation corresponds to making transitions in an automaton or rather computing the left quotient $L_a = \{w \mid aw \in L\}$ of the language L by the letter a. Indeed, we will see that Brzozowski's ingenious derivative operation [7], which mimics this computation recursively on the syntax of regular expressions, arises very naturally when defining regular operations corecursively on tries (Section 3). To validate our definitions, we prove by coinduction that they satisfy the axioms of Kleene algebra (Section 4). After having presented our formalization, we step back and connect concrete intuitive notions (such as tries) with abstract coalgebraic terminology (Section 5). Furthermore, we discuss our formalization and its relation to other work on corecursion and coinduction with or without proof assistants (Section 6).

The material presented in this paper is based on the publicly available Isabelle/HOL formalization [27] and is partly described in the author's Ph.D. thesis [30].

Preliminaries

Isabelle/HOL is a proof assistant for higher-order logic, built around a small trusted inference kernel. The kernel accepts only non-recursive type and constant definitions. High-level specification mechanisms, which allow the user to enter (co)recursive specifications, reduce this input to something equivalent but non-recursive. The original (co)recursive specification is later derived as a theorem. For a comprehensive introduction to Isabelle/HOL we refer to a recent textbook [17, Part I].

In Isabelle/HOL types τ are built from type variables α, β, etc., via type constructors κ written postfix (e.g., $\alpha\ \kappa$). Some special types are the product type $\alpha \times \beta$ and the function type $\alpha \to \beta$, for which the type constructors are written infix. Infix operators bind less tightly than the postfix or prefix ones. Other important types are the type of Booleans *bool* inhabited by exactly two different values $\top$ (truth) and $\bot$ (falsity) and the types $\alpha\ \mathit{list}$ and $\alpha\ \mathit{set}$ of lists and sets of elements of type α. For Boolean connectives and sets common mathematical notation is used. A special constant is equality $= :: \alpha \to \alpha \to \mathit{bool}$, which is polymorphic (it exists for any type, including the function type, on which it is extensional, i.e., $(\forall x.\ f\ x = g\ x) \longrightarrow f = g)$. Lists are constructed from $[] :: \alpha\ \mathit{list}$ and $\# :: \alpha \to \alpha\ \mathit{list} \to \alpha\ \mathit{list}$; the latter written infix and often omitted, i.e., we write aw for $a \# w$. The notation $|w|$ stands for the length of the list w, i.e., $|[]| = 0$ and $|aw| = 1 + |w|$.

2 Languages as Infinite Tries

We define the type of formal languages as a codatatype of infinite tries, that is, (prefix) trees of infinite depth branching over the alphabet. We represent the alphabet by the type parameter α. Each node in a trie carries a Boolean label, which indicates whether the (finite) path to this node constitutes a word inside or outside of the language. The function type models branching: for each letter $x :: \alpha$ there is a subtree, which we call x*-subtree*.

$$\texttt{codatatype}\ \alpha\ \mathit{lang} = \mathsf{L}\ (o : \mathit{bool})\ (\delta : \alpha \to \alpha\ \mathit{lang})$$

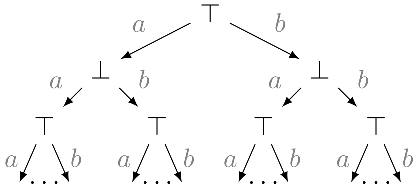


Figure 1 Infinite trie *even*.

The `codatatype` command defines the type $\alpha\,\mathit{lang}$ together with a *constructor* $\mathsf{L} :: \mathit{bool} \to (\alpha \to \alpha\,\mathit{lang}) \to \alpha\,\mathit{lang}$ and two *selectors* $o :: \alpha\,\mathit{lang} \to \mathit{bool}$ and $\delta :: \alpha\,\mathit{lang} \to \alpha \to \alpha\,\mathit{lang}$ For a binary alphabet $\alpha = \{a, b\}$, the trie *even* shown in Figure 1 is an inhabitant of $\alpha\,\mathit{lang}$. The label of its root is given by $o\ \mathit{even} = \top$ and its subtrees by another trie $\mathit{odd} = \delta\ \mathit{even}\ a = \delta\ \mathit{even}\ b$. Similarly, we have $o\ \mathit{odd} = \bot$ and $\mathit{even} = \delta\ \mathit{odd}\ a = \delta\ \mathit{odd}\ b$. Note that we could have equally written $\mathit{even} = \mathsf{L}\ \top\ (\lambda\_.\ \mathit{odd})$ and $\mathit{odd} = \mathsf{L}\ \bot\ (\lambda\_.\ \mathit{even})$ to obtain the same mutual characterization of *even* and *odd*.

We gave our type the name $\alpha\,\mathit{lang}$, to remind us to think of its inhabitants as formal languages. In the following, we use the terms language and trie synonymously.

Beyond defining the type and the constants, the `codatatype` command also exports a wealth of properties about them such as $o\,(\mathsf{L}\ b\ d) = b$, the injectivity of L, or more interestingly the coinduction rule. Informally, coinduction allows us to prove equality of tries which cannot be distinguished by finitely many selector applications.

Clearly, we would like to identify the trie *even* with the regular language of all words of even length $\{w \in \{a, b\}^* \mid |w|\ \mathsf{mod}\ 2 = 0\}$, also represented by the regular expression $((a + b) \cdot (a + b))^*$. Therefore, we define the notion of word membership $\Subset$ on tries by primitive (or structural) recursion on the word using Isabelle's `primrec` command.

```
primrec ⋐ :: α list → α lang → bool where
    [] ⋐ L = o L
    aw ⋐ L = w ⋐ δ L a
```

Using $\Subset$, each trie can be assigned a language in the traditional set of lists view.

```
definition out ::  α lang → α list set where
    out L = {w | w ⋐ L}
```

With this definition, we obtain $\mathsf{out}\ \mathit{even} = \{w \in \{a, b\}^* \mid |w|\ \mathsf{mod}\ 2 = 0\}$.

3 Regular Operations on Tries

So far, we have only specified some concrete infinite tries informally. Formally, we will use primitive corecursion, which is dual to primitive recursion. Primitively recursive functions consume one constructor before proceeding recursively. Primitively corecursive functions produce one *guarding* constructor whose arguments are allowed to be either non-recursive terms or a corecursive call (applied to arbitrary non-recursive arguments).The `primcorec` command reduces a primitively corecursive specification to a non-recursive definition, which is accepted by Isabelle's inference kernel [3]. Internally, the reduction employs a dedicated combinator for primitive corecursion on tries generated by the `codatatype` command. The `primcorec` command slightly relaxes the above restriction of primitive corecursion by allowing syntactic conveniences, such as lambda abstractions, `case`-, and `if`-expressions, to appear between the guarding constructor and the corecursive call.

3.1 Primitively Corecursive Operations

We start with some simple examples: the languages of the base cases of regular expressions. Intuitively, the trie $\varnothing$ representing the empty language is labeled with $\bot$ everywhere and the trie ε representing the empty word language is labeled with $\top$ at its root and with $\bot$ everywhere else. The trie $\mathsf{A}\ a$ representing the singleton language of the one letter word a is labeled with $\bot$ everywhere except for the root of its a-subtree. This intuition is easy to capture formally.

$$\begin{array}{ll} \texttt{primcorec}\ \varnothing :: \alpha\ \mathit{lang}\ \texttt{where} & \qquad \texttt{primcorec}\ \varepsilon :: \alpha\ \mathit{lang}\ \texttt{where} \\ \quad \varnothing = \mathsf{L}\ \bot\ (\lambda x.\ \varnothing) & \qquad \quad \varepsilon = \mathsf{L}\ \top\ (\lambda x.\ \varnothing) \end{array}$$

$$\begin{array}{l} \texttt{primcorec}\ \mathsf{A} :: \alpha \to \alpha\ \mathit{lang}\ \texttt{where} \\ \quad o\ (\mathsf{A}\ a) = \bot \\ \quad \delta\ (\mathsf{A}\ a) = \lambda x.\ \texttt{if}\ a = x\ \texttt{then}\ \varepsilon\ \texttt{else}\ \varnothing \end{array}$$

Among these three definitions only $\varnothing$ is truly corecursive.

The specifications of $\varnothing$ and ε differ syntactically from the one of A. The constants $\varnothing$ and ε are defined using the so called *constructor view*. The constructor view allows the user to enter equations of the form constant or function equals constructor, where the arguments of the constructor are restricted as described above. Such definitions should be familiar to any (lazy) functional programmer.

In contrast, the specification of A is expressed in the *destructor view*. Here, we specify the constant or function being defined by observations or experiments via *selector equations*. The allowed experiments on a trie are given by its selectors o and δ. We can observe the label at the root and the subtrees. Specifying the observation for each selector – again restricted either to be a non-recursive term or to contain the corecursive call only at the outermost position (ignoring lambda abstractions, `case`-, and `if`-expressions) – yields a unique characterization of the function being defined.

It is straightforward to rewrite specifications in either of the views into the other one. The `primcorec` command performs this rewriting internally and outputs the theorems corresponding to the user's input specification in both views. The constructor view theorems serve as executable code equations. Isabelle's code generator [10] can use these equations to generate code which make sense in programming languages with lazy evaluation. In contrast, the destructor view offers safe simplification rules even when applied eagerly during rewriting as done by Isabelle's simplifier. Note that constructor view specifications such as $\varnothing = \mathsf{L}\ \bot\ (\lambda x.\ \varnothing)$ will cause the simplifier to loop when applied eagerly.

Now that the basic building blocks $\varnothing$, ε, and A are in place, we turn our attention to how to combine them to obtain more complex languages. We start with the simpler combinators for union, intersection, and complement, before moving to the more interesting concatenation and iteration. The union $+$ of two tries should denote set union of languages (i.e., $\mathsf{out}\ (L + K) = \mathsf{out}\ L \cup \mathsf{out}\ K$ should hold). It is defined corecursively by traversing the two tries in parallel and computing for each pair of labels their disjunction. Intersection $\cap$ is analogous. Complement $\overline{}$ simply inverts every label.

$$\begin{array}{l} \texttt{primcorec}\ + :: \alpha\ \mathit{lang} \to \alpha\ \mathit{lang} \to \alpha\ \mathit{lang}\ \texttt{where} \\ \quad o\ (L + K) = o\ L \vee o\ K \\ \quad \delta\ (L + K) = \lambda x.\ \delta\ L\ x + \delta\ K\ x \end{array}$$

$$\begin{array}{l} \texttt{primcorec}\ \cap :: \alpha\ \mathit{lang} \to \alpha\ \mathit{lang} \to \alpha\ \mathit{lang}\ \texttt{where} \\ \quad o\ (L \cap K) = o\ L \wedge o\ K \\ \quad \delta\ (L \cap K) = \lambda x.\ \delta\ L\ x \cap \delta\ K\ x \end{array}$$

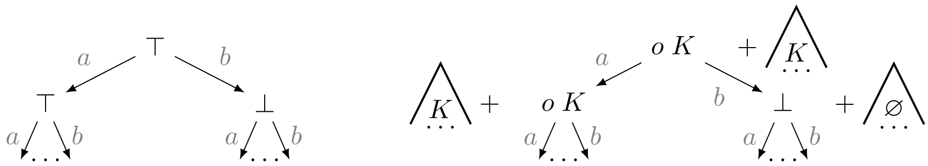


Figure 2 Tries for L (left) and the concatenation $L \cdot K$ (right).

$\texttt{primcorec}\ \overline{} :: \alpha\ \mathit{lang} \to \alpha\ \mathit{lang}\ \texttt{where}$
$\quad o\ \overline{L} = \neg\ o\ L$
$\quad \delta\ \overline{L} = \lambda x.\ \overline{\delta\ L\ x}$

Let us look at the specifying selector equations which we have seen so far from a different perspective. Imagine L and K being not tries but instead syntactic regular expressions, A, $+$, $\cap$, and $\overline{}$ constructors of a datatype for regular expressions, and o and δ two operations that we define recursively on this syntax. From that perspective, the operations are familiar: rediscovered Brzozowski derivatives of regular expressions [7] and the empty word acceptance (often also called *nullability*) test on regular expressions in the destructor view equations for the selectors δ and o. There is an important difference, though: while Brzozowski derivatives work with syntactic objects, our tries are semantic objects on which equality denotes language equivalence. For example, we will later prove $\varnothing + L = L$ for tries, whereas $\varnothing + L \neq L$ holds for regular expressions. The coinductive view reveals that derivatives and the acceptance test are the two fundamental ingredients that completely characterize regular languages and arise naturally when only considering the semantics.

3.2 Reducing Corecursion Up-to to Primitive Corecursion

Concatenation $\cdot$ is next on the list of regular operations that we want to define on tries. Thinking of Brzozowski derivatives and the acceptance test, we would usually specify it by the following two equations.

$o\ (L \cdot K) = o\ L \wedge o\ K$
$\delta\ (L \cdot K) = \lambda x.\ (\delta\ L\ x \cdot K) + (\texttt{if}\ o\ L\ \texttt{then}\ \delta\ K\ x\ \texttt{else}\ \varnothing)$

A difficulty arises here, since this specification is not primitively corecursive – the right hand side of the second equation contains a corecursive call but not at the topmost position (but rather under $+$ here). We call this kind of corecursion *up to* $+$.

Without tool support for corecursion up-to, concatenation must be defined differently – as a composition of other primitively corecursive operations. Intuitively, we would like to separate the above δ-equation into two along the $+$ and sum them up afterwards. Technically, the situation is more involved. Since the δ-equation is corecursive, we cannot just create two tries by primitive corecursion.

Figure 2 depicts the trie that should result from concatenating an arbitrary trie K to the concrete given trie L. Procedurally, the concatenation must replace every subtree T of L that has $\top$ at the root (those are positions where words from L end) by the trie $U + K$ where U is the trie obtained from T by changing its root from $\top$ to $o\ K$. For uniformity with the above δ-equation, we imagine subtrees F of L with label $\bot$ at the root as also being replaced by $F + \varnothing$, which, as we will prove later, has the same effect as leaving F alone.

Figure 3 presents one way to bypass the restrictions imposed by primitive corecursion. We are not allowed to use $+$ *after* proceeding corecursively, but we may arrange the arguments of $+$ in a broader trie over a doubled alphabet formally modeled by pairing letters of the

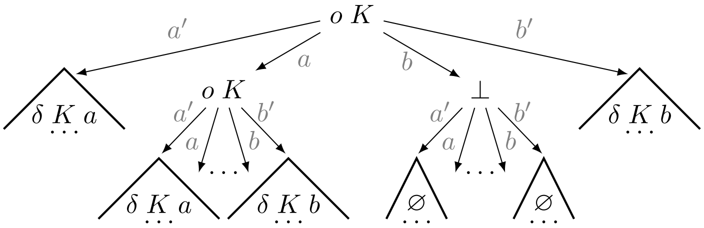


Figure 3 Trie for deferred concatenation $L \mathbin{\hat{\cdot}} K$.

alphabet with a Boolean flag. In Figure 3 we write a for $(a, \top)$ and a' for $(a, \bot)$. Because it defers the summation, we call this primitively corecursive procedure *deferred concatenation* $\hat{\cdot}$.

`primcorec` $\hat{\cdot}$ :: $\alpha\ \mathit{lang} \to \alpha\ \mathit{lang} \to (\alpha \times \mathit{bool})\ \mathit{lang}$ `where`
  $o\ (L \mathbin{\hat{\cdot}} K) = o\ L \wedge o\ K$
  $\delta\ (L \mathbin{\hat{\cdot}} K) = \lambda(x, b).$ `if` b `then` $\delta\ L\ x \mathbin{\hat{\cdot}} K$ `else if` $o\ L$ `then` $\delta\ K\ x \mathbin{\hat{\cdot}} \varepsilon$ `else` $\varnothing$

Note that unlike in the Figure 3, where we informally plug the trie $\delta\ K\ x$ as some x'-subtrees, the formal definition must be more careful with the types. More precisely, $\delta\ K\ x$ is of type $\alpha\ \mathit{lang}$, while something of type $(\alpha \times \mathit{bool})\ \mathit{lang}$ is expected. This type mismatch is resolved by further concatenating ε to $\delta\ K\ x$ (again in a deferred fashion) without corrupting the intended semantics.

Once the trie for the deferred concatenation has been built, the desired trie for concatenation can be obtained by a second primitively corecursive traversal that sums the x- and x'-subtrees *before* proceeding corecursively.

`primcorec` $\hat{\oplus}$ :: $(\alpha \times \mathit{bool})\ \mathit{lang} \to \alpha\ \mathit{lang}$ `where`
  $o\ (\hat{\oplus}\ L) = o\ L$
  $\delta\ (\hat{\oplus}\ L) = \lambda x.\ \hat{\oplus}\ (\delta\ L\ (x, \top) + \delta\ L\ (x, \bot))$

Finally, we can define the concatenation as the composition of $\hat{\cdot}$ and $\hat{\oplus}$. The earlier standard selector equations for $\cdot$ are provable for this definition.

`definition` $\cdot$ :: $\alpha\ \mathit{lang} \to \alpha\ \mathit{lang} \to \alpha\ \mathit{lang}$ `where`
  $L \cdot K = \hat{\oplus}\ (L \mathbin{\hat{\cdot}} K)$

The situation with iteration is similar. The selector equations following the Brzozowski derivative of L^* yield a non-primitively corecursive specification: it is corecursive up to $\cdot$.

$o\ (L^*) = \top$
$\delta\ (L^*) = \lambda x.\ \delta\ L\ x \cdot L^*$

As before, the restriction is circumvented by altering the operation slightly. We define the binary operation *deferred iteration* $L \mathbin{\hat{*}} K$, whose language should represent $L \cdot K^*$ (although we have not defined * yet). When constructing the subtrees of $L \mathbin{\hat{*}} K$ we keep pulling copies of the second argument into the first argument before proceeding corecursively (the second argument itself stays unchanged).

`primcorec` $\hat{*}$ :: $\alpha\ \mathit{lang} \to \alpha\ \mathit{lang} \to \alpha\ \mathit{lang}$ `where`
  $o\ (L \mathbin{\hat{*}} K) = o\ L$
  $\delta\ (L \mathbin{\hat{*}} K) = \lambda x.\ (\delta\ (L \cdot (\varepsilon + K))\ x) \mathbin{\hat{*}} K$

Supplying ε as the first argument to $\hat{*}$ defines iteration for which the original selector equations hold.

```
definition _* :: α lang → α lang where
    L* = ε ^* L
```

We have defined all the standard regular operations on tries. Later we will prove that those definitions satisfy the axioms of Kleene algebra, meaning that they behave as expected. Already now we can compose the operations to define new tries, for example the introductory $\mathit{even} = ((\mathsf{A}\ a + \mathsf{A}\ b) \cdot (\mathsf{A}\ a + \mathsf{A}\ b))^*$.

Beyond Regular Languages

Before we turn to proving, let us exercise one more corecursive definition. Earlier, we have assigned each trie a set of lists via the function out. Primitive corecursion enables us to define the converse operation.

```
primcorec in :: α list set → α lang where
    o (in L) = [] ∈ L
    δ (in L) = λa. in {w | aw ∈ L}
```

The function out and in are both bijections. We show this by proving that their compositions (either way) are both the identity function. One direction, out (in L) = L, follows by set extensionality and a subsequent induction on words. The reverse direction requires a proof by coinduction, which is the topic of the next section.

Using in we can turn every (even undecidable) set of lists into a trie. This is somewhat counterintuitive, since, given a concrete trie, its word problem seems easily decidable (via the function $\Subset$). But of course in order to compute the trie out of a set of lists L via in the word problem for L must be solved – we are reminded that higher-order logic is not a programming language where everything is executable, but a logic in which we write down specifications (and not programs). For regular operations from the previous section the situation is different. For example, Isabelle's code generator can produce valid Haskell code that evaluates $\mathit{abaa} \Subset (\mathsf{A}\ a \cdot (\mathsf{A}\ a + \mathsf{A}\ b))^*$ to $\top$.

4 Proving Equalities on Tries

We have seen the definitions of many operations, justifying their meaningfulness by appealing to the reader's intuition. This is often not enough to guarantee correctness, especially if we have a theorem prover at hand. So let us formally prove in Isabelle that the regular operations on tries form a Kleene algebra by proving Kozen's famous axioms [13] as (in)equalities on tries.

Codatatypes are equipped with the perfect tool for proving equalities: the coinduction principle. Intuitively, this principle states that the existence of a relation R that is *closed* under the codatatype's observations (given by selectors) implies that elements related by R are equal. Being closed means here that the for all R-related codatatype elements their immediate (non-corecursive) observations are equal and the corecursive observations are again related by R. In other words, if we cannot distinguish elements of a codatatype by (finite) observations, they must be equal. Formally, for our codatatype $\alpha\ \mathit{lang}$ we obtain the following coinduction rule.

$$\frac{R\ L\ K \qquad \forall L_1\ L_2.\ R\ L_1\ L_2 \longrightarrow (o\ L_1 = o\ L_2 \wedge \forall x.\ R\ (\delta\ L_1\ x)\ (\delta\ L_2\ x))}{L = K}$$

```
1   theorem ∅ + L = L
2   proof (rule coinduct_lang)
3       def R L₁ L₂ = (∃K. L₁ = ∅ + K ∧ L₂ = K)

4       show R (∅ + L) L by simp

5       fix L₁ L₂
6       assume R L₁ L₂
7       then obtain K where L₁ = ∅ + K and L₂ = K by simp
8       then show o L₁ = o L₂ ∧ ∀x. R (δ L₁ x) (δ L₂ x) by simp
9   qed
```

Figure 4 A detailed proof by coinduction.

The second assumption of this rule formalizes the notion of being closed under observations: If two tries are related then their root's labels are the same and all their subtrees are related. A relation that satisfies this assumption is called a *bisimulation*. Thus, proving an equation by coinduction amounts to exhibiting a bisimulation witness that relates the left and the right hand sides.

Let us observe the coinduction rule, which we call $\mathit{coinduct}_{lang}$, in action. Figure 4 shows a detailed proof of the Kleene algebra axiom that the empty language is the left identity of union that is accepted by Isabelle.

After applying the coinduction rule backwards (line 2), the proof has three parts. First, we supply a definition of a witness relation R (line 3). Second, we prove that R relates the left and the right hand side (line 4). Third, we prove that R is a bisimulation (lines 5–8). Proving $R\ (\varnothing + L)\ L$ for our particular definition of R is trivial after instantiating the existentially quantified K with L. Proving the bisimulation property is barely harder: for two tries L_1 and L_2 related by R we need to show $o\ L_1 = o\ L_2$ and $\forall x.\ R\ (\delta\ L_1\ x)\ (\delta\ L_2\ x)$. Both properties follow by simple calculations rewriting L_1 and L_2 in terms of a trie K (line 7), whose existence is guaranteed by $R\ L_1\ L_2$, and simplifying with the selector equations for $+$ and $\varnothing$.

$$o\ L_1 = o\ (\varnothing + K) = (o\ \varnothing \lor o\ K) = (\bot \lor o\ K) = o\ K = o\ L_2$$

$$\begin{aligned} & R\ (\delta\ L_1\ x)\ (\delta\ L_2\ x) = R\ (\delta\ (\varnothing + K)\ x)\ (\delta\ K\ x) \\ & = R\ (\delta\ \varnothing\ x + \delta\ K\ x)\ (\delta\ K\ x) = R\ (\varnothing + \delta\ K\ x)\ (\delta\ K\ x) \\ & = (\exists K'.\ \varnothing + \delta\ K\ x = \varnothing + K' \land \delta\ K\ x = K') = \top \end{aligned}$$

The last step is justified by instantiating K' with $\delta\ K\ x$.

So in the end, the only part that required ingenuity was the definition of the witness R. But was it truly ingenious? It turns out that in general, when proving a conditional equation $P\ \overline{x} \longrightarrow l\ \overline{x} = r\ \overline{x}$ by coinduction, where $\overline{x}$ denotes a list of variables occurring freely in the equation, the canonical choice for the bisimulation witness is $R\ a\ b = (\exists \overline{x}.\ a = l\ \overline{x} \land b = r\ \overline{x} \land P\ \overline{x})$. There is no guarantee that this definition yields a bisimulation. However, after performing a few proofs by coinduction, this particular pattern emerges as a natural first choice to try. Indeed, the choice is so natural, that it was worth to automate it in the *coinduction* proof method [3]. This method instantiates the coinduction rule $\mathit{coinduct}_{lang}$ with the canonical bisimulation witness constructed from the goal, where the existentially quantified variables must be given explicitly using the *arbitrary* modifier. Then it applies the instantiated rule in a resolution step, discharges the first assumption, and unpacks the existential quantifiers from R in the remaining subgoal (the `obtain` step in the above proof). Many proofs collapse to an automatic one-line proof using this convenience, including the one above. Some examples follow.

`theorem` $\varnothing + L = L$ `by` $(\mathit{coinduction\ arbitrary}\colon L)\ \mathit{auto}$

`theorem` $L + L = L$ `by` $(\mathit{coinduction\ arbitrary}\colon L)\ \mathit{auto}$

`theorem` $L_1 + L_2 = L_2 + L_1$ `by` $(\mathit{coinduction\ arbitrary}\colon L_1\ L_2)\ \mathit{auto}$

`theorem` $(L_1 + L_2) + L_3 = L_1 + L_2 + L_3$ `by` $(\mathit{coinduction\ arbitrary}\colon L_1\ L_2\ L_3)\ \mathit{auto}$

`theorem` $\mathsf{in}\ (\mathsf{out}\ L) = L$ `by` $(\mathit{coinduction\ arbitrary}\colon L)\ \mathit{auto}$

`theorem` $\mathsf{in}\ (L \cup K) = \mathsf{in}\ L + \mathsf{in}\ K$ `by` $(\mathit{coinduction\ arbitrary}\colon L\ K)\ \mathit{auto}$

As indicated earlier, sometimes the *coinduction* method does not succeed. It is instructive to consider one example where this is the case: $o\ L \longrightarrow \varepsilon + L = L$.

If we attempt to prove the above statement by coinduction instantiated with the canonical bisimulation witness $R\ L_1\ L_2 = (\exists K.\ L_1 = \varepsilon + K\ \wedge\ L_2 = K\ \wedge\ o\ K)$, after some simplification we are stuck with proving that R is a bisimulation.

$$\begin{aligned}
&R\ (\delta\ L_1\ x)\ (\delta\ L_2\ x) = R\ (\delta\ (\varepsilon + K)\ x)\ (\delta\ K\ x)\\
&= R\ (\delta\ \varepsilon\ x + \delta\ K\ x)\ (\delta\ K\ x) = R\ (\varnothing + \delta\ K\ x)\ (\delta\ K\ x)\\
&= R\ (\delta\ K\ x)\ (\delta\ K\ x) = \exists K'.\ \delta\ K\ x = \varepsilon + K' \wedge \delta\ K\ x = K' \wedge o\ K'
\end{aligned}$$

The remaining goal is not provable: we would need to instantiate K' with $\delta\ K\ x$, but then, we are left to prove $o\ (\delta\ K\ x)$, which we cannot deduce (we only know $o\ K$). If we, however, instantiate the coinduction rule with $R^{=}\ L_1\ L_2 = R\ L_1\ L_2 \vee L_1 = L_2$, we are able to finish the proof. This means that our canonical bisimulation witness R is not a bisimulation, but $R^{=}$ is. In such cases R is called a *bisimulation up to equality*.

Instead of modifying the *coinduction* method to instantiate the rule $\mathit{coinduct}_{\mathit{lang}}$ with $R^{=}$, it is more convenient to capture this improvement directly in the coinduction rule. This results in the following rule which we call *coinduction up to equality* or $\mathit{coinduct}^{=}_{\mathit{lang}}$.

$$\frac{R\ L\ K \qquad \forall L_1\ L_2.\ R\ L_1\ L_2 \longrightarrow (o\ L_1 = o\ L_2 \wedge \forall x.\ R^{=}\ (\delta\ L_1\ x)\ (\delta\ L_2\ x))}{L = K}$$

Note that $\mathit{coinduct}^{=}_{\mathit{lang}}$ is not just an instance of $\mathit{coinduct}_{\mathit{lang}}$, with R replaced by $R^{=}$. Instead, after performing this replacement, the first assumption is further simplified to $R\ L\ K$ – we would not use coinduction in the first place, if we could prove $R^{=}\ L\ K$ by reflexivity. Similarly, the reflexivity part in the occurrence on the left of the implication in the second assumption is needless and therefore eliminated. The resulting coinduction up to equality principles are independent of the particular codatatypes and thus uniformly produced by the `codatatype` command. Using this coinduction up to equality rule, we have regained the ability to write one-line proofs.

`theorem` $o\ L \longrightarrow \varepsilon + L = L$ `by` $(\mathit{coinduction\ arbitrary}\colon L\ \mathit{rule}\colon \mathit{coinduct}^{=}_{\mathit{lang}})\ \mathit{auto}$

This brings us to the next hurdle. Suppose that we already have proved the standard selector equations for concatenation $\cdot$. (This requires finding some auxiliary properties of $\hat{\ }$ and $\hat{\oplus}$ but is an overall straightforward usage of coinduction up to equality.) Next, we want to reason about $\cdot$. For example, we prove its distributivity over $+$: $(L + K) \cdot M = (L \cdot M) + (K \cdot M)$. As before, we are stuck proving that the canonical bisimulation witness $R\ L_1\ L_2 = (\exists L'\ K'\ M'.\ L_1 = (L' + K') \cdot M' \wedge L_2 = (L' \cdot M') + (K' \cdot M'))$ is a bisimulation (and this time even for up to equality).

$$
\begin{aligned}
&R^{=}\,(\delta\ L_1\ x)\ (\delta\ L_2\ x) = R^{=}\,(\delta\ ((L' + K')\cdot M')\ x)\ (\delta\ ((L'\cdot M') + (K'\cdot M'))\ x)\\
&= \begin{cases}
R^{=}\,((\delta\ L'\ x + \delta\ K'\ x)\cdot M') & \\
\quad ((\delta\ L'\ x\cdot M') + (\delta\ K'\ x\cdot M')) & \texttt{if}\ \neg o\ L' \wedge \neg o\ K'\\
R^{=}\,((\delta\ L'\ x + \delta\ K'\ x)\cdot M' + \delta\ M'\ x) & \\
\quad ((\delta\ L'\ x\cdot M' + \delta\ M'\ x) + (\delta\ K'\ x\cdot M')) & \texttt{if}\ o\ L' \wedge \neg o\ K'\\
R^{=}\,((\delta\ L'\ x + \delta\ K'\ x)\cdot M' + \delta\ M'\ x) & \\
\quad ((\delta\ L'\ x\cdot M') + (\delta\ K'\ x\cdot M' + \delta\ M'\ x)) & \texttt{if}\ \neg o\ L' \wedge o\ K'\\
R^{=}\,((\delta\ L'\ x + \delta\ K'\ x)\cdot M' + \delta\ M'\ x) & \\
\quad ((\delta\ L'\ x\cdot M' + \delta\ M'\ x) + (\delta\ K'\ x\cdot M' + \delta\ M'\ x)) & \texttt{if}\ o\ L' \wedge o\ K'
\end{cases}\\
&= \begin{cases}
\top & \texttt{if}\ \neg o\ L' \wedge \neg o\ K'\\
R^{=}\,((\delta\ L'\ x + \delta\ K'\ x)\cdot M'\ \colorbox{lightgray}{$+$}\ \delta\ M'\ x) & \\
\quad ((\delta\ L'\ x\cdot M' + \delta\ K'\ x\cdot M')\ \colorbox{lightgray}{$+$}\ \delta\ M'\ x) & \text{otherwise}
\end{cases}
\end{aligned}
$$

The remaining subgoal cannot be discharged by the definition of R. In principle it could be discharged by equality (the two tries are equal), but this is almost exactly the property we originally started proving, so we have not simplified the problem by coinduction but rather are going in circles here. However, if our relation could somehow split its arguments across the outermost $+$ highlighted in gray, we could prove the left pair being related by R and the right pair by $=$. The solution is easy: we allow the relation to do just that. Accordingly, we define the congruence closure R^{+} of a relation R under $+$ inductively by the following rules.

$$
\frac{L = K}{R^{+}\,L\ K}\qquad
\frac{R\ L\ K}{R^{+}\,L\ K}\qquad
\frac{R^{+}\,L\ K}{R^{+}\,K\ L}\qquad
\frac{R^{+}\,L_1\ L_2\quad R^{+}\,L_2\ L_3}{R^{+}\,L_1\ L_3}\qquad
\frac{R^{+}\,L_1\ K_1\quad R^{+}\,L_2\ K_2}{R^{+}\,(L_1 + L_2)\ (K_1 + K_2)}
$$

The closure R^{+} is then used to strengthen the coinduction rule, just as the earlier reflexive closure $R^{=}$ strengthening. Note that the closure $R^{=}$ can also be viewed as inductively generated by the first two of the above rules. In summary, we obtain *coinduction up to congruence of* $+$, denoted by $\mathit{coinduct}^{+}_{\mathit{lang}}$.

$$
\frac{R\ L\ K\quad \forall L_1\ L_2.\ R\ L_1\ L_2 \longrightarrow (o\ L_1 = o\ L_2 \wedge \forall x.\ R^{+}\,(\delta\ L_1\ x)\ (\delta\ L_2\ x))}{L = K}
$$

This rule is easily derived from plain coinduction by instantiating R in $\mathit{coinduct}_{\mathit{lang}}$ with R^{+} and proceeding by induction on the definition of the congruence closure.

As intended $\mathit{coinduct}^{+}_{\mathit{lang}}$ makes the proof of distributivity into another automatic one-liner. This is because our new proof principle, coinduction up to congruence of $+$, matches exactly the definitional principle of corecursion up to $+$ used in the selector equations of $\cdot$.

$\texttt{theorem}\ (L + K)\cdot M = (L\cdot M) + (K\cdot M)$
$\quad\texttt{by}\ (\mathit{coinduction\ arbitrary}\colon L\ K\ M\ \mathit{rule}\colon \mathit{coinduct}^{+}_{\mathit{lang}})\ \mathit{auto}$

For properties involving iteration *, whose selector equations are corecursive up to $\cdot$, we will need a further strengthening of the coinduction rule (using the congruence closure under $\cdot$). Overall, the most robust solution to keep track of the different rules is to maintain a coinduction rule up to all previously defined operations on tries: we define $R^{\bullet}$ to be the congruence closure of R under $+$, $\cdot$, and * and then use the following rule for proving.

$$
\frac{R\ L\ K\quad \forall L_1\ L_2.\ R\ L_1\ L_2 \longrightarrow (o\ L_1 = o\ L_2 \wedge \forall x.\ R^{\bullet}\,(\delta\ L_1\ x)\ (\delta\ L_2\ x))}{L = K}
$$

$$\begin{tikzcd}
\alpha \arrow[r, "s"] \arrow[d, "f"'] & \alpha\ F \arrow[d, "\mathsf{map}_F\ f"] \\
\beta \arrow[r, "t"] & \beta\ F
\end{tikzcd}$$

Figure 5 Commutation property of a coalgebra morphism.

We will not spell out all axioms of Kleene algebra [13] and their formal proofs [27] here. Most proofs are automatic; some require a few manual hints in which order to apply the congruence rules. Note that the axioms also contain some inequalities, such as $\varepsilon + L \cdot L^* \leq L^*$, and even conditional inequalities, such as $L + M \cdot K \leq M \longrightarrow L \cdot K^* \leq M$. However, $L \leq K$ is defined as $L + K = K$, such that proofs by coinduction also are applicable here.

5 Coalgebraic Foundations

We briefly connect the formalized but still intuitive notions, such as tries, from earlier sections with the key coalgebraic concepts and terminology that is usually used to present the coalgebraic view on formal languages. Thereby, we explain how particularly useful abstract objects gave rise to concrete tools in Isabelle/HOL. More theoretical and detailed introductions to coalgebra can be found elsewhere [23, 12].

Given a functor F an $(F\text{-})$*coalgebra* is a *carrier* object A together with a map $A \rightarrow F\ A$ – the *structural map* of a coalgebra. In the context of higher-order logic – that is in the category of types which consists of types as objects and of functions between types as arrows – a functor is a type constructor F together with a map function $\mathsf{map}_F :: (\alpha \rightarrow \beta) \rightarrow \alpha\ F \rightarrow \beta\ F$ that preserves identity and composition: $\mathsf{map}_F\ \mathsf{id} = \mathsf{id}$ and $\mathsf{map}_F\ (f \circ g) = \mathsf{map}_F\ f \circ \mathsf{map}_F\ g$. An F-coalgebra in HOL is therefore simply a function $s :: \alpha \rightarrow \alpha\ F$. A function $f :: \alpha \rightarrow \beta$ is a coalgebra *morphism* between two coalgebras $s :: \alpha \rightarrow \alpha\ F$ and $t :: \beta \rightarrow \beta\ F$ if it satisfies the commutation property $t \circ f = \mathsf{map}_F\ f \circ s$, also depicted by the commutative diagram in Figure 5.

An $(F\text{-})$coalgebra to which there exists an unique morphism from any other coalgebra is called a *final* $(F\text{-})$*coalgebra*. Not all functors F admit a final coalgebra. Two different final coalgebras are necessarily isomorphic. Final coalgebras correspond to codatatypes in Isabelle/HOL. Isabelle's facility for defining codatatypes maintains a large class of functors – bounded natural functors [29] – for which a final coalgebra does exists. Moreover, for any bounded natural functor F, Isabelle can construct its final coalgebra with the codatatype CF as the carrier and define a bijective constructor $\mathsf{C}_F :: CF\ F \rightarrow CF$ and its inverse, the destructor $\mathsf{D}_F :: CF \rightarrow CF\ F$. The latter takes the role of the structural map of the coalgebra.

`codatatype` $CF = \mathsf{C}_F\ (\mathsf{D}_F : CF\ F)$

Finally, we are ready to connect these abstract notions to our tries. The codatatype of tries $\alpha\ lang$ is the final coalgebra of the functor $\beta\ D = bool \times (\alpha \rightarrow \beta)$ with the associated map function $\mathsf{map}_D\ g = \mathsf{id} \times (\lambda f.\ g \circ f)$, where $(f \times g)\ (x,\ y) = (f\ x,\ g\ y)$. The structural map of this final coalgebra is the function $\mathsf{D}_D = \langle o,\ \delta\rangle$, where $\langle f,\ g\rangle\ x = (f\ x, g\ x)$.

The finality of $\alpha\ lang$ gives rise to the definitional principles of primitive coiteration and corecursion. In Isabelle the coiteration principle is embodied by the primitive *coiterator* $\mathsf{coiter} :: (\tau \rightarrow \tau\ D) \rightarrow \tau \rightarrow \alpha\ lang$, that assigns to the given D-coalgebra the unique morphism from itself to the final coalgebra. In other words, the primitive coiterator allows us to define functions of type $\tau \rightarrow \alpha\ lang$ by providing a D-coalgebra on τ, i.e., a function

$$\begin{tikzcd}
\tau \arrow[r, "s"] \arrow[d, "\mathsf{coiter}\ s"'] & \tau\ D \arrow[d, "\mathsf{map}_D\ (\mathsf{coiter}\ s)"] \\
\alpha\ lang \arrow[r, "\langle o,\, \delta\rangle"] & \alpha\ lang\ D
\end{tikzcd}$$

Figure 6 Unique morphism $\mathsf{coiter}\ s$ to the final coalgebra $(\alpha\, lang,\ \langle o,\ \delta\rangle)$.

$$\begin{tikzcd}
\tau \arrow[r, "s"] \arrow[d, "\mathsf{corec}\ s"'] & (\alpha\ lang + \tau)\ D \arrow[d, "\mathsf{map}_D\ [\mathsf{id},\, \mathsf{corec}\ s]"] \\
\alpha\ lang \arrow[r, "\langle o,\, \delta\rangle"] & \alpha\ lang\ D
\end{tikzcd}$$

Figure 7 Characteristic theorem of the corecursor.

of type $\tau \to bool \times (\alpha \to \tau)$ that essentially describes a deterministic (not necessarily finite) automaton without an initial state. To clarify this automaton analogy, it is customary to present the F-coalgebra s as two functions $s = \langle \mathsf{o},\ \mathsf{d}\rangle$ with τ being the states of the automaton, $\mathsf{o} : \tau \to bool$ denoting accepting states, and $\mathsf{d} : \alpha \to \tau \to \tau$ being the transition function. From a given s, we uniquely obtain the function $\mathsf{coiter}\ s$ that assigns to a separately given initial state $t : \tau$ the language $\mathsf{coiter}\ s\ t : \alpha\, lang$ and makes the diagram in Figure 6 commute. Note that Figure 6 is an instance of Figure 5.

Corecursion differs from coiteration by additionally allowing the user to stop the coiteration process by providing a fixed non-corecursive value. In Isabelle this is mirrored by another combinator: the *corecursor* $\mathsf{corec} :: (\tau \to (\alpha\, lang + \tau)\ D) \to \tau \to \alpha\, lang$ where the sum type $+$ offers the possibility either to continue corecursively as before (represented by the type τ) or to stop with a fixed value of type $\alpha\, lang$. The corecursor satisfies the characteristic property shown in Figure 7, where the square brackets denote a case distinction on $+$, i.e. $[f,\ g]\ x = \mathtt{case}\ x\ \mathtt{of}\ \mathsf{Inl}\ l \Rightarrow f\ l \mid \mathsf{Inr}\ r \Rightarrow g\ r$ and Inl and Inr are the standard embeddings of $+$. Corecursion is not more expressive than coiteration (since corec can be defined in terms of coiter), but it is more convenient to use. For instance, the non-corecursive specifications of ε and A, and the `else` branch of $\hat{\cdot}$ exploit this additional flexibility.

The `primcorec` command [3] reduces a user given specification to a non-recursive definition using the corecursor. For example, the union operation $+$ is internally defined as $\lambda L\ K.\ \mathsf{corec}\ (\lambda(L,\ K).\ (o\ L \wedge o\ K,\ \lambda a.\ \mathsf{Inr}\ (\delta\ L\ a,\ \delta\ K\ a)))\ (L,\ K)$. The D-coalgebra given as the argument to corec resembles the right hand sides of the selector equations for $+$ (with the corecursive calls replaced by Inr). As end users, most of the time we are happy being able to write the high-level corecursive specifications, without having to explicitly supply coalgebras.

It is worth noting that the final coalgebra $\alpha\, lang$ itself corresponds to the automaton, whose states are languages, acceptance is given by $\mathsf{o}\ L = o\ L$, and the transition function by $\mathsf{d}\ a\ L = \delta\ L\ a$. For these definitions, we obtain $\mathsf{corec}\ \langle \mathsf{o},\ \mathsf{d}\rangle\ (L : \alpha\, lang) = L$. For regular languages this automaton corresponds to the minimal automaton (since equality on tries corresponds to language equivalence), which is finite by the Myhill–Nerode theorem. This correspondence is not very practical though, since we typically label states of automata with something finite, in particular not with languages (represented by infinite tries).

A second consequence of the finality of $\alpha\, lang$ is the coinduction principle that we have seen earlier. It follows from the fact that final coalgebras are quotients by bisimilarity, where bisimilarity is defined as the existence of a bisimulation relation.

6 Discussion and Related Work

Our development is a formalized counterpart of Rutten's introduction to the coalgebraic view on languages [24]. In this section we discuss further related work.

Adding Further Operations

With the coinductive representation adding new operations corresponds to defining a new corecursive function on tries. Compared with adding a new constructor to the inductive datatype of regular expressions and extending all previously defined recursive functions on regular expressions to account for this new case, this a is rather low-cost library extension. Wadler called this tension between extending syntactic and semantic objects the Expression Problem [31]. Note that codatatypes alone are not the solution to the Expression Problem – they just populate the other side of the spectrum with respect to datatypes. In fact, adding new selectors to our tries would be as painful as adding new constructors to the datatype of regular expressions. Rendel et al. [20] outline how automatic conversions between the inductive and the coinductive view can help solving the Expression Problem.

We have extended our library with the regular shuffle product operation on languages and an operation that transforms a context free grammar in weak Greibach normal form into a trie with the same language [27]. Both operations are corecursive up to $+$ (just as $\cdot$ is).

Coalgebraic View on Formal Languages

The coalgebraic approach to languages has recently received some attention. Landmark results in language theory were rediscovered and generalized. Silva's recent survey [26] highlights some of those results including the proofs of correctness of Brzozowski's subtle deterministic finite automaton minimization algorithm [5]. The coalgebraic approach yields some algorithmic advantages, too. Bonchi and Pous present a coinductive algorithm for checking equivalence of non-deterministic automata that outperforms all previously known algorithms by one order of magnitude [6]. Another recent development is our formally verified coalgebraic algorithm for deciding weak monadic second-order logic of one successor (WS1S) [28]. This formalization employs the Isabelle library presented here.

Tutorials on Coinduction

The literature is abound with tutorials on coinduction. So why bother writing yet another one? First, because we finally can do it in Isabelle/HOL, which became a coinduction-friendly proof assistant recently [3]. Earlier studies of coinduction in Isabelle had to engage in tedious manual constructions just to define a codatatype [18]. Second, coinductive techniques are still not as widespread as they could be (and we believe should be, since they constitute a convenient proof tool for questions about semantics). Third, many tutorials [12, 14, 11, 8, 9, 25], with or without a theorem prover, exercise streams to the extent that one starts to believe having seen every single stream example one can imagine. In contrast, Rutten [24] demonstrates that it is entirely feasible to start a tutorial with a structure slightly more complicated than streams, but familiar to everybody with a computer science degree. Moreover, Rot, Bonsangue, and Rutten [21, 22] present an accessible introduction to coinduction up to congruence using the coinductive view of formal languages similarly to our Section 4. Our work additionally focuses on corecursion up-to and puts Rutten's exposition in the context of a proof assistant.

Non-Primitive Corecursion in Proof Assistants

Automation for corecursion in proof assistants is much less developed than its recursive counterpart. Currently, Isabelle/HOL provides only a command to handle primitively corecursive specifications. The Coq proof assistant can do slightly more: it supports corecursion up to constructors [8]. Looking at our examples, however, this means that Coq will not be able to prove productivity of the natural concatenation and iteration specifications automatically, since both go beyond up-to constructors. Instead, our reduction to primitive corecursion can be employed to bypass Coq's productivity checker.

Agda's combination of copatterns (i.e., destructor view) and sized types [2, 1] is the most advanced implemented support for corecursion in proof assistants to date. However, using sized types often means that one has to encode proofs of productivity manually in the type of the defined function. Thus, it should be possible to define concatenation and iteration using their natural specifications in Agda when we accept the need for heavier type annotations.

Recently, we proposed a general framework for reducing corecursion up to so called friendly operations to primitive corecursion in Isabelle/HOL [4]. An operation is friendly if, under lazy evaluation, it does not peek too deeply into its arguments, before producing at least one constructor. For example, the friendly operation $L + K = \mathsf{L}\ (o\ L \lor o\ K)\ (\lambda x.\ \delta\ L\ x + \delta\ K\ x)$ destructs only one layer of constructors, in order to produce the topmost L. Since $+$ is friendly, and $\cdot$ is corecursive up to $+$, using this framework will allow us to use the natural specifications for $\cdot$ without changing any types. (The same applies for *.) In contrast, the primitively corecursive equation $\mathsf{deep}\ L = \mathsf{L}\ (o\ L)\ (\lambda x.\ \mathsf{deep}\ (\delta\ (\delta\ L\ x)\ x))$ destructs two layers of constructors (via δ) before producing one and is therefore not friendly. Indeed, we will not be able to reduce the equation $\mathsf{bad} = \mathsf{L}\ \top\ (\lambda\_.\ \mathsf{deep}\ \mathsf{bad})$ (which is corecursive up to deep) to a primitively corecursive specification. And there is a reason for it: bad is not uniquely specified by the above equation, or in other words not productive.

Support for friendly operations will advance Isabelle over its competitors, once fully implemented. To achieve the reduction to primitive corecursion the framework follows an abstract, category theory inspired construction. Yet, what this reduction yields in practice is relatively close to our manual construction for concatenation. (In contrast, the iteration case takes some shortcuts, which the abstract view does not offer.)

Formal Languages in Proof Assistants

Such a basic thing as the traditional set-of-words view on formal languages is formalized in most proof assistants. In contrast, we are not aware of any other formalization of the coalgebraic view on formal languages in a proof assistant.

Here, we want to compare our formalization with the Isabelle incarnation of the set-of-words view developed by Krauss and Nipkow for the correctness proof of their regular expression equivalence checker [15]. Both libraries are comparably concise. In 500 lines Krauss and Nipkow prove almost all axioms of Kleene algebra and the characteristic equations for the left quotients (the δ-specifications in our case). They reuse Isabelle's libraries for sets and lists, which come with carefully tuned automation setup. Still, their proofs tend to require additional induction proofs of auxiliary lemmas, especially when reasoning about iteration. Our formalization is 700 lines long. We prove all axioms of Kleene algebra and connect our representation to the set-of-words view via the bijections out and in. Except for those bijections our formalization does not rely on any other library. Moreover, when we changed our 5000 lines long formalization of a coalgebraic decision procedure for WS1S [28] to use the infinite tries instead of the set-of-words view, our proofs about WS1S became approximately 300 lines shorter. Apparently, a coalgebraic library is a good fit for a coalgebraic procedure.

Paulson presents a concise formalization of automata theory based on hereditarily finite sets [19]. For the semantics he reuses Krauss and Nipkow's set-of-words formalization.

7 Conclusion

We have presented a particular Formal Structure for Computation and Deduction: infinite tries modeling formal languages. Although this representation is semantic and infinite, it is suitable for computation – in particular we obtain a matching algorithm for free on tries constructed by regular operations. Deduction does not come short either: coinduction is the convenient reasoning tool for infinite tries. Coinductive proofs are concise, especially for (in)equational theorems such as the axioms of Kleene algebra.

Codatatypes might be just the right tool for thinking algorithmically about semantics. We hope to have contributed to their dissemination by outlining some of their advantages.

Acknowledgments. I thank Andrei Popescu for introducing me to the coinductive view on formal languages. I am grateful to Tobias Nipkow and David Basin for encouraging this work. Jasmin Blanchette, Andrei Popescu, Ralf Sasse, and the anonymous FSCD reviewers gave many precious hints on early drafts of this paper helping to improve both the presentation and the content. Jurriaan Rot suggested important related work.

References

1 Andreas Abel and Brigitte Pientka. Wellfounded recursion with copatterns: A unified approach to termination and productivity. In G. Morrisett and T. Uustalu, editors, *ICFP'13*, pages 185–196. ACM, 2013.

2 Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. Copatterns: programming infinite structures by observations. In R. Giacobazzi and R. Cousot, editors, *POPL'13*, pages 27–38. ACM, 2013.

3 Jasmin Christian Blanchette, Johannes Hölzl, Andreas Lochbihler, Lorenz Panny, Andrei Popescu, and Dmitriy Traytel. Truly modular (co)datatypes for Isabelle/HOL. In *ITP 2014*, volume 8558 of *LNCS*, pages 93–110. Springer, 2014.

4 Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. Foundational extensible corecursion. In J. Reppy, editor, *ICFP 2015*, pages 192–204. ACM, 2015.

5 Filippo Bonchi, Marcello M. Bonsangue, Jan J. M. M. Rutten, and Alexandra Silva. Brzozowski's algorithm (co)algebraically. In R.L. Constable and A. Silva, editors, *Logic and Program Semantics – Essays Dedicated to Dexter Kozen on the Occasion of His 60th Birthday*, volume 7230 of *LNCS*, pages 12–23. Springer, 2012.

6 Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In R. Giacobazzi and R. Cousot, editors, *POPL'13*, pages 457–468. ACM, 2013.

7 Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964. doi:10.1145/321239.321249.

8 Adam Chlipala. *Certified Programming with Dependent Types – A Pragmatic Introduction to the Coq Proof Assistant*. MIT Press, 2013. URL: http://mitpress.mit.edu/books/certified-programming-dependent-types.

9 Eduardo Giménez and Pierre Castéran. A tutorial on [co-]inductive types in Coq. 1998. URL: http://www.labri.fr/perso/casteran/RecTutorial.pdf.

10 Florian Haftmann and Tobias Nipkow. Code generation via higher-order rewrite systems. In M. Blume, N. Kobayashi, and G. Vidal, editors, *FLOPS 2010*, volume 6009 of *LNCS*, pages 103–117. Springer, 2010.

11 Ralf Hinze. Concrete stream calculus: An extended study. *J. Funct. Program.*, 20(5-6):463–535, 2010.

12 Bart Jacobs and Jan Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222–259, 1997.

13 Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Inf. Comput.*, 110(2):366–390, 1994.

14 Dexter Kozen and Alexandra Silva. Practical coinduction. Technical report, Cornell University, 2012. URL: `http://hdl.handle.net/1813/30510`.

15 Alexander Krauss and Tobias Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *J. Automated Reasoning*, 49:95–106, 2012. published online March 2011.

16 Andreas Lochbihler and Johannes Hölzl. Recursive functions on lazy lists via domains and topologies. In G. Klein and R. Gamboa, editors, *ITP 2014*, volume 8558 of *LNCS*, pages 341–357. Springer, 2014.

17 Tobias Nipkow and Gerwin Klein. *Concrete Semantics: With Isabelle/HOL*. Springer, 2014. URL: `http://www.in.tum.de/~nipkow/Concrete-Semantics`.

18 Lawrence C. Paulson. Mechanizing coinduction and corecursion in higher-order logic. *J. Logic Comp.*, 7(2):175–204, 1997.

19 Lawrence C. Paulson. A formalisation of finite automata using hereditarily finite sets. In A.P. Felty and A. Middeldorp, editors, *CADE-25*, volume 9195 of *LNCS*, pages 231–245. Springer, 2015.

20 Tillmann Rendel, Julia Trieflinger, and Klaus Ostermann. Automatic refunctionalization to a language with copattern matching: with applications to the expression problem. In K. Fisher and J.H. Reppy, editors, *ICFP 2015*, pages 269–279. ACM, 2015.

21 Jurriaan Rot, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Coinductive proof techniques for language equivalence. In A. Horia Dediu, C. Martín-Vide, and B. Truthe, editors, *LATA 2013*, volume 7810 of *LNCS*, pages 480–492. Springer, 2013.

22 Jurriaan Rot, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Proving language inclusion and equivalence by coinduction. *Inf. Comput.*, 246:62–76, 2016.

23 J. J. M. M. Rutten. Universal coalgebra: A theory of systems. *Theor. Comput. Sci.*, 249:3–80, 2000.

24 Jan J. M. M. Rutten. Automata and coinduction (an exercise in coalgebra). In D. Sangiorgi and R. de Simone, editors, *CONCUR 1998*, volume 1466 of *LNCS*, pages 194–218. Springer, 1998. `doi:10.1007/BFb0055624`.

25 Anton Setzer. How to reason coinductively informally. To appear in R. Kahle, T. Strahm, and T. Studer (Eds.): Festschrift on occasion of Gerhard Jäger's 60th birthday. Available from `http://www.cs.swan.ac.uk/~csetzer/articles/jaeger60Birthdaymain.pdf`, 2015.

26 Alexandra Silva. A short introduction to the coalgebraic method. *ACM SIGLOG News*, 2(2):16–27, 2015.

27 Dmitriy Traytel. A codatatype of formal languages. In *Archive of Formal Proofs*. 2013. URL: `http://afp.sf.net/entries/Coinductive_Languages.shtml`.

28 Dmitriy Traytel. A coalgebraic decision procedure for WS1S. In Stephan Kreutzer, editor, *CSL 2015*, volume 41 of *LIPIcs*, pages 487–503. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.

29 Dmitriy Traytel, Andrei Popescu, and Jasmin Christian Blanchette. Foundational, compositional (co)datatypes for higher-order logic – Category theory applied to theorem proving. In *LICS 2012*, pages 596–605. IEEE Computer Society, 2012.

30 Dmytro Traytel. *Formalizing Symbolic Decision Procedures for Regular Languages*. PhD thesis, TU München, 2015. URL: http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20151016-1273011-1-9.

31 Philip Wadler. The Expression Problem. Available online at: http://tinyurl.com/wadler-ep, 1998.

Normalisation by Random Descent*

Vincent van Oostrom[1] and Yoshihito Toyama[2]

1 **Department of Computer Science, University of Innsbruck, Innsbruck, Austria**
`Vincent.van-Oostrom@uibk.ac.at`

2 **RIEC, Tohoku University, Sendai, Japan**
`toyama@riec.tohoku.ac.jp`

Abstract

We present abstract hyper-normalisation results for strategies. These results are then applied to term rewriting systems, both first and higher-order. For example, we show hyper-normalisation of the left–outer strategy for, what we call, left–outer pattern rewrite systems, a class comprising both Combinatory Logic and the $\lambda\beta$-calculus but also systems with critical pairs. Our results apply to strategies that need not be deterministic but do have Newman's random descent property: all reductions to normal form have the same length, with Huet and Lévy's external strategy being an example. Technically, we base our development on supplementing the usual notion of commutation diagram with a notion of order, expressing that the measure of its right leg does not exceed that of its left leg, where measure is an abstraction of the usual notion of length. We give an exact characterisation of such global commutation diagrams, for pairs of reductions, by means of local ones, for pairs of steps, we dub Dyck diagrams.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs, F.4.1 Mathematical Logic, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases strategy, hyper-normalisation, commutation, random descent

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.32

1 Introduction

The (hyper-)normalisation of the leftmost–outermost strategy is a fundamental result in Combinatory Logic and the λ-calculus, cf. [3, 7]. For the special case of the λ-calculus, the simple idea underlying the present paper is that normalisation of the leftmost–outermost strategy is due to it being both *deterministic*, there is at most one leftmost–outermost step from any given λ-term, and *compatible* with β, in the sense that if $M \to_\beta N$ then repeatedly performing the leftmost–outermost strategy on both M and N results either in a common reduct or in infinite reductions from both. Compatibility guarantees (Section 5) that each term β-convertible to some normal form is also convertible to that normal form by leftmost–outermost steps. Determinism guarantees that if there is such a *conversion* to normal form, then there *exists* a leftmost–outermost *reduction* from the term to the normal form, and so *all* leftmost–outermost reductions from that term terminate. A method for proving hyper-normalisation is obtained from this by strengthening compatibility with an *order* constraint expressing that in the above the leftmost–outermost reduction from M be at least as long as that from N. Ordered compatibility guarantees that β-steps never increase the *distance*, i.e. the length of the leftmost–outermost reduction of a term to its normal

[*] The research for this paper was initiated during a visit in November 2013 funded by RIEC, Tohoku University, of the first author to the second author.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 32; pp. 32:1–32:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

form, from which hyper-normalisation follows as leftmost–outermost steps do decrease that distance. We present an abstract account of this idea based on the following observations.

The first observation (Section 4) is that determinism can be relaxed to Newman's *random descent* property, all reductions to normal form have the same length [13, 20, 21, 16]. This vastly broadens the scope of the method. In addition to deterministic strategies such as leftmost–outermost, it covers, e.g., interaction net strategies, linear β-reduction, and Huet and Lévy's external strategy [8], allowing to contract redexes that are outermost until eliminated (by contracting the redex itself or some overlapping redex [19]). Technically, whereas for deterministic systems the notion of distance is well-defined because given an object there is a *unique reduction* to its normal form, it is still well-defined for random descent systems since although reductions to normal form then need no longer be unique they have a *unique length*.

The second observation (Section 3) is that all reductions from an object to normal form having the same length is equivalent to the *order* constraint that for each pair of reductions from the object to normal form, the first is at least as long as the second. Working towards deciding it, we characterise this property of peaks of reductions by means of a property of *local* peaks (of steps). More precisely, we show it necessary and sufficient that such local peaks be completable by means of a *Dyck*-conversion, a conversion in which the number of forward steps never (for any prefix) exceeds the number of backward steps. Technically, we establish the above in a *commutation* setting where the reductions being ordered may be reductions in two distinct rewrite systems. In Section 6 we turn this local criterion into a *critical pair* criterion for a concrete class of higher-order term rewrite systems, dubbed *left–outer Dyck*, comprising both Combinatory Logic and the $\lambda\beta$-calculus.

The third observation is that we may abstract the notion of length in the random descent property into a notion of measure, allowing steps having different measures to coexist. This covers, for example, systems having 'macro' steps abbreviating several 'micro' steps. We build this into our setup right from the start (Section 2), by introducing *derivation* and *conversion* monoids which allow to measure reductions respectively conversions.

For terminating systems, the classical division of work for proving confluence is to first *localise* the confluence property for abstract rewrite systems (Newman's Lemma), and then using that establish confluence for term rewrite systems by means of a *critical pair* criterion (Huet's Critical Pair Lemma). We structure our paper accordingly, first localising random descent to the *local Dyck property* (Sections 3, 4) for abstract rewrite systems, and then using that to establish a critical pair criterion, the *left–outer Dyck property*, for establishing hyper-normalisation of the left–outer strategy for term rewrite systems (Sections 5, 6).

We employ the untyped λ-calculus with β- and/or η-reduction [2] as a running example, marking the examples where en passant new results are obtained by a double dagger (‡).

Contribution Apart from unifying our earlier results [20, 21, 16], with a clean separation into results for abstract and term rewriting, the main contributions of this paper are the notions of measured rewrite system, ordered commutation, compatibility and Dyck diagrams.

2 Preliminaries

We assume basic knowledge of term rewriting and the λ-calculus [1, 2, 19], and use notation from [19]. We employ abstract rewrite systems (ARSs) and strategies for them as defined in [19, Chapters 8 and 9]; cf. also [16]. In particular, *steps* are first-class citizens of abstract rewrite systems[1] and a *strategy* for a rewrite system is a sub-rewrite system on the same set of objects, with the same set of normal forms.[2] Throughout we assume

$\blacktriangleright$, $\to$, ${\color{red}\blacktriangleright}$, ${\color{blue}\rightarrowtriangle}$ are rewrite systems on the same set of objects, unless stated otherwise.

A strategy $\to$ for the rewrite system $\blacktriangleright$ is ($\blacktriangleright$-)*normalising* if $\to$ is terminating on objects that are $\blacktriangleright$-normalising, i.e. if for all a with $a \blacktriangleright^* b$ for some $\blacktriangleright$-normal form b, a is $\to$-terminating. The strategy $\to$ is $\blacktriangleright$-*hyper-normalising* if $\to/\blacktriangleright$, i.e. $\blacktriangleright^* \cdot \to \cdot \blacktriangleright^*$, is $\blacktriangleright$-normalising.

▶ **Example 1.** For the $\lambda\beta$-calculus, selecting an outermost redex for contraction yields a strategy (since any term not in β-normal form contains some outermost redex) that is non-deterministic, the *outermost* strategy. Starting from the outermost strategy, selecting the *leftmost* redex (in the tree ordering, cf. Definition 27) for rewriting yields a strategy for it that is deterministic. Composing both yields a strategy for β-reduction that is deterministic again, the *leftmost–outermost* strategy. The leftmost strategy starting from β-reduction, is also well-defined but non-deterministic. The leftmost–outermost strategy is hyper-normalising [3], but the leftmost and outermost strategies are not even normalising, cf. $(\lambda x.x\Omega)(\lambda xy.z)\Omega$.

For convenience we recapitulate notational conventions and their mnemonic values.

▶ Notation 2. We employ:

$\blacktriangleright$ arrow notation to denote an ARS having steps as first-class citizens;
$\to$ part of notation $\blacktriangleright$ to express that $\to$ is a $\blacktriangleright$-strategy, i.e. part of $\blacktriangleright$;
$\leftarrow$ converse of notation $\to$ to expresses that $\leftarrow$ is converse of $\to$; and
${\color{red}\blacktriangleleft}\!{\color{blue}\rightarrowtriangle}$ union of notations ${\color{red}\blacktriangleleft}$ and ${\color{blue}\rightarrowtriangle}$ to express ${\color{red}\blacktriangleleft}\!{\color{blue}\rightarrowtriangle}$ is union of ${\color{red}\blacktriangleleft}$ and ${\color{blue}\rightarrowtriangle}$.

We use sub/superscripts to express restricting/extending the corresponding notion.

At the core of this paper are the following *commutation* versions for pairs of rewrite systems of standard notions for single rewrite systems (PN is the obvious 'peak' version of NF).

▶ **Definition 3.** We say the *normal form* property (NF) holds if $a \; {\color{red}\blacktriangleleft}\!{\color{blue}\rightarrowtriangle}^* \; b$ with a in ${\color{blue}\rightarrowtriangle}$-normal form implies $a \;{}^*{\color{red}\blacktriangleleft}\; b$, the *peak normal form* property (PN) holds if $a \;{}^*{\color{red}\blacktriangleleft} \cdot {\color{blue}\rightarrowtriangle}^*\; b$ with a in ${\color{blue}\rightarrowtriangle}$-normal form implies $a \;{}^*{\color{red}\blacktriangleleft}\; b$, the *Church–Rosser* property (CR) holds if $a \; {\color{red}\blacktriangleleft}\!{\color{blue}\rightarrowtriangle}^* \; b$ implies $a \;{\color{blue}\rightarrowtriangle}^* \cdot {}^*{\color{red}\blacktriangleleft}\; b$, and *commutation* (CO) holds if $a \;{}^*{\color{red}\blacktriangleleft} \cdot {\color{blue}\rightarrowtriangle}^*\; b$ implies $a \;{\color{blue}\rightarrowtriangle}^* \cdot {}^*{\color{red}\blacktriangleleft}\; b$.

CR is also known as e.g. factorisation, postponement, preponement, and separation, cf. [4]. Instantiating both ${\color{red}\blacktriangleright}$ and ${\color{blue}\rightarrowtriangle}$ to $\to$ yields the usual notions NF, PN, CR and CO (then called *confluence*). That NF $\iff$ PN $\impliedby$ CR $\iff$ CO is folklore, cf. [4]. In order to be able to express that one rewrite system is 'better' than another, we enrich these notions/diagrams in two ways: we equip steps with a measure (with respect to which conversions are compared) and we allow for infinite reductions (defining all such to be 'worse' than finite ones).

▶ **Definition 4.** Let $\mathcal{M}$ be a monoid $(M, +, \bot)$. We call $\blacktriangleright$ an $\mathcal{M}$-*measured* rewrite system if it comes equipped with a map from $\blacktriangleright$-steps to $M - \{\bot\}$. The (*derivation*) *measure* of a sequence of $\blacktriangleright$-steps is the sum of the measures of its steps, from left to right. The (*conversion*) *measure* of a sequence of $\blacktriangleleft$- and $\blacktriangleright$-steps is a pair having as first component the sum of the measures of its $\blacktriangleleft$-steps, from right to left, and as second component the sum of the measures of its $\blacktriangleright$-steps, from left to right. $\mathcal{M}$ is a *derivation* monoid if it comes equipped with a well-founded partial order $\le$ such that $\bot$ is the *least* element ($\bot \le m$ for all m)

[1] Only for systems having at most one step between any pair of objects do we speak of rewrite *relations*.
[2] Sub-rewrite systems that do change the set of normal forms, e.g. call-by-value for the $\lambda\beta$-calculus, are *not* strategies (they result in different *calculi*).

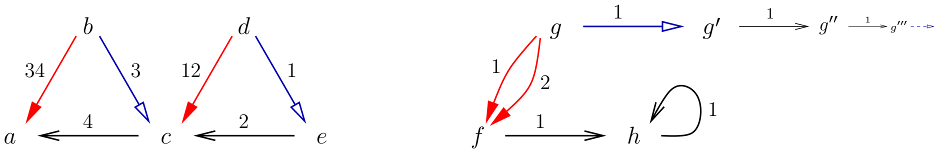


Figure 1 Measured rewrite systems on monoid of strings of digits ordered by embedding. Here $\to$ is used to denote steps in belonging to both ${\color{blue}\vartriangleright}$ and ${\color{red}\blacktriangleright}$, with the same measure.

and $+$ is *strictly monotonic* in both arguments (if $m < n$ then $m + k < n + k$ and $k + m < k + n$, for all m, n, k). A derivation monoid is *cancellative* if $m + k \geq n + k$ entails $m \geq n$ for all m, n, k. A *conversion* monoid is a commutative derivation monoid.

Excluding $\perp$ as measure of steps will ensure that by prefixing/suffixing a step to a reduction its measure strictly increases. We use subscripts to indicate measures; in Fig. 1 digit-strings.

▶ **Example 5.** In the measured rewrite system $\blacktriangleright$ that is the union of ${\color{red}\blacktriangleright}$ and ${\color{blue}\vartriangleright}$ in Fig. 1, we have $a\;{}_{1234}{\color{red}\blacktriangleleft}\!{\color{blue}\vartriangleright}^*_{31}\; e$ as witnessed by the conversion $a\;{}_{34}{\color{red}\blacktriangleleft}\; b\;{\color{blue}\vartriangleright}_3\; c\;{}_{12}{\color{red}\blacktriangleleft}\; d\;{\color{blue}\vartriangleright}_1\; e$, with the first component 1234 of the measure obtained by concatenating the measures 12 and 34 of the ${\color{red}\blacktriangleleft}$-steps, and the second component 31 by the measures 3 and 1 of the ${\color{blue}\vartriangleright}$-steps. In general, strings with concatenation and empty string ordered by embedding constitute a cancellative but non-commutative derivation monoid. Assuming commutativity yields the cancellative conversion monoid of multisets with multiset sum and empty multiset. More generally, equipping a string/multiset monoid with the order generated by the union of embedding with some simply terminating string/multiset rewrite system yields a derivation/conversion monoid which need *not* be cancellative; consider the orders generated by $ab \to ac/[a, b] \to [a, c]$.

▶ **Example 6.** Measuring all steps of a rewrite system by 1 in the conversion monoid of the natural numbers with addition and zero equipped with less–than–or-equal, the measure μ of a reduction $\blacktriangleright^*_{\mu}$ corresponds to its *length*, and for ${}_n\blacktriangleleft\!\blacktriangleright^*_m$ the integer $m - n$ corresponds to the *difference* of the conversion, the number of $\blacktriangleright$-steps minus the number of $\blacktriangleleft$-steps [20, 21].

Next, we consider ordering [16, Definition 5] and pasting [17, Examples 4,9] diagrams.

▶ **Definition 7.** A conversion is *ordered* if $m \geq n$ for its measure (m, n), and *cyclic* if its source and target are the same. *Shift* equivalence is generated by identifying a cyclic conversion $C \cdot D$ with $D \cdot C$. A (*conversion*) *diagram* is a pair C, D of conversions with the same sources and targets. Its *induced* conversion is $C^{-1} \cdot D$, inducing notions of measure and shift equivalence on diagrams, cf. [24]. *Pasting* diagrams shift equivalent to C, D and D, E (*on* D) gives C, E.

Occurrences of D, determined by picking an object on the cyclic conversion (the source of D) and a length (of D), may be used to disambiguate pasting. For commutative monoids shift equivalent diagrams have the same measure. A *derivation* diagram comprises two reductions.

▶ **Example 8.** In Fig. 1, we have, e.g., two diagrams $b\;{\color{red}\blacktriangleright}_{34}\; a, b\;{\color{blue}\vartriangleright}_3\; c\;{\color{red}\blacktriangleright}_4\; a$ and $d\;{\color{red}\blacktriangleright}_{12}\; c, d\;{\color{blue}\vartriangleright}_1\; e\;{\color{red}\blacktriangleright}_2\; c$ having underlying conversions $a\;{}_{34}{\color{red}\blacktriangleleft}\; b\;{\color{blue}\vartriangleright}_3\; c\;{\color{red}\blacktriangleright}_4\; a$ respectively $c\;{}_{12}{\color{red}\blacktriangleleft}\; d\;{\color{blue}\vartriangleright}_1\; e\;{\color{red}\blacktriangleright}_2\; c$. The former is shift equivalent to the diagram $c\;{}_3{\color{blue}\vartriangleleft}\; b\;{\color{red}\blacktriangleright}_{34}\; a\;{}_4{\color{red}\blacktriangleleft}\; c, c$ the latter to $c, c\;{}_{12}{\color{red}\blacktriangleleft}\; d\;{\color{blue}\vartriangleright}_1\; e\;{\color{red}\blacktriangleright}_2\; c$, and pasting them on c yields $c\;{}_3{\color{blue}\vartriangleleft}\; b\;{\color{red}\blacktriangleright}_{34}\; a\;{}_4{\color{red}\blacktriangleleft}\; c, c\;{}_{12}{\color{red}\blacktriangleleft}\; d\;{\color{blue}\vartriangleright}_1\; e\;{\color{red}\blacktriangleright}_2\; c$. Whereas the first of the original diagrams is an ordered derivation diagram (both the ‘counterclockwise’ and ‘clockwise’ measures of the 1st are 34) its shift equivalent is not ($34 \not\geq 43$), and the result of pasting is neither shift equivalent to a derivation diagram nor to an ordered diagram.

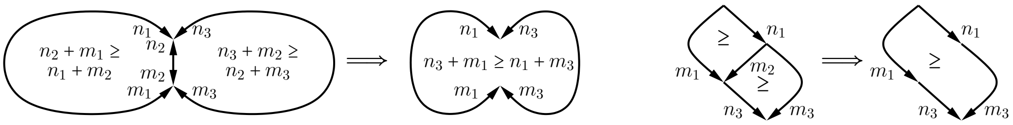


Figure 2 Preservation of order ('counterclockwise' ≥ 'clockwise) by pasting on ${}_{n_2}\!\blacktriangleleft\!\!\blacktriangleright^{*}_{m_2}$.

Pasting derivation diagrams on reductions requires associativity of +, for conversion diagrams it also requires commutativity, and for pasting on conversions also cancellation is needed:

▶ **Lemma 9.**
- *Pasting preserves order for cancellative conversion monoids (Fig. 2 left);*
- *Pasting on a reduction (m_2 or n_2 is $\bot$ in Fig. 2) preserves order for conversion monoids;*
- *Pasting ordered derivation diagrams on a reduction with the source of one and target of the other, gives a diagram shift equivalent to an ordered derivation diagram (Fig. 2 right).*

Proof. Let conversions C_i have measure (n_i, m_i) in the pasted diagrams C_1, C_2 and C_2, C_3. In the first two items, orderedness yields $n_2 + m_1 \geq n_1 + m_2$ and $n_3 + m_2 \geq n_2 + m_3$.

- Combining both yields $n_2 + m_1 + n_3 + m_2 \geq n_1 + m_2 + n_2 + m_3$ from which we conclude to $n_3 + m_1 \geq n_1 + m_3$ by commutativity and cancelling $m_2 + n_2$;
- If w.l.o.g. $n_2 = \bot$, the assumptions yield $n_3 + m_1 \geq n_3 + n_1 + m_2 \geq n_1 + m_3$ by commutativity;
- If w.l.o.g. the reduction has the same target as the original 1st diagram and the same source as the 2nd (see Fig. 2 right), then orderedness of those original derivation diagrams yields $m_1 \geq n_1 + m_2$ and $m_2 + n_3 \geq m_3$, so $m_1 + n_3 \geq n_1 + m_2 + n_3 \geq n_1 + m_3$. ◀

▶ **Definition 10.** The rewrite system $\rightarrowtriangle^{\infty}$ [4] has the same objects as $\rightarrowtriangle$ and a step from a to b for each infinite $\rightarrowtriangle$-rewrite sequence from a and for any b. Given a monoid $\mathcal{M}$, the monoid $\mathcal{M}^{\top}$ is obtained by adjoining a fresh element $\top$ to the carrier and defining $\top + \top = \top + m = m + \top = \top$ for all $m \in M$. Given a $\mathcal{M}$-measured rewrite system $\rightarrowtriangle$, mapping $\rightarrowtriangle^{\infty}$-steps to $\top$ gives rise to an $\mathcal{M}^{\top}$-measured rewrite system $\rightarrowtriangle \cup \rightarrowtriangle^{\infty}$, and similarly for a pair $\blacktriangleright, \rightarrowtriangle$ (see Definition 4).

Although $\rightarrowtriangle^{\infty}$ is not common in rewriting yet, it is in relational program semantics [4]. The extension from $\mathcal{M}$ to $\mathcal{M}^{\top}$ preserves commutativity, and cancelling elements of $\mathcal{M}$. (Although not needed here, note the first item of Lemma 9 needs m_2 or n_2 to be finite to go through). Two rewrite systems particularly important for this paper are $(\rightarrowtriangle \cup \rightarrowtriangle^{\infty})^{*}$ and $(\blacktriangleleft \cup \rightarrowtriangle \cup \rightarrowtriangle^{\infty})^{*}$ which we will refer to as *extended* reduction ($\rightarrowtriangle^{\circledast}$) and conversion ($\blacktriangleleft\!\!\rightarrowtriangle^{\circledast}$), respectively. Beware that locations of superscipts matter, e.g. $\leftarrowtriangle^{\infty}$ would be distinct from the converse of $\rightarrowtriangle^{\infty}$.

▶ **Example 11.** In the measured rewrite system of Fig. 1 $f\ {}_{2}\!\blacktriangleleft\!\!\rightarrowtriangle^{\circledast}_{\top}\ b$ since $f\ {}_{2}\!\blacktriangleleft\ g \rightarrowtriangle^{\infty}_{\top}\ b$ as g admits an infinite $\rightarrowtriangle$-reduction $g \rightarrowtriangle g' \rightarrowtriangle \ldots$. We do *not* have $b \leftarrowtriangle\!\!\blacktriangleright^{\circledast} f$.

We may assume $\rightarrowtriangle^{\infty}$ to occur only, if at all, at the end of an extended reduction or conversion because $\rightarrowtriangle^{\circledast} = \rightarrowtriangle^{*} \cup \rightarrowtriangle^{\infty} = \rightarrowtriangle^{*} \cdot (\rightarrowtriangle^{\infty})^{=}$ and $\blacktriangleleft\!\!\rightarrowtriangle^{\circledast} = \blacktriangleleft\!\!\rightarrowtriangle^{*} \cdot (\rightarrowtriangle^{\infty})^{=}$, with superscript $=$ denoting reflexive closure. This process does not increase the first component of the measure and leaves the second unchanged. To differentiate between elements of $\mathcal{M}$ and $\mathcal{M}^{\top}$, we henceforth use $m, n, k, \ldots$ to range over the former and $\mu, \nu, \kappa, \ldots$ to range over the latter. We call the former *finite* as can be vindicated by setting infinite sums of non-$\bot$-elements to $\top$ and noting that infinite measures are not affected by (un)folding $\rightarrowtriangle^{\infty}$: if $a \rightarrowtriangle^{\infty}_{\top} b$ is witnessed by $a \rightarrowtriangle_{m} a' \rightarrowtriangle_{m'} a'' \rightarrowtriangle_{m''} \ldots$ then its measure is $m + m' + m' + \ldots = \top$, and so is the measure of $a \rightarrowtriangle_{m} a' \rightarrowtriangle^{\infty}_{\top} b$ because $m + \top = \top$. Thus, a reduction is infinite if and only if the corresponding extended reduction has measure $\top$ with respect to the length measure (Example 6).

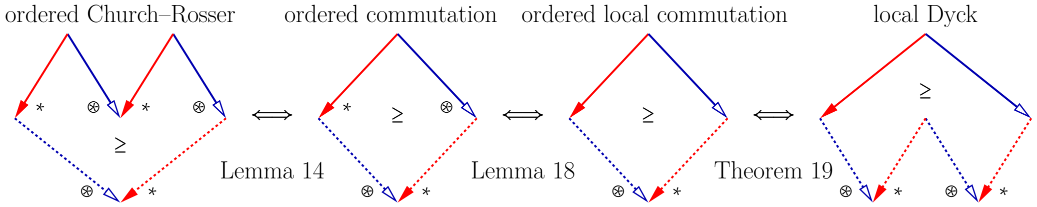


Figure 3 Localising ordered Church–Rosser, restricting the ∀, widening the ∃, to local Dyck.

3 Ordered commutation and Dyck diagrams

We introduce a property, *ordered* Church–Rosser, sufficient for the measure of the $\blacktriangleright$-reductions from a given object to be an *upper bound* on the measures of its $\rightarrowtriangle$-reductions. This will be used in Sections 5–6 to show normalisation of the latter via that of the former. Here we work towards deciding the property, *localising* it to the *local Dyck* property, Fig. 3.

Throughout, we assume $\blacktriangleright, \rightarrowtriangle$ are measured rewrite systems for the same derivation monoid.

▶ **Definition 12.** We say the *ordered* normal form property (ONF) holds if $a \;{}_{n}\!\blacktriangleleft\!\!\rightarrowtriangle^{\circledast}_{\mu}\; b$ with a in $\rightarrowtriangle$-normal form implies $a \;{}^{*}_{n'}\!\blacktriangleleft\; b$, the *ordered* peak normal form property (OPN) holds if $a \;{}^{*}_{n}\!\blacktriangleleft \cdot \rightarrowtriangle^{\circledast}_{\mu}\; b$ with a in $\rightarrowtriangle$-normal form implies $a \;{}^{*}_{n'}\!\blacktriangleleft\; b$, the *ordered* Church–Rosser property (OCR) holds if $a \;{}_{n}\!\blacktriangleleft\!\!\rightarrowtriangle^{\circledast}_{\mu}\; b$ implies $a \rightarrowtriangle^{*}_{\mu'} \cdot {}^{*}_{n'}\!\blacktriangleleft\; b$, and *ordered* commutation (OCO) holds if $a \;{}^{*}_{n}\!\blacktriangleleft \cdot \rightarrowtriangle^{\circledast}_{\mu}\; b$ implies $a \rightarrowtriangle^{\circledast}_{\mu'} \cdot {}^{*}_{n'}\!\blacktriangleleft\; b$, with the *order* constraint $n + \mu' \geq \mu + n'$ ($\mu' = \bot$ by default).

In words, a commutation diagram is ordered if the measure of its left leg is as large as that of its right leg. Note that this corresponds exactly to orderedness (see Definition 7) of the corresponding derivation diagram for derivation monoids. Similarly, for OCR the constraint corresponds to orderedness of the corresponding conversion diagram for conversion monoids. Remark that if OCR/OCO and $a \blacktriangleleft\!\!\rightarrowtriangle^{\circledast} b/\; a \;{}^{*}\!\blacktriangleleft \cdot \rightarrowtriangle^{\circledast} b$, then if a is $\rightarrowtriangle$-terminating so is b and the corresponding OCR, OCO diagram is finite. Lemma 9 vindicates pasting such diagrams. That OCO need *not* imply commutation or confluence, follows by considering non $\rightarrowtriangle$-terminating such a; OCO holds in Example 13, but commutation not for $f \;{}_{i}\!\blacktriangleleft\; g \rightarrowtriangle_{1} g'$.[3]

▶ **Example 13.** In Fig. 1, OCO and OPN are easily seen to hold by considering the three 'interesting' local peaks $a \;{}_{34}\!\blacktriangleleft\; b \rightarrowtriangle_{3} c$, $c \;{}_{12}\!\blacktriangleleft\; d \rightarrowtriangle_{1} e$ and $f \;{}_{i}\!\blacktriangleleft\; g \rightarrowtriangle_{1} g'$, that are completed into ordered commutation diagrams by respectively $a \;{}_{4}\!\blacktriangleleft\; c$, $c \;{}_{2}\!\blacktriangleleft\; e$ and $f \rightarrowtriangle^{\infty}_{\top} g'$.

Composing the first two of these local peaks (on c) yields a conversion $a \;{}_{1234}\!\blacktriangleleft\!\!\rightarrowtriangle^{*}_{31}\; e$ that can (only) be completed into a commutation diagram by $a \;{}^{*}_{24}\!\blacktriangleleft\; e$, which does not satisfy the order constraint as $1234 \ngeq 3124$, showing neither OCR nor ONF holds, cf. Example 8.

The four notions relate to each other as in the unordered (finite) case.

▶ **Lemma 14.** *ONF $\iff$ OPN $\Longleftarrow$ OCR $\iff$ OCO, for conversion monoids.*

Proof. All implications hold by definition except for OCO $\Longrightarrow$ OCR and OPN $\Longrightarrow$ ONF. These are shown by induction on the number of peaks in a conversion, pasting diagrams as in the unordered case [4]. Since the diagrams are conversion diagrams, a conversion monoid is needed (cf. Example 13) to let order be preserved by pasting on reductions (Lemma 9). ◀

[3] The converse also fails as witnessed by $b_0 \rightarrowtriangle b_1 \rightarrowtriangle \ldots$ and $a \blacktriangleleft b_i$ for all i.

Note that the four notions are asymmetric in that they consider infinite $\rightarrowtriangle$-reductions but only finite $\blacktriangleright$-reductions. However, this sufficies for *bounding* $\rightarrowtriangle$-reductions by $\blacktriangleright$-reductions:

▶ Proposition 15. If OPN and $\blacktriangleright, \rightarrowtriangle$ have the same normal forms, then for any peak $a \,{}^{\circledast}_{\nu}\!\blacktriangleleft \cdot \rightarrowtriangle^{\circledast}_{\mu} b$ of maximal reductions, $\nu \geq \mu$ and if the left leg is finite so is the right leg and $a = b$.

Proof. If the left leg of the peak has infinite length, then ν is $\top$ and we conclude trivially. Otherwise, the peak has shape $a \,{}^{*}_{n}\!\blacktriangleleft \cdot \rightarrowtriangle^{\circledast}_{\mu} b$ for some n with a in normal form. By OPN for it, there exists $a \,{}^{*}_{m'}\!\blacktriangleleft b$ such that $n \geq \mu + m'$. As n is finite, so is μ, so the right leg of the peak is finite and by maximality must end in a normal form and $a = b$. ◀

In particular, if $\blacktriangleright, \rightarrowtriangle$ are strategies for the same rewrite system, as is, e.g., the case for the systems in Fig. 1, normalisation of the former entails normalisation of the latter.[4]

Having shown their usefulness, we turn to localising the properties. *Localisation* of a ∀∃-property aims at finding an equivalent property that *restricts* the domain of the ∀-quantifier and *widens* that of the ∃-quantifier, to enable or ease deciding it (automatically). The classical example is localisation of the Church–Rosser property (∀conversions ∃valley) by *restricting* first to peaks, then further to local peaks [13] and finally by *widening* to conversions below the source [22], for terminating rewrite systems. Here, as we already have OCR $\iff$ OCO, we restrict OCO to ordered local commutation and widen that to the local Dyck property.

▶ **Definition 16.** Ordered *local* commutation arises from OCO by restricting both legs of the peak to reductions of length 1. A diagram comprising a local peak $a \,{}_{n}\!\blacktriangleleft \cdot \rightarrowtriangle_{m} b$ and extended conversion $a \,{}_{n'}\!\blacktriangleleft\!\!\rightarrowtriangle^{\circledast}_{\mu'} b$ is a *Dyck* diagram if $n + \mu' \geq m + n'$ and the *Dyck*-condition holds: for every prefix (of which there are finitely many) $a \,{}_{n''}\!\blacktriangleleft\!\!\rightarrowtriangle^{\circledast}_{m''} c$ of the conversion $n + m'' > n''$. We say the systems are *locally Dyck* if each such peak can be completed into a Dyck diagram.

Our naming is based on that for the length measure the number of backward ($\blacktriangleleft$) steps in the conversion then never exceeds the number of forward ($\rightarrowtriangle$) steps, as in the Dyck language.

▶ **Example 17.** Let for some N, the abstract rewrite system $\blacktriangleright$ be given by $b_i \blacktriangleright b_{i+1} \blacktriangleleft a_i \blacktriangleright c_{i+1} \blacktriangleleft c_i$ for all $1 \leq i \leq N$ and $b_{N+1} \blacktriangleright c_{N+1}$, and $\rightarrowtriangle$ be the $\blacktriangleright$-strategy comprising all $\blacktriangleright$-steps except those from a_i to b_{i+1}, both with respect to the length measure. The only interesting local peaks are $b_{i+1} \blacktriangleleft a_i \rightarrowtriangle c_{i+1}$ which can be completed into a Dyck diagram; for $i < N$ by $b_{i+1} \rightarrowtriangle b_{i+2} \blacktriangleleft a_{i+1} \rightarrowtriangle c_{i+2} \blacktriangleleft c_{i+1}$ (the conditions for its 5 prefixes are respectively $1 > 0$, $2 > 0$, $2 > 1$, $3 > 1$ and $3 > 2$) and for $i = N$ by $b_{N+1} \rightarrowtriangle c_{N+1}$. For the system, this takes a number of conversion (back-and-forth) steps linear in N, whereas naïvely completing into ordered commutation diagrams requires a quadratic number of reduction steps, cf. [17, Example 8]. Hence, using the following, the length of $\blacktriangleright$-reductions bounds those of $\rightarrowtriangle$-reductions.

▶ **Lemma 18.** *Ordered commutation iff ordered local commutation.*

Proof. The only–if-direction is trivial. For the if-direction, it suffices to consider peaks $a \,{}^{*}_{\hat{n}}\!\blacktriangleleft \cdot \rightarrowtriangle^{\circledast}_{\hat{\mu}} b$ such that a is $\rightarrowtriangle$-terminating as otherwise we conclude by $a \rightarrowtriangle^{\infty}_{\top} b$. We show such a peak can be completed by $a \rightarrowtriangle^{\circledast}_{\mu'} \cdot \,{}^{*}_{n'}\!\blacktriangleleft b$ with $\hat{n} + \mu' \geq \hat{\mu} + n'$ into an OCO diagram, finite by the Remark above Example 13, by induction on $(a, \hat{n})$ ordered by the *leg* order

[4] For such strategies the proposition shows that OPN is *sufficient* for $\rightarrowtriangle$ being universally better than $\blacktriangleright$, in the sense of [16], and hence [16] that $\rightarrowtriangle$ is normalising, minimal (if $\rightarrowtriangle \subseteq \blacktriangleright$) and $\blacktriangleright$ is perpetual, maximal (if $\blacktriangleright \subseteq \rightarrowtriangle$). OPN is not *necessary* for it: $b \blacktriangleleft a \rightarrowtriangle c$.

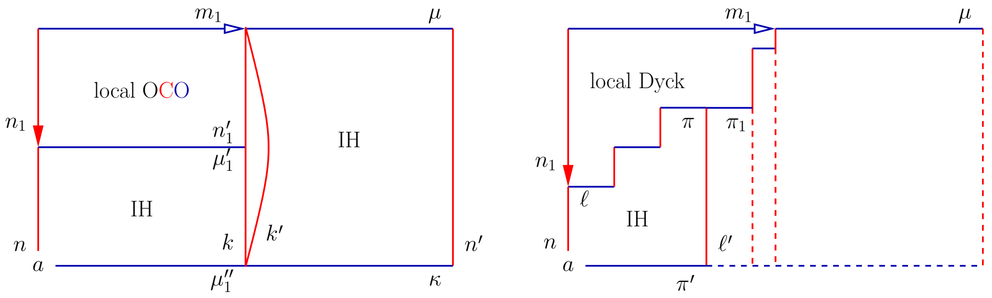


Figure 4 Proofs of Lemma 18 (left) and Theorem 19 (right).

$>_\bullet$, the lexicographic product of $\rightarrowtriangle^{+}$ and $>$. If either leg of the peak is empty, it is trivial. Otherwise, it has shape $a\ {}_{n}^{*}\!\blacktriangleleft\ a_1\ {}_{n_1}\!\blacktriangleleft \cdot \rightarrowtriangle_{m_1} b_1 \rightarrowtriangle^{\circledast}_{\mu} b$ with $\hat{n} = n_1 + n$ and $\hat{m} = m_1 + \mu$; Fig. 4.

⟦ Ordered local commutation applied to $a_1\ {}_{n_1}\!\blacktriangleleft \cdot \rightarrowtriangle_{m_1} b_1$ yields $a_1 \rightarrowtriangle^{\circledast}_{\mu'_1} c_1\ {}_{n'_1}^{*}\!\blacktriangleleft\ b_1$ with $n_1 + \mu'_1 \geq m_1 + n'_1$. Consider the peak $a\ {}_{n}^{*}\!\blacktriangleleft\ a_1 \rightarrowtriangle^{\circledast}_{\mu'_1} c_1$. The induction hypothesis applies to it as $\hat{n} = n_1 + n > n$, giving $a \rightarrowtriangle^{\circledast}_{\mu''_1} a'\ {}_{k}^{*}\!\blacktriangleleft\ c_1$ with $n + \mu''_1 \geq \mu'_1 + k$. Vertically pasting this diagram to the one for the local peak yields a diagram satisfying $n_1 + n + \mu''_1 \geq n_1 + \mu'_1 + k \geq m_1 + n'_1 + k$, i.e. that is ordered. Abbreviating $n'_1 + k$ to k' it has a valley $a \rightarrowtriangle^{\circledast}_{\mu''_1} a'\ {}_{k'}^{*}\!\blacktriangleleft\ b_1$. ⟧

Now consider the peak $a'\ {}_{k'}^{*}\!\blacktriangleleft\ b_1 \rightarrowtriangle^{\circledast}_{\mu} b$. The induction hypothesis applies to it as either $a \rightarrowtriangle^{*}_{\mu''_1} a'$ is not empty, or it is empty and then $a = a'$, μ''_1 is $\bot$, and instantiating the inequality above yields $n_1 + n \geq m_1 + k' > k'$. Hence we obtain a valley $a' \rightarrowtriangle^{\circledast}_{\kappa} \cdot\ {}_{n'}^{*}\!\blacktriangleleft\ b$ with $k' + \kappa \geq \mu + n'$. Setting μ' to $\mu''_1 + \kappa$ we conclude by horizontal diagram pasting, yielding the order constraint $n_1 + n + \mu' = n_1 + n + \mu''_1 + \kappa \geq m_1 + k' + \kappa \geq m_1 + \mu + n'$. ◀

▶ **Theorem 19.** *Ordered commutation iff locally Dyck, for cancellative conversion monoids.*

Proof. For the only–if-direction it suffices to remark that in an ordered local commutation diagram $\rightarrowtriangle$-steps precede $\blacktriangleleft$-steps, so that orderedness entails the Dyck-condition. For the if-direction, we proceed exactly as in the proof of Lemma 18 but using the local Dyck property instead of ordered local commutation. That is, we replace the part between ⟦ and ⟧ there by:

The local Dyck property applied to $a_1\ {}_{n_1}\!\blacktriangleleft \cdot \rightarrowtriangle_{m_1} b_1$ yields $a_1\ {}_{n'_1}^{*}\!\blacktriangleleft\!\!\rightarrowtriangle^{\circledast}_{\mu'_1} b_1$ such that $n_1 + \mu'_1 \geq m_1 + n'_1$ and satisfying the Dyck-condition: for every prefix $a_1\ {}_{\ell}^{*}\!\blacktriangleleft\!\!\rightarrowtriangle^{\circledast}_{\pi} c$ of the conversion, $n_1 + \pi > \ell$. We show by a sub-induction on the length of the prefix, that there exists a valley $a \rightarrowtriangle^{\circledast}_{\pi'} d\ {}_{\ell'}^{*}\!\blacktriangleleft\ c$ completing $a\ {}_{n}^{*}\!\blacktriangleleft\ a_1\ {}_{\ell}^{*}\!\blacktriangleleft\!\!\rightarrowtriangle^{\circledast}_{\pi} c$ into an ordered diagram, i.e. such that $\ell + n + \pi' \geq \pi + \ell'$. The case of the empty prefix being trivial, assume the property holds for a given prefix up to c, and distinguish cases on the next step of the prefix.

If $c\ {}_{\ell_1}\!\blacktriangleleft\ c_1$, then we simply affix it: $a\ {}_{n}^{*}\!\blacktriangleleft\ a_1\ {}_{\ell}^{*}\!\blacktriangleleft\!\!\rightarrowtriangle^{\circledast}_{\pi} c\ {}_{\ell_1}\!\blacktriangleleft\ c_1$ is completed by the valley $a \rightarrowtriangle^{\circledast}_{\pi'} d\ {}_{\ell'}^{*}\!\blacktriangleleft\ c\ {}_{\ell_1}\!\blacktriangleleft\ c_1$ into an ordered diagram: $\ell_1 + \ell + n + \pi' \geq \ell_1 + \pi + \ell'$.

If $c \rightarrowtriangle_{\pi_1} c_1$ or $c \rightarrowtriangle^{\infty}_{\pi_1} c_1$, then consider the peak between it and $d\ {}_{\ell'}^{*}\!\blacktriangleleft\ c$, see Fig. 4. That the main induction applies to it follows (by a decrease in the 1st component) in case π' is not $\bot$ and otherwise (by a decrease in the 2nd component) by combining orderedness with the Dyck condition: $n_1 + \ell + n \geq n_1 + \pi + \ell' > \ell + \ell'$ hence $n_1 + n > \ell'$ by cancelling ℓ. Thus we obtain a valley $d \rightarrowtriangle^{\circledast}_{\pi'_1} d_1\ {}_{\ell'}^{*}\!\blacktriangleleft\ c_1$ completing the peak to an ordered diagram. (By the Remark above Example 13, this shows $c \rightarrowtriangle^{\infty}_{\pi_1} c_1$ is in fact impossible.) Pasting it to the one of the IH yields an ordered diagram with valley $a \rightarrowtriangle^{\circledast}_{\pi'} d \rightarrowtriangle^{\circledast}_{\pi'_1} d_1\ {}_{\ell'}^{*}\!\blacktriangleleft\ c_1$, as desired.

Let $a \rightarrowtriangle^{\circledast}_{\mu''_1} a'\ {}_{k'}^{*}\!\blacktriangleleft\ b_1$ be the valley thus obtained for the whole of $a_1\ {}_{n'_1}^{*}\!\blacktriangleleft\!\!\rightarrowtriangle^{\circledast}_{\mu'_1} b_1$. By the induction hypothesis it satisfies the order constraint $n'_1 + n + \mu''_1 \geq \mu'_1 + k'$. Combining it with the order constraint $n_1 + \mu'_1 \geq m_1 + n'_1$ of the local Dyck diagram, adding the respective sides and cancelling $n'_1 + \mu'_1$ gives $n_1 + n + \mu''_1 \geq m_1 + k'$, showing that the valley completes the peak $a \;{}_{n}^{*}\!\blacktriangleleft a_1 \;{}_{n_1}\!\blacktriangleleft \cdot \rightarrowtriangle_{m_1} b_1$ into an ordered commutation diagram. ◀

For non-cancellative conversion monoids the if-direction may fail. The theorem allows one to *localise* showing that a strategy $\blacktriangleright$ bounds another strategy $\rightarrowtriangle$, by checking all *local* peaks between both to be completable into Dyck diagrams. We give 2 typical examples, cf. [16].

▶ **Example 20** (‡)**.** For the λ-calculus with η-reduction $\blacktriangleright$ (see Example 40 for the rewrite rule), consider the innermost strategy $\rightarrowtriangle$, and, based on the idea that checking applicability of the η-rule involves checking absence of the bound variable from the body, measure a step by the *size* of its body. By orthogonality, linearity and preservation of innermost redexes, a local peak $N \;{}_{n}\!\blacktriangleleft M \rightarrowtriangle_{m} P$ either is trivial ($N = P$) or it can be completed by a valley of shape $N \rightarrowtriangle_{m'} Q \;{}_{n'}\!\blacktriangleleft P$. If the redexes are disjoint, then $m' = m$ and $n' = n$, so the diagram is Dyck. Otherwise, the latter is in the body of the former and we conclude to $n + m' \geq m + n'$ again because $m' = m$ and $n' = n - 2$ (an @ and λ have disappeared). Thus $\rightarrowtriangle$ is optimal, giving a lower bound on the (size) measure of η-reduction.

▶ **Example 21** (‡)**.** For the λ-calculus with β-reduction, let $\blacktriangleright$ be the leftmost–outermost strategy, $\rightarrowtriangle$ be the needed strategy [3], and consider a (non-trivial) local peak $N \blacktriangleleft M \rightarrowtriangle P$. By orthogonality and projection of leftmost–outermost steps over other steps $N \rightarrow_{\beta}^{*} Q \;{}_{\beta}\!\leftarrow P$ with the former a development of the residuals of $M \rightarrowtriangle P$. Factorising the former into needed steps followed by non-needed steps, and observing that the latter is a leftmost–outermost step again $N \rightarrowtriangle^{+} N' \rightarrow_{\beta}^{*} Q \blacktriangleleft P$ with the first non-empty by definition of neededness. By repeatedly contracting a leftmost–outermost redex starting from N' and performing its projection on Q until (if this happens at all) the terms reached by both are the same, yields a $\rightarrowtriangle$-reduction from N' (using leftmost–outermost redexes are needed) and a $\blacktriangleright$-reduction from Q of the same (possibly infinite) length. Therefore, $N \rightarrowtriangle_{\geq 1} N' \rightarrowtriangle_{\mu}^{\circledast} Q' \;{}_{\mu}^{\circledast}\!\blacktriangleleft Q \;{}_{1}\!\blacktriangleleft P$, i.e. a Dyck diagram (note that if μ is ∞, then $N' \rightarrowtriangle_{\infty}^{\circledast} Q$). Thus $\blacktriangleright$ is pessimal, giving an upper bound on the (length) measure of needed β-reduction (but not on non-needed; cf. $(\lambda x.y)\Omega$).

4 Ordered confluence and random descent

We show that instantiating both $\blacktriangleright, \rightarrowtriangle$ to the same rewrite system $\rightarrow$, and assuming it to be measured by a *conversion* monoid, all conditions of the previous section are equivalent, and sufficient to conclude that $\rightarrow$ *bounds itself.* We show that the bounding system ($\blacktriangleright$) being the same as the system being bound ($\rightarrowtriangle$) allows to replace the *order* constraint in the ordered normal form property by *equality*, yielding Newman's *random descent* property expressing that "if an end-form exists it is reached by random descent" [13, 20, 21, 16]. We localise it to the local Dyck property and give several examples.

▶ **Definition 22.** $\rightarrow$ has *random descent* (RD) [16] if $a \;{}_{n}\!\leftrightarrow_{\mu}^{\circledast} b$ with a in normal form, implies $a \;{}_{n'}^{*}\!\leftarrow b$ with $n = \mu + n'$. *Peak random descent* (PR) is obtained by restricting to $a \;{}_{n}^{*}\!\leftarrow \cdot \rightarrow_{\mu}^{\circledast} b$.

RD and PR being more strict versions of NF respectively PN, they have similar properties. In particular, RD $\implies$ PR, and if $a \;{}^{*}\!\leftarrow b$ with a in normal form and either RD or PR holds, then b is terminating. Note the n' in the definition only depends on b and is unique since applying either property to $a \;{}_{n'}^{*}\!\leftarrow b \rightarrow_{n''}^{*} a'$ for normal forms a, a' gives $a = a'$ and $n' = n''$. This justifies defining the *distance* $\mathsf{d}(b)$ of such an object b convertible to normal form to be that n', setting $\mathsf{d}(b)$ to ∞ for objects not convertible to normal form.

▶ **Example 23.** β-reduction does not have the random descent property for the length measure as witnessed e.g. by $(\lambda xz.zxx)((\lambda x.x)y)$ which allows reductions to normal form $\lambda z.zyy$ of lengths both 2 and 3. The restriction of β to the leftmost–outermost strategy does have RD because it is deterministic. The restriction to *linear* β-redexes, i.e. such that the bound variable occurs exactly once in the body, has RD because redex-patterns cannot be replicated.[5] On its own, η-reduction has RD as it is linear and orthogonal.

▶ **Lemma 24.** *ONF $\iff$ OPN $\iff$ OCR $\iff$ OCO $\iff$ PR $\iff$ RD, all are equivalent to ordered local confluence, and for cancellative monoids to the local Dyck property.*

Proof. By Lemma 14 and the above, to conclude to the implications on the first line, it suffices to show ONF $\implies$ RD and OCO $\impliedby$ PR. For the former, suppose $a \mathrel{{}_{n}{\leftrightarrow}^{\circledast}_{\mu}} b$ with a in normal form. By ONF, $a \mathrel{{}_{n'}{\overset{*}{\leftarrow}}} b$ with $n \geq \mu + n'$, so μ is finite. By ONF for $a \mathrel{{}_{n'}{\overset{*}{\leftarrow}}} b \mathrel{{}_{\mu}{\leftrightarrow}^{*}_{n}} a$, also $\mu + n' \geq n$. For the latter, consider a peak $a \mathrel{{}_{n}{\overset{*}{\leftarrow}}} \cdot \mathrel{\rightarrow^{\circledast}_{\mu}} b$. If a allows an infinite reduction, we conclude. Otherwise, $c \mathrel{{}_{m'}{\overset{*}{\leftarrow}}} a$ for some normal form c. By PR for $c \mathrel{{}_{m'}{\overset{*}{\leftarrow}}} a \mathrel{{}_{n}{\overset{*}{\leftarrow}}} \cdot \mathrel{\rightarrow^{\circledast}_{\mu}} b$, we obtain $a \mathrel{{}_{n'}{\overset{*}{\leftarrow}}} b$ with $n+m' = \mu+n'$. The second line follows by Lemma 18 and Theorem 19. ◀

As in the previous section, the implication from a peak-property to its conversion-property fails for derivation monoids (Example 13), and the lemma allows to *localise* random descent. The examples here and in the previous section are illustrative both of localisation and of the flexibility offered by measuring by length, size of subterm or pattern, rule, ... Many possibilities come to mind with as extreme case measuring a reduction by 'itself', say as the string or multiset of its steps. Our final example uses the left–outer order on positions.

▶ **Example 25.** The single rule term rewrite system $f(x,x) \mathrel{\blacktriangleright} f(x,f(x,a))$ has random descent (for the length measure) because there are no critical peaks and the rule is *variable preserving* in that all variables appear the same number of times in both the left- and right-hand sides. The latter condition guarantees that in the so-called variable-overlap case of the critical pair lemma, both legs of the resulting diagram have exactly the same length.

▶ **Example 26.** The term rewrite system given by

$$f(x,x) \mathrel{\blacktriangleright_1} f(x,g(x)) \quad f(x,x) \mathrel{\blacktriangleright_2} f(x,h(x)) \quad g(x) \mathrel{\blacktriangleright_1} h(x) \quad c \mathrel{\blacktriangleright_1} g(c)$$

has random descent with respect to the indicated *rule* measures (Definition 44, [17, Sect. 4.2]). As in Example 25 the rules are variable preserving, but now there is the critical peak (and its symmetric version) $f(x,g(x)) \leftarrow_1 f(x,x) \rightarrow_2 f(x,h(x))$ which however is completable by the step $f(x,g(x)) \rightarrow_1 f(x,h(x))$ into a Dyck diagram where both legs have measure 2.

▶ **Definition 27.** On positions in terms, the *left* relation is defined by $p \cdot i \cdot q \prec_l p \cdot j \cdot q'$ and the *outer* (or *prefix*) relation by $p \prec_o p \cdot i \cdot q$, for arbitrary positions p, q, q' and natural numbers $i < j$. The *left–outer* relation is defined by $\prec_{lo} = \prec_l \cup \prec_o$.

The relations $\prec_o$, $\prec_l$ and $\prec_{lo}$ are strict orders, $\prec_o$ and $\prec_l$ are disjoint, and $\preceq_{lo}$ is total.

▶ **Example 28** (‡)**.** The *spine* positions of a λ-term M are, if it has shape $\lambda\vec{x}.y\vec{M}$ then the displayed positions and the spine positions of the M_i prefixed by their position in M, and otherwise its *head spine* positions. The *head spine* positions of terms x, $\lambda x.M_1$, M_1M_2 are all positions $\preceq_{lo}$-related to the position of M_1 and all head spine positions of M_1 prefixed by its position in M. A *spine* redex-pattern is a redex-pattern at spine position. Always contracting such we call a *spine* strategy, which is justified by the fact that any term not in $\beta(\eta)$-normal form has at least one spine $\beta(\eta)$-redex-pattern.[6] That the spine strategy has random descent

[5] This is not a strategy for β-reduction in λ-calculus as e.g. $(\lambda xz.zxx)y$ is a normal form for it.

[6] A λ-term not in η-normal form need not contain a spine η-redex-pattern, e.g. $(\lambda x.x)\lambda y.zy$.

for β-reduction [3, Proposition 4.21] follows from that the β-rule does not overlap with itself, and that contracting a spine β-redex-pattern leaves a unique descendant (residual) of any non-overlapping spine position (redex-pattern). As the η-rule does not overlap with itself and is variable preserving, descendants are unique after η-steps and η-reduction has random descent. Since the (two) critical peaks between the β- and η-rules are trivial, they are locally Dyck, from which it follows that the spine strategy has random descent for $\beta\eta$-reduction.

The underlying intuition, generalising that for externality [8], is that the spine is a prefix of a term that persists linearly (contrary to externality, it may involve change, but only 'linear change'). It applies to some other term rewrite systems, e.g. Combinatory Logic as well. We conclude by an easy result allowing to infer RD for strategies.

▶ **Lemma 29.** *If $\rightarrowtriangle$ has random descent then so does any strategy $\rightarrow$ for it.*

▶ **Example 30.** Since the spine strategy has random descent for $\beta\eta$-reduction (see above), so does the left–outer (see, the text above, Definition 41) strategy.

5 Compatibility

We present a method to establish hyper-normalisation of strategies for abstract rewrite systems, based on a diagram we (inspired by Staples' notion of *compatible* refinement, cf. [19, Exercise 1.3.9]) dub *compatibility* governing the interaction between the strategy and the system, and show it can be made flexible by well-foundedly indexing steps giving rise to the notion of *decreasing* compatibility. As an application, one may immediately conclude normalisation of the needed strategy for $\lambda\beta\eta$, cf. [9, Chapter IV] and [3], from that of the spine strategy (Example 38), using that the former is *bounded* by the latter (Example 21). Our methods rely on the strategy having the random descent property, for an arbitrary cancellative conversion monoid, as introduced above.

We assume $\rightarrow$ is a strategy having random descent for the rewrite system $\rightarrowtriangle$.

▶ **Definition 31.** $\rightarrow$ is (*ordered*) *compatible* with $\rightarrowtriangle$, if $a \rightarrowtriangle b$ entails $a \;{}_{n}{\leftrightarrow}^{\circledast}_{\mu}\; b$ (with $\mu \geq n$).

▶ **Example 32** (‡)**.** Spine reduction is compatible with backward $\beta(\eta)$-steps and ordered compatible with forward such steps. We provide a proof of this later, via decreasing compatibility, but the intuition for that it holds is that, given a $\beta(\eta)$-step $M \rightarrowtriangle N$ one may contract an arbitrary spine redex in M. In case this $\rightarrow$-step yields N then we are done. Otherwise, one may contract 'the same' spine redex in N, project $M \rightarrowtriangle N$ over both, and repeat the process until the first case applies. We also conclude when this process proceeds indefinitely, as then we have constructed infinite $\rightarrow$-reductions from M and N.

▶ **Lemma 33.** *$\rightarrow$ is (ordered) compatible with $\rightarrowtriangle$ iff it is (ordered) compatible with $\rightarrowtriangle^*$.*

Proof. The if-direction is trivial by $\rightarrowtriangle$ being contained in $\rightarrowtriangle^*$ and the only–if-direction follows by an easy induction on the length of $\rightarrowtriangle^*$-reductions (using commutativity). ◀

▶ **Theorem 34.** *If $\rightarrow$ is a random descent $\rightarrowtriangle$-strategy compatible with $\leftarrowtriangle$, it is $\rightarrowtriangle$-normalising. If, moreover, $\rightarrow$ is ordered compatible with $\rightarrowtriangle$, it is $\rightarrowtriangle$-hyper-normalising and $\rightarrowtriangle$ has NF.*

Proof. Suppose $\rightarrow$ is a random descent $\rightarrowtriangle$-strategy that is compatible with $\leftarrowtriangle$, and $a \rightarrowtriangle^* b$ with b in $\rightarrowtriangle$-normal form. Lemma 33 yields that $\rightarrow$ is compatible with $\leftarrowtriangle^*$, which applied to $b \leftarrowtriangle^* a$ entails $b \;{}_{n}{\leftrightarrow}^{\circledast}_{\mu}\; a$. By random descent a is $\rightarrow$-terminating.

Suppose, moreover, $\rightarrow$ is ordered compatible with $\rightarrowtriangle$, and a is $\rightarrowtriangle$-normalising. To prove $\rightarrow/\rightarrowtriangle$-reduction terminates on a, it suffices to show that for $a \rightarrowtriangle a'$ we have $\mathsf{d}(a) \geq \mathsf{d}(a')$ and if in fact $a \rightarrow a'$ then $\mathsf{d}(a) > \mathsf{d}(a')$. Distinguish cases on whether or not $a \rightarrowtriangle a'$ is a $\rightarrow$-step.

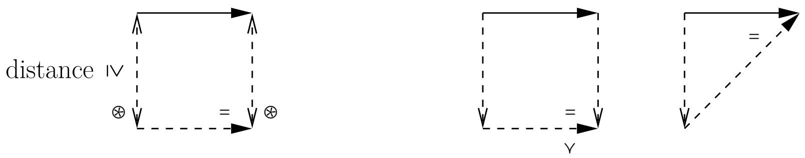


Figure 5 Decreasing compatibility (without condition, left) and its instance of Corollary 37.

- If $a \rightarrow_m a'$, then by RD for $b \leftarrow^*_{\mathsf{d}(a)} a \rightarrow_m a'$, $b \leftarrow^*_{\mathsf{d}(a')} a'$ with $\mathsf{d}(a) = m + \mathsf{d}(a') > \mathsf{d}(a')$.
- If $a \blacktriangleright a'$ then by $\rightarrow$ being ordered compatible with $\blacktriangleright$, we have $a\ {}_n{\leftrightarrow}^{\circledast}_{\mu}\ a'$ with $\mu \geq n$. By RD for $b \leftarrow^*_{\mathsf{d}(a)} a\ {}_n{\leftrightarrow}^{\circledast}_{\mu}\ a'$, then $b \leftarrow^*_{\mathsf{d}(a')} a'$ with $n + \mathsf{d}(a) = \mu + \mathsf{d}(a')$ (so μ is finite). Hence $\mathsf{d}(a) \geq \mathsf{d}(a')$ since in any cancellative derivation monoid if $n + k = m + \ell$ and $m \geq n$, then $n + k = m + \ell \geq n + \ell$ so by cancellation $k \geq \ell$ (without cancellation this property need not hold; let $\geq$ be generated by the multiset rule $[a, b] \rightarrow [a, a]$ and consider $[a, b, a]$).

Finally, we conclude to NF of $\blacktriangleright$ since the assumptions yield $\rightarrow$ is compatible with $\blacktriangleleft\!\blacktriangleright$ hence with $\blacktriangleleft\!\blacktriangleright^*$ by Lemma 33, and since $\rightarrow$ is a $\blacktriangleright$-strategy. ◀

By introducing a well-founded order on steps, as a parameter to the definition of compatibility, we increase its flexibility, cf. [20, Corollary 3.7], [21, Corollary 3]. The intuition captured is that $\blacktriangleright$-steps go 'at least as much forward (μ) as backward (n)' with respect to $\rightarrow$, recursively.

▶ **Definition 35.** We say $\rightarrow$ is (*ordered*) *decreasingly* compatible with $\blacktriangleright$, if for some well-founded order $\prec$ on the steps of $\blacktriangleright$, for all $\blacktriangleright$-steps ϕ, it holds $\blacktriangleright_{\phi} \subseteq {}_n{\leftrightarrow}^{\circledast}_{\mu} \cdot \blacktriangleright^{=}_{\phi(\mu=n)} \cdot {}_{n'}{\leftrightarrow}^{\circledast}_{\mu'}$ with $\mu \geq n$ (and $\mu + \mu' \geq n' + n$), where $\phi(\mathsf{true})$ denotes $\curlyvee\phi$, the set of steps $\prec$-related to ϕ, and $\phi(\mathsf{false})$ denotes the set of all steps.

▶ Proposition 36. If $\rightarrow$ is a random descent strategy for $\blacktriangleright$, then $\rightarrow$ is (ordered) compatible with $\blacktriangleright$ iff it is (ordered) decreasingly compatible with $\blacktriangleright$.

Proof. For the only–if-direction, set $\prec$ to the empty relation and μ, n both to $\bot$. For the if-direction, let $\rightarrow$ have random descent and be (ordered) decreasingly compatible with $\blacktriangleright$. We show for $\rightarrow$-terminating a, that if $a \blacktriangleright_{\phi} b$ then $a\ {}_k{\leftrightarrow}^{\circledast}_{\lambda}\ b$ (with $\lambda \geq k$), by well-founded induction on the pair $(\mathsf{d}(a), \phi)$ ordered by the lexicographic product of $>$ and $\succ$, using 'vertical pasting' (cf. Fig. 5). We load the induction hypothesis to show that λ is finite. ◀

We present a simple, easily applicable sufficient criterion. (Note that it is weaker than Theorem 34 as it requires more, than needed, for objects *not* convertible to normal form.)

▶ **Corollary 37.** *If $\rightarrow$ is a random descent strategy with respect to the length measure, for $\blacktriangleright$ and $\prec$ a well-founded order $\prec$ on the steps of $\blacktriangleright$ such that for all $\blacktriangleright$-steps ϕ, it holds $\blacktriangleright_{\phi} \subseteq (\rightarrow \cdot \blacktriangleright^{=}_{\curlyvee\phi}) \cup (\rightarrow \cdot \blacktriangleright^{=} \cdot \leftarrow)$, then $\rightarrow$ is hyper-normalising and $\blacktriangleright$ has NF.*

Proof. By Theorem 34 and Proposition 36, instantiating measures to 0/1. ◀

▶ **Example 38** (‡)**.** To prove that the spine strategy $\rightarrow$ is hyper-normalising for $\beta(\eta)$ in the λ-calculus, it suffices to show the assumptions of Corollary 37 are satisfied when setting $\blacktriangleright$ to (nonempty) $\beta(\eta)$-multisteps, taking as well-founded order $\prec$ on them the *development* order, generated by ordering a multistep above each of its residuals after contracting anyone of its redex-patterns. This is a well-founded order by the finite developments theorem (see [2]). Let $\phi : M \multimap\!\!\blacktriangleright N$ be a nonempty $\beta(\eta)$-multistep and $\psi : M \rightarrow M'$ a spine step.

If the respective (sets of) redex-patterns of ϕ and ψ do not have overlap, then we may compute the residual of each after the other with common reduct, say, N'. As spine redex-patterns are (uniquely) preserved after taking residuals, we have $\psi/\phi : N \to N'$. Observing that the 'other' residual ϕ/ψ either is empty (covered by reflexivity) or a nonempty multistep, we conclude to the 'right case' of Corollary 37.

If the redex-pattern of ψ has overlap with some redex-pattern, say ϕ_1, in ϕ, then we may develop ϕ as $M \to_{\phi_1} M' \multimap_{\phi/\phi_1} N$. We conclude to the 'left case' of Corollary 37 as ϕ/ϕ_1 is either empty or $\prec$-smaller than ϕ.

6 Hyper-normalisation for left–outer Dyck systems

We turn the reasoning in Example 38 into a critical pair criterion for a general class of term rewrite systems comprising both combinatory logic and the $\lambda\beta$-calculus. We dub the criterion *left–outer Dyck*, as it will entail (by Theorem 19) that all critical peaks can be completed into Dyck diagrams by means of *critically left–outer* steps, i.e. by left–outer steps that are closed under substitutions and (left–outer) contexts. The latter have RD so the criterion guarantees (by Theorem 34) hyper-normalisation for the left–outer strategy, cf. [21, Section 9].

We formalise our results in the setting of higher-order term rewrite systems where terms are simply typed λ-terms modulo $\alpha\beta\eta$-equality over simply typed (variables and) function symbols [23], using η-long β-normal forms as unique representatives of $\alpha\beta\eta$-equivalence classes of terms. To obtain decidability of criticality of left–outer steps, we focus on Nipkow's higher-order pattern rewrite systems [10], restricted to systems that are left-normal and local.

▶ **Definition 39.** A term is a pattern [12] if the free variables in it have sequences of pairwise distinct bound variables as arguments. A pattern is

- *linear* if each free variable that occurs in it, occurs in it exactly once;
- *fully-extended* [6] if each free variable occurring in it has as arguments a sequence comprising the variables bound above it;
- *local* [15, Footnote 1] if it is both linear and fully-extended;
- *left-normal* [14, 9] if each free position in it only $\prec_{lo}$-relates (see Definition 27) to other such, with positions in subterms having a free variable as head being *free*.

These notions extend to rewrite rules and systems via their left-hand side(s).

▶ **Example 40.** The higher-order rewrite rules corresponding to β- and η-reduction are:

$$@(\lambda x.M(x))N \to M(N) \qquad\qquad \lambda x.@Mx \to M$$

where λ and @ (henceforth left implicit) are appropriately typed function symbols, usually called abs respectively app [10, 19]. The left-hand sides $(\lambda x.M(x))N, \lambda x.Mx$ are patterns: the free variables M and N only have sequences of pairwise distinct bound variables, x and the empty sequence (twice), as arguments. The former, $(\lambda x.M(x))N$, is both local and left-normal: linear as its free variables M, N both occur once in it, fully-extended since M has the variable x bound above it as argument and N has no arguments, and left-normal as the free position $1 \cdot 2 \cdot 2 \cdot 1$ (of $M(x)$) only $\prec_{lo}$-relates to 2 (the position of N). The latter, $\lambda x.Mx$, is neither local nor left-normal. It is linear but not fully-extended since M does not have the variable x bound above it among its (empty) list of arguments, and not left-normal as the free position $2 \cdot 1 \cdot 1 \cdot 2$ (of M) $\prec_{lo}$-relates to $2 \cdot 1 \cdot 2$ (the position of x, not free). That is, the β-rule on its own constitutes a left-normal and local higher-order pattern rewrite system, but not so (locality falters) combined with the η-rule.

On first-order terms locality coincides with linearity. Throughout we use $\mathrel{-\!\!\blacktriangleright}$ and $\mathrel{-\!\!\circ\!\!\!\blacktriangleright}$ to denote the one-step respectively (nonempty) multistep [19] abstract rewrite system underlying a local and left-normal higher-order pattern rewrite system. We focus on *left–outer* rewriting, denoted by $\to$, i.e. the restriction of $\mathrel{-\!\!\blacktriangleright}$ to contracting redexes at $<_{lo}$-more positions:

▶ **Definition 41.** For $<$ a strict order on positions, a position q of a redex-pattern is *$<$-more* if there is overlap between the redex-pattern and each redex-pattern at a position $p < q$, and *$<$-most* if there is no redex-pattern at a position $p < q$.

That $\to$ indeed is a strategy for $\mathrel{-\!\!\blacktriangleright}$ holds since any term not in normal form contains a *leftmost–outermost* redex, i.e. a redex at $<_{lo}$-most position. A left–outer redex need not be leftmost–outermost, e.g. when overlapped from above by a redex that *is* leftmost–outermost. Locality allows to characterise (critically) left–outer steps via (critically) left–outer contexts.

▶ **Definition 42.**
- A context is *left–outer* if it is single-hole and there is no redex-pattern at a position that $<_{lo}$-relates to the position of the hole, cf. [21, Definition 17];
- A context C is *critically* left–outer if for every substitution σ and left–outer context D, the context $D[C^\sigma]$ is left–outer. A step in such a context is a *critically left-outer* step.

Any critically left–outer context, in particular the empty context $\Box$, is left–outer. The context $f(x,\Box)$ is left–outer, but not critically so if $f(a,y) \mathrel{-\!\!\blacktriangleright} \ldots$, since instantiating x by a turns it into a non-left–outer context. Similarly, $f(\Box)$ is not critically left–outer if $g(f(x)) \mathrel{-\!\!\blacktriangleright} \ldots$.

▶ Proposition 43.
- q is left–outer in $C[\ell^\sigma]_q$, for ℓ a left-hand side, iff C is left–outer;
- $q \cdot p$ is left–outer in $D[(C[\ell^\sigma]_p)^\tau]_q$, if D is left–outer and C is critically left–outer;
- A context is critically left–outer iff it is a single-hole context such that each symbol at a position that $<_{lo}$-relates to the position of the hole, is not a free variable and cannot be overlapped with a redex-pattern.

Proof.
- This follows from that if a term contains a redex at position p, then changing it, e.g. replacing a subterm by a hole or vice versa, at any position $p <_{lo} q$ not overlapping the redex-pattern, does not change redexhood, by locality.
- Using substitutions are homomorphic, $D[(C[\ell^\sigma]_p)^\tau]_q = D[C^\tau[(\ell^\sigma)^\tau]_p]_q = D[C^\tau[\ell^{\sigma;\tau}]_p]_q$. By the assumption that D is left–outer and C is critically left–outer, $D[C^\tau]$ is left–outer. Combining both, we conclude by the previous item.
- By locality and left-normality. ◀

In the first-order case the proposition yields a decision procedure for whether or not a rewrite step is critically left–outer, since testing for the presence of variables in terms and unification of (parts of) left-hand sides of rules with terms/contexts are both effective. In the higher-order case this is not immediate in general: although (parts of) left-hand sides of rules are assumed to be patterns, the term/context may be arbitrary, a non-pattern. We use $\to_{\#}$ to denote the restriction of the left–outer strategy $\to$ to critically left–outer steps.

▶ **Definition 44.** A rewrite system $\mathrel{-\!\!\blacktriangleright}$ is *left–outer Dyck* if each critical peak[7] can be completed into a Dyck diagram by a conversion $\leftrightarrow^{\circledast}$ comprising $\to_{\#}$-steps only, for a given rule measure. Here a *rule* measure is a measure only depending on the rule applied.

The $\lambda\beta$-calculus and Combinatory Logic are left–outer Dyck, in the absence of critical pairs. An ARS is locally Dyck iff its associated TRS is left–outer Dyck (measures *are* rule measures). Closure under contexts and substitutions makes rule measures suited for critical pair criteria.

[7] We employ a symmetric notion of critical *peak* arising from the, usually asymmetrically defined, notion of critical *pair*, by allowing either step of the peak to be the/a root step of the pair.

▶ **Lemma 45.** *If $\blacktriangleright$ is left–outer Dyck, then $\to$ is locally Dyck.*

Proof. Consider a local peak of left–outer steps $t \leftarrow \cdot \rightarrow s$. By totality of $\leq_{lo}$, the positions of the respective contracted redexes are either identical or one is $<_{lo}$-related to the other. In either case the redex-patterns must have overlap, giving rise to a critical peak $t' \leftarrow \cdot \rightarrow s'$ of steps that are again left–outer such that the peak is encompassed via, say, left–outer context C and substitution σ, i.e. such that $t = C[t'^{\sigma}] \leftarrow \cdot \rightarrow C[s'^{\sigma}] = s$. By the assumption that the system is left–outer Dyck, for the given rule measure, the critical peak can be completed into a Dyck diagram by a $\rightarrow_{\#}$-conversion from t' to s'. By definition and by rule measures only depending on the rule applied, encompassing the conversion into the (left–outer) context C and substitution σ, yields a $\rightarrow$-Dyck-conversion from $t = C[t'^{\sigma}]$ to $C[s'^{\sigma}] = s$, as desired. ◀

▶ **Corollary 46.** *If $\blacktriangleright$ is left–outer Dyck, then $\to$ has random descent.*

Proof. By Lemmata 24 and 45. ◀

▶ **Example 47.** The left-normal, local term rewrite system with rules

$$a \blacktriangleright b \qquad f(x) \blacktriangleright g(x) \qquad h(f(b)) \blacktriangleright c \qquad c \blacktriangleright d \qquad h(g(b)) \blacktriangleright d$$

is left–outer Dyck, hence the left–outer strategy has random descent. The only interesting critical peaks are between the second and third rules, $h(g(b)) \blacktriangleleft h(f(b)) \blacktriangleright c$. The peak is non-trivial but shown to be *root balanced joinable* [21] by $h(g(b)) \rightarrow_{\#} d \leftarrow_{\#} c$.

▶ **Example 48.** The left-normal, local term rewrite system with rules

$$a \blacktriangleright g(a) \qquad f(a) \blacktriangleright f(c) \qquad g(x) \blacktriangleright d \qquad c \blacktriangleright d$$

is left–outer Dyck, hence the left–outer strategy has random descent. Only overlap between the first and the second rule (and the other way around) gives rise to an interesting critical peak $f(g(a)) \blacktriangleleft f(a) \blacktriangleright f(c)$. The peak is completable into a Dyck diagram but is *not* root balanced joinable: the redex-patterns contracted in the joining valley $f(g(a)) \rightarrow_{\#} f(d) \leftarrow_{\#} f(c)$ occur in the critically left–outer but non-empty, context $f(\Box)$.

Left-normality and locality can be viewed as syntactic conditions guaranteeing that left–outer redexes are *external* [8], they descend [19] uniquely until overlapped by the contracted redex:

▶ Proposition 49. Left–outer steps descend along non-overlapping multisteps.

Proof. Locality guarantees that being a redex or not only depends on its pattern, not on its variables being instantiated appropriately. This prevents creating a redex above the left–outer redex by steps below or parallel to its redex-pattern, just by changing instantiation of its variables. Left-normality guarantees that no redex-pattern can be created above a left–outer redex by means of contracting a redex parallel to (to the right of) it. ◀

▶ Remark. Non-fully-extendness of the η-rule causes that a left–outer redex u may descend to a non-left–outer redex along a non-left–outer step in $\lambda\beta\eta$, e.g. in $\lambda x.u((\lambda y.z)x)x \blacktriangleright \lambda x.uzx$.

▶ **Theorem 50.** *If a rewrite system is left–outer Dyck, then it has the normal form property, and the left–outer strategy is hyper-normalising for it.*

Proof. By assumption and Corollary 46 the left–outer strategy $\to$ has random descent. To conclude that the rewrite system $\blacktriangleright$ has the normal form property and $\to$ is hyper-normalising for it, it suffices to show that the same hold for the the (nonempty) multistep rewrite system

$\mathrel{-\!\!\!\circ\!\!\!\blacktriangleright}$ because $\mathrel{-\!\!\blacktriangleright} \subseteq \mathrel{-\!\!\!\circ\!\!\!\blacktriangleright} \subseteq \mathrel{-\!\!\blacktriangleright}^*$. To that end, it suffices by Theorem 34 to show that $\to$ is ordered compatible with the rewrite system $\mathrel{-\!\!\blacktriangleright}$ and compatible with its converse $\mathrel{\blacktriangleleft\!\!-}$, which in turn follow by Proposition 36 from that $\to$ is ordered decreasingly compatible with $\mathrel{-\!\!\!\circ\!\!\!\blacktriangleright}$, and decreasingly compatible with its converse, which we both show simultaneously by well-foundedly ordering multisteps by the development order (see Example 38), considering an arbitrary nonempty multistep $W : t \mathrel{-\!\!\!\circ\!\!\!\blacktriangleright} s$. By $\to$ being a strategy there is a left–outer step from t, say $u : t \to_m t'$. We distinguish cases on whether or not u overlaps some step in W.

If u overlaps no step in W, then both are orthogonal and we may compute their mutual residuals $W/u : t' \mathrel{-\!\!\!\circ\!\!\!\blacktriangleright} s'$ and $u/W : s \mathrel{-\!\!\!\circ\!\!\!\blacktriangleright} s'$. By Proposition 49, left–outer redex-patterns are (uniquely) preserved after taking residuals, so $u/W : s \to_m s'$ by rule measures only depending on the rule. Thence $\mathrel{-\!\!\!\circ\!\!\!\blacktriangleright}_W \subseteq \to_m \cdot \mathrel{-\!\!\!\circ\!\!\!\blacktriangleright}^= \cdot {}_m\!\leftarrow$, from which the conditions for (ordered) decreasing compatibility of $\to$ with (the converse of) $\mathrel{\blacktriangleleft\!\!\!\circ\!\!\!-}_W$ follow.

If the redex-pattern of u has overlap with some redex-pattern, say w, in W, then we may develop W as $w : t \mathrel{-\!\!\blacktriangleright} t''$ followed by $W' : t'' \mathrel{-\!\!\!\circ\!\!\!\blacktriangleright} s$ with $W' = W/w$. Since the redex-patterns u, w yielding the peak $t' \leftarrow \cdot \mathrel{-\!\!\blacktriangleright} t''$ have overlap in its source t, the peak encompasses some critical peak $r' \leftarrow \cdot \mathrel{-\!\!\blacktriangleright} r''$, say via context C and substitution σ. The context C is left–outer as a prefix of the left–outer context in which u occurs. By the assumption that $\mathrel{-\!\!\blacktriangleright}$ is left–outer Dyck, the peak and its symmetric version can be completed into Dyck diagrams by $\to_{\#}$-conversions from r' to r'' and vice versa, respectively. Encompassing these again by the left–outer context C and substitution σ yields, by the steps in the conversions being *critically* left–outer and by rule measures only depending on the rule applied, Dyck diagrams for $\to$-conversions from t' to t'' and vice versa. By the former, $\mathrel{-\!\!\!\circ\!\!\!\blacktriangleright}_W \subseteq \to_m \cdot {}_n\!\leftrightarrow^{\circledast}_{\mu} \cdot \mathrel{-\!\!\!\circ\!\!\!\blacktriangleright}$ with $m + \mu > n$ from which we conclude to ordered decreasing compatibility of $\to$ with $\mathrel{-\!\!\!\circ\!\!\!\blacktriangleright}_W$. By the latter, $\mathrel{\blacktriangleleft\!\!\!\circ\!\!\!-}_W \subseteq \mathrel{\blacktriangleleft\!\!\!\circ\!\!\!-}_{W'} \cdot \leftrightarrow^{\circledast}$ from which we conclude to decreasing compatibility of $\to$ with $\mathrel{\blacktriangleleft\!\!\!\circ\!\!\!-}_W$, as W is larger than W' in the development order. ◀

▶ **Example 51.** The rewrite systems of Examples 47 and 48, $\lambda\beta$- and CL-reduction are left-normal, local and locally Dyck, so the left–outer strategy is hyper-normalising for each.

▶ **Example 52.** Consider the local, left-normal first-order term rewrite system given by rules

$$\mathsf{zeros} \mathrel{-\!\!\blacktriangleright}_1 0 : \mathsf{zeros} \qquad \mathsf{hd}(x : y) \mathrel{-\!\!\blacktriangleright}_1 x \qquad \mathsf{hd}(\mathsf{zeros}) \mathrel{-\!\!\blacktriangleright}_2 0$$

with measures as indicated. Its critical peak $0 \;{}_2\!\mathrel{\blacktriangleleft\!\!-} \mathsf{hd}(\mathsf{zeros}) \mathrel{-\!\!\blacktriangleright}_1 \mathsf{hd}(0 : \mathsf{zeros})$ is completed by $0 \;{}_1\!\mathrel{\blacktriangleleft\!\!-} \mathsf{hd}(0 : \mathsf{zeros})$ into a Dyck diagram, so the left–outer strategy is hyper-normalising.

7 Conclusion

We have generalised (hyper-)normalisation results from [20, 21] using the random descent property from [16]. Our development is based on a clean separation between the abstract and term rewrite results. At the abstract level we have introduced novel methods to compare strategies, by Dyck diagrams, and to prove their (hyper-)normalisation, by compatibility. At the term level, we have introduced a class of higher-order term rewrite systems, left–outer Dyck systems, comprising $\lambda\beta$ and CL.

Theorem 50 generalises [7, Theorem 25] on which their further developments are based. The generalisation is proper in that our results are restricted neither to deterministic strategies nor to first-order term rewrite systems, while sharing the advantage of being completely local (even more so). We expect it to be possible to incorporate several techniques from their work, in particular basic normalisation and factorisation,[8] into our approach. The approach to normalisation due to [11] is mostly incomporable to ours. On the one hand, that approach is based on square permutations (a special case of random descent) and on having finite 'permutation equivalence' classes (not needed in our approach). On the other hand, there normalisation for notions of result other than normal forms (think of head-normal forms) are considered. We intend to apply approach to hyper-normalisation to, e.g., λ-calculi with explicit substitutions or the necessary strategy [18], and compare our results to those of [5].

Acknowledgements. We thank B. Felgenhauer, J. Nagele, the attendants of the TCS seminar (Amsterdam) and the Master Seminar (Innsbruck), and the FSCD reviewers for feedback.

References

1 F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

2 H.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.

3 H.P. Barendregt, J.R.K. Kennaway, J.W. Klop, and M.R. Sleep. Needed reduction and spine strategies for the lambda calculus. *Information & Computation*, 75(3):191–231, 1987. `doi:10.1016/0890-5401(87)90001-0`.

4 N. Dershowitz. On lazy commutation. In *Languages: From Formal to Natural*, volume 5533 of *Lecture Notes in Computer Science*, pages 59–82. Springer, 2009. `doi:10.1007/978-3-642-01748-3_5`.

5 J.R.W. Glauert, R. Kennaway, and Z. Khasidashvili. Stable results and relative normalization. *Journal of Logic and Computation*, 10(3):323–348, 2000. `doi:10.1093/logcom/10.3.323`.

6 M. Hanus and C. Prehofer. Higher-order narrowing with definitional trees. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, volume 1103 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 1996. `doi:10.1007/3-540-61464-8_48`.

7 N. Hirokawa, A. Middeldorp, and G. Moser. Leftmost outermost revisited. In *Proceedings of the 26th International Conference on Rewriting Techniques and Applications*, volume 36 of *Leibniz International Proceedings in Informatics*, pages 209–222, 2015. `doi:10.4230/LIPIcs.RTA.2015.209`.

8 Gérard Huet and Jean-Jacques Lévy. Computations in orthogonal rewriting systems, I. In *Computational Logic: Essays in Honor of Alan Robinson.* The MIT Press, 1991. (accessed 27-4-2016). URL: `http://pauillac.inria.fr/~levy/pubs/81robinson1.pdf`.

9 J.W. Klop. *Combinatory Reduction Systems.* PhD thesis, Rijksuniversiteit Utrecht, 1980. (accessed 27-4-2016). URL: `http://janwillemklop.nl/Jan_Willem_Klop/Bibliography_files/9.PhDthesis-total.pdf`.

10 R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(1):3–29, 1998. `doi:10.1016/S0304-3975(97)00143-6`.

11 P.-A. Melliès. A stability theorem in rewriting theory. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*, pages 287–298. IEEE Computer Society Press, 1998. `doi:10.1109/LICS.1998.705665`.

[8] But note that for a rewrite system $\rightarrowtriangle$ given by $a \rightarrowtriangle b \rightarrowtriangle c$, $a' \rightarrowtriangle b' \rightarrowtriangle c'$, $a \rightarrowtriangle a'$, $b \rightarrowtriangle b'$ and $c \rightarrowtriangle c'$, and a strategy $\to$ obtained by omitting the step from a to b, factorisation fails, for $a \rightarrowtriangle b \to c$, but our methods, in particular Theorem 50, do apply.

12 D. Miller. Unification of simply typed lambda-terms as logic programming. In *Proceedings of the 8th International Conference on Logic Programming*, pages 253–281. The MIT Press, 1991. (accessed 27-4-2016). URL: `http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/iclp91.pdf`.

13 M. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942. `doi:10.2307/1968867`.

14 M.J. O'Donnell. *Computing in Systems Described by Equations*, volume 58 of *Lecture Notes in Computer Science*. Springer, 1977. `doi:10.1007/3-540-08531-9`.

15 V. van Oostrom. Finite family developments. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, volume 1232 of *Lecture Notes in Computer Science*, pages 308–322. Springer, 1997. `doi:10.1007/3-540-62950-5_80`.

16 V. van Oostrom. Random descent. In *Proceedings of the 18th International Conference on Rewriting Techniques and Applications*, volume 4533 of *Lecture Notes in Computer Science*, pages 314–328. Springer, 2007. `doi:10.1007/978-3-540-73449-9_24`.

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

18 R.C. Sekar and I.V. Ramakrishnan. Programming in equational logic: Beyond strong sequentiality. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science*, pages 230–241. IEEE Computer Society Press, 1990. `doi:10.1109/LICS.1990.113749`.

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

20 Y. Toyama. Strong sequentiality of left-linear overlapping term rewriting systems. In *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science*, pages 274–284. IEEE Computer Society Press, 1992. `doi:10.1109/LICS.1992.185540`.

21 Y. Toyama. Reduction strategies for left–linear term rewriting systems. In *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 198–223. Springer, 2005. `doi:10.1007/11601548_13`.

22 F. Winkler and B. Buchberger. A criterion for eliminating unnecessary reductions in the Knuth–Bendix algorithm. In *Proceedings of the Colloquium on Algebra, Combinatorics and Logic in Computer Science, Volume II*, volume 42 of *Colloquia Mathematica Societatis J. Bolyai*, pages 849–869, 1986.

23 D.A. Wolfram. *The Clausal Theory of Types*, volume 21 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1993.

24 H. Zantema, B. König, and H.J.S. Bruggink. Termination of cycle rewriting. In *Proceedings of the 25th International Conference on Rewriting Techniques and Applicationsand the 12th International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science, pages 476–490. Springer, 2014. `doi:10.1007/978-3-319-08918-8_33`.

Ground Confluence Prover Based on Rewriting Induction

Takahito Aoto[1] and Yoshihito Toyama[2]

1 **Faculty of Engineering, Niigata University, Niigata, Japan**
aoto@ie.niigata-u.ac.jp
2 **RIEC, Tohoku University, Sendai, Japan**
toyama@riec.tohoku.ac.jp

Abstract

Ground confluence of term rewriting systems guarantees that all ground terms are confluent. Recently, interests in proving confluence of term rewriting systems automatically has grown, and confluence provers have been developed. But they mainly focus on confluence and not ground confluence. In fact, little interest has been paid to developing tools for proving ground confluence automatically. We report an implementation of a ground confluence prover based on rewriting induction, which is a method originally developed for proving inductive theorems.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems, I.2.3 Deduction and Theorem Proving

Keywords and phrases Ground Confluence, Rewriting Induction, Non-Orientable Equations, Term Rewriting Systems

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.33

1 Introduction

Ground confluence of term rewriting systems (TRSs for short) guarantees that all ground terms are confluent. Not (general) confluence but ground confluence often matters in applications where not equational validity but *inductive validity* is of concern, including refutational completeness of inductive theorem proving and correctness of program transformations (e.g. [10, 11, 21]).

Classical works on ground confluence include [7, 16]. These works stem from *inductionless induction* which has been studied in the context of proving inductive validity by variations of Knuth-Bendix completion. Further studies on ground confluence orient for dealing with expressive rewrite rules over complex data structures, such as order-sorted signature, conditional rewrite rules and regular tree language constraints [7, 8, 15]. In this context, Bouhoula [8] reported on a tool for proving ground confluence. However, not only his procedure assumes reductivity of the system (a stronger notion of termination for conditional TRSs), but it also depends on a procedure for proving joinable inductive theorems which have to be dealt with a specialized proof procedure [9].

Confluence implies ground confluence but not vice versa as witnessed by:

▶ **Example 1.** Let $\mathcal{F} = \{\mathsf{plus} : \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat}, \mathsf{s} : \mathsf{Nat} \to \mathsf{Nat}, \mathsf{0} : \mathsf{Nat}\}$ and

$$\mathcal{R} = \left\{ \begin{array}{llllllll} \mathsf{plus}(\mathsf{0},\mathsf{0}) & \to & \mathsf{0} & (a) & \mathsf{plus}(\mathsf{s}(x),y) & \to & \mathsf{s}(\mathsf{plus}(x,y)) & (b) \\ \mathsf{plus}(x,\mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{plus}(y,x)) & (c) \end{array} \right\}$$

$\mathcal{R}$ is not confluent, as $\mathsf{s}(\mathsf{s}(\mathsf{plus}(y,x))) \leftarrow \mathsf{s}(\mathsf{plus}(x,\mathsf{s}(y))) \leftarrow \mathsf{plus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{s}(\mathsf{plus}(y,\mathsf{s}(x))) \to \mathsf{s}(\mathsf{s}(\mathsf{plus}(x,y)))$. However, $\mathcal{R}$ is ground confluent.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 33; pp. 33:1–33:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Recently, several confluence provers have been developed (e.g. [6, 23, 17, 27]) and results in automatable techniques for confluence proving. In contrast, it seems that little interest has been paid to the development of tools for proving ground confluence automatically.

In this paper, we report an implementation of a ground confluence prover. Key features of our prover are as follows:

- It is based on a simple framework: our framework is many-sorted (first-order) TRSs; considering not uni-sorted but many-sorted signature is a minimal requirement for natural setting of the problem as it involves inductive arguments.
- It requires a minimal input: the input of our tool is only a description of a many-sorted TRS. In particular, we do not assume the ordering for making systems reductive and a partition of function symbols into constructors and defined symbols—this is in contrast to the setting often found in the literature of ground confluence [7, 8, 15].
- It employs a simple method: our tool is based on rewriting induction, which is nowadays a well-understood method for inductive theorem proving (e.g. [1]). We anticipate it should be easy to develop similar (or even more sophisticated) tools based on our method.

Furthermore, we have prepared a collection of examples which can be used to estimate the status of the power of ground confluence proving tools.

2 Preliminaries

We assume basic familiarity with (many-sorted) term rewriting (e.g. [24]).

We use $\uplus$ for the disjoint union and $\setminus$ for the subtraction. The transitive reflexive (reflexive, symmetric, reflexive symmetric, equivalence) closure of a relation $\to$ is denoted by $\xrightarrow{*}$ (resp. $\xrightarrow{=}$, $\leftrightarrow$, $\overset{=}{\leftrightarrow}$, $\overset{*}{\leftrightarrow}$). For any quasi-order $\succsim$, we put $\succ = \succsim \setminus \precsim$ and $\approx = \succsim \cap \precsim$. A quasi-order $\succsim$ is *well-founded* if so is its strict part $\succ$. We abuse a set notation $\{a_1, \ldots, a_n\}$ with multisets. The *multiset extension* of a partial order $\succ$ is denoted by $\succ_m$.

Let $\mathcal{S}$ be a set of *sorts*. Each *many-sorted function* f is equipped with its *sort declaration* $f : \alpha_1 \times \cdots \times \alpha_n \to \alpha_0$, where $\alpha_0, \ldots, \alpha_n \in \mathcal{S}$ $(n \geq 0)$; the *arity* n is denoted by $ar(f)$. The set of *terms* over the set of many-sorted function symbols $\mathcal{F}$ and the set of variables $\mathcal{V}$ is denoted by $\mathrm{T}(\mathcal{F}, \mathcal{V})$. The set of function symbols (variables) contained in a term t is denoted by $\mathcal{F}(t)$ (resp. $\mathcal{V}(t)$). The set of *ground terms* over $\mathcal{G} \subseteq \mathcal{F}$ is denoted by $\mathrm{T}(\mathcal{G})$. The set of *positions* of a term t is denoted by $\mathrm{Pos}(t)$. The *empty* position is denoted by ϵ. The symbol in t at position p is denoted by $t(p)$.

A *context* is a term containing a special constant $\Box$, called a *hole*. Let C be a context containing precisely one hole. Then the term obtained from C by replacing the hole with t is denoted by $C[t]$. A *substitution* is a mapping from $\mathcal{V}$ to $\mathrm{T}(\mathcal{F}, \mathcal{V})$. A *ground substitution* is a mapping from $\mathcal{V}$ to $\mathrm{T}(\mathcal{F})$. For substitutions σ, usually it is required that the domain $\{x \in \mathcal{V} \mid \sigma(x) \neq x\}$ of σ is finite, but we omit that condition to ease the notation so that $t\sigma_g$ is a ground term for any term t and ground substitution σ_g. A *most general unifier* of terms s and t is denoted by $\mathrm{mgu}(s, t)$. A *rewrite relation (quasi-order)* is a relation (resp. quasi-order) on terms *closed under contexts and substitutions*. A rewrite relation (quasi-order) is a *reduction* relation (resp. quasi-order) if it is well-founded.

(Indirected) *equations* $l \doteq r$ and $r \doteq l$ are identified. A *directed equation* is denoted by $l \to r$. For a set E of equations (directed equations) the smallest rewrite relation containing E is denoted by $\leftrightarrow_E$ (resp. $\to_E$). For a set of directed equations E, let $\mathrm{LHS}(E) = \{l \mid l \to r \in E\}$ and $\mathrm{LHS}(f, E) = \{l \mid l \to r \in E, l(\epsilon) = f\}$ for each $f \in \mathcal{F}$. A directed equation $l \to r$ is a *rewrite rule* if $l \notin \mathcal{V}$ and $\mathcal{V}(l) \supseteq \mathcal{V}(r)$ hold. A (many-sorted) *term rewriting system* (*TRS* for short) is a finite set of rewrite rules. The set of $\mathcal{R}$-normal forms is denoted by $\mathrm{NF}(\mathcal{R})$. The set of *critical pairs* of a TRS $\mathcal{R}$ is denoted by $\mathrm{CP}(\mathcal{R})$.

A TRS $\mathcal{R}$ is *terminating* if $\to_{\mathcal{R}}$ is well-founded. Terms s and t are *joinable* w.r.t. the rewrite relation $\to_{\mathcal{R}}$ (denoted by $s \downarrow_{\mathcal{R}} t$) if $s \xrightarrow{*}_{\mathcal{R}} u$ and $t \xrightarrow{*}_{\mathcal{R}} u$ for some u. A TRS $\mathcal{R}$ is *(ground) confluent* if $s \downarrow_{\mathcal{R}} t$ holds for any (ground) terms s, t such that $u \xrightarrow{*}_{\mathcal{R}} s$ and $u \xrightarrow{*}_{\mathcal{R}} t$ for some (resp. ground) term u. Terms s and t are *ground convertible* if $s\sigma_g \overset{*}{\leftrightarrow}_{\mathcal{R}} t\sigma_g$ holds for any ground substitution σ_g. An equation $s \doteq t$ is an *inductive theorem* of a TRS $\mathcal{R}$, or *inductively valid* in $\mathcal{R}$, if s and t are ground convertible. We write $\mathcal{R} \models_{ind} E$ for a set E of equations if every $s \doteq t \in E$ is an inductive theorem. Let $\succsim$ be a rewrite quasi-order. We write $s \overset{*}{\leftrightarrow}_{\precsim\mathcal{R}} t$ if there exists $s = u_0 \leftrightarrow u_1 \leftrightarrow_{\mathcal{R}} \cdots \leftrightarrow_{\mathcal{R}} u_n = t$ such that $s \succsim u_i$ or $t \succsim u_i$ for every u_i $(1 \leq i \leq n)$. Terms s and t are *bounded ground convertible* w.r.t. $\succsim$ if $s\sigma_g \overset{*}{\leftrightarrow}_{\precsim\mathcal{R}} t\sigma_g$ holds for any ground substitution σ_g. A set E of equations is bounded ground convertible if s and t are bounded ground convertible for any $s \doteq t \in E$.

The next lemma is a direct consequence of a generalized Newman's Lemma [26]; see Exercise 1.3.12 in [24].

▶ **Lemma 2.** *Let $\succsim$ be a reduction quasi-order and $\mathcal{R}$ be a TRS such that $\mathcal{R} \subseteq \succ$. If* $\mathrm{CP}(\mathcal{R})$ *is bounded ground convertible w.r.t. $\succsim$, then $\mathcal{R}$ is ground confluent.*

We consider a partition of function symbols into the set $\mathcal{D}$ of *defined symbols*, and the set $\mathcal{C}$ of *constructors* i.e. $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$. Terms in $\mathrm{T}(\mathcal{C}, V)$ are *constructor terms.* Then a mapping from $\mathcal{V}$ to $\mathrm{T}(\mathcal{C})$ is called a *ground constructor substitution.* A term of the form $f(c_1, \ldots, c_n)$ for some $f \in \mathcal{D}$ and $c_1, \ldots, c_n \in \mathrm{T}(\mathcal{C}, \mathcal{V})$ is said to be *basic.* The set of basic subterms of s is written as $\mathcal{B}(s)$. A TRS $\mathcal{R}$ is said to be *quasi-reducible* if no ground basic terms are normal. Clearly, if $\mathcal{R}$ is a quasi-reducible terminating TRS then for any ground term s there exists t such that $s \xrightarrow{*} t \in \mathrm{T}(\mathcal{C})$.

3 Rewriting Induction for Ground Confluence

We now provide a background theory of our tool. In this section, we put $\mathcal{D} = \{l(\epsilon) \mid l \to r \in \mathcal{R}\}$ and $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$. The fundamental ingredient of our ground confluence prover is the following inference system of rewriting induction.

▶ **Definition 3** (rewriting induction for ground confluence)**.** The input of a rewriting induction procedure is a TRS $\mathcal{R}$, a set E of equations and a reduction quasi-order $\succsim$ such that $\mathcal{R} \subseteq \succ$. In Figure 1, we list the inference rules of the rewriting induction that act on pairs of a set of equations and a set of directed equations. We write $\langle E, H \rangle \rightsquigarrow \langle E', H' \rangle$ when an inference rule is applied (from upper to lower). The procedure (non-deterministically) generates a derivation starting from $\langle E, \emptyset \rangle$. If the derivation ends with some $\langle \emptyset, H \rangle$ (i.e. $\langle E, \emptyset \rangle \overset{*}{\rightsquigarrow} \langle \emptyset, H \rangle$ for some H), then the procedure succeeds. The procedure fails if there are no inference rules to apply. The derivation may also diverge.

In the figure, we use $\circ$ for the composition of relations and Expd is defined as:

$$\mathrm{Expd}_u(s, t) = \{C[r]\sigma \to t\sigma \mid s = C[u], \sigma = \mathrm{mgu}(u, l), l \to r \in \mathcal{R}\}.$$

For a set H of directed equations and a reduction quasi-order $\succsim$, we let

$$\mathrm{inv}(H) = \{r \to l \mid l \to r \in H\} \quad \text{and} \quad H^{\diamond} = \{l\sigma \to r\sigma \mid l \to r \in H, l\sigma \diamond r\sigma\}$$

where $\diamond \in \{\succ, \succsim\}$. Note that, for each $\diamond \in \{\succ, \succsim\}$, $s \to_{H^{\diamond}} t$ iff $s \to_H t$ and $s \diamond t$.

▶ Remark. In contrast to usual rewriting induction system for proving inductive theorems (see e.g. [1]), $s \succ t$ is not assumed in *Expand* rule. The point is essential to deal with

Expand

$$\frac{\langle E \uplus \{s \doteq t\},\ H\rangle}{\langle E \cup \{s'_i \doteq t_i\}_i,\ H \cup \{s \to t\}\rangle} \quad \begin{array}{l} u \in \mathcal{B}(s), \{s_i \to t_i\}_i = \mathrm{Expd}_u(s,t), \\ s_i \xrightarrow{*}_{H \cup \mathrm{inv}(H)^{\succsim}} s'_i \end{array}$$

Simplify

$$\frac{\langle E \uplus \{s \doteq t\},\ H\rangle}{\langle E \cup \{s' \doteq t\},\ H\rangle} \quad s \to_{\mathcal{R} \cup H^{\succ}} \circ \xrightarrow{*}_{H \cup \mathrm{inv}(H)^{\succsim}} s'$$

Delete

$$\frac{\langle E \uplus \{s \doteq t\},\ H\rangle}{\langle E,\ H\rangle} \quad s \overset{=}{\leftrightarrow}_H t$$

Figure 1 Inference rules of rewriting induction.

non-orientable equations. In the system of [2], $s \to_{H \cup \mathrm{inv}(H)^{\succ}} s'$ is allowed in *Simplify* rule, but here only $s \to_{H^{\succ}} s'$ is allowed. Compared to the system in [13], elements of H are directed equations to keep record of information on which side is expanded in *Expand* rule. These modifications are required to guarantee the bounded ground convertibility.

▶ **Example 4.** Let us consider $\mathcal{R}$ of Example 1. Then, we have the following rewriting induction derivation of from $\langle E_0, \emptyset\rangle$ where $E_0 = \mathrm{CP}(\mathcal{R})$.

$$\begin{array}{rl}
 & \langle\{\mathsf{s}(\mathsf{plus}(x,\mathsf{s}(y))) \doteq \mathsf{s}(\mathsf{plus}(y,\mathsf{s}(x))\},\ \emptyset\rangle \\
\overset{*}{\rightsquigarrow}_{\mathit{Simplify}} & \langle\{\mathsf{s}(\mathsf{s}(\mathsf{plus}(y,x))) \doteq \mathsf{s}(\mathsf{s}(\mathsf{plus}(x,y)))\},\ \emptyset\rangle \\
\rightsquigarrow_{\mathit{Expand}} & \langle\{\mathsf{s}(\mathsf{s}(0)) \doteq \mathsf{s}(\mathsf{s}(0)),\ \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{plus}(y',x)))) \doteq \mathsf{s}(\mathsf{s}(\mathsf{plus}(x,\mathsf{s}(y')))), \\
 & \qquad\qquad\qquad\qquad \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{plus}(x',y)))) \doteq \mathsf{s}(\mathsf{s}(\mathsf{plus}(\mathsf{s}(x'),y)))\}, \\
 & \ \{\mathsf{s}(\mathsf{s}(\mathsf{plus}(y,x))) \to \mathsf{s}(\mathsf{s}(\mathsf{plus}(x,y)))\}\rangle \\
\overset{*}{\rightsquigarrow}_{\mathit{Simplify}} & \langle\{\mathsf{s}(\mathsf{s}(0)) \doteq \mathsf{s}(\mathsf{s}(0)),\ \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{plus}(y',x)))) \doteq \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{plus}(y',x)))), \\
 & \qquad\qquad\qquad\qquad \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{plus}(x',y)))) \doteq \mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{plus}(x',y))))\}, \\
 & \ \{\mathsf{s}(\mathsf{s}(\mathsf{plus}(y,x))) \to \mathsf{s}(\mathsf{s}(\mathsf{plus}(x,y)))\}\rangle \\
\overset{*}{\rightsquigarrow}_{\mathit{Delete}} & \langle\emptyset,\ \{\mathsf{s}(\mathsf{s}(\mathsf{plus}(y,x))) \to \mathsf{s}(\mathsf{s}(\mathsf{plus}(x,y)))\}\rangle
\end{array}$$

Then, the rewriting induction procedure returns success.

Henceforth, we assume a reduction quasi-order $\succsim$ such that $\mathcal{R} \subseteq {\succ}$—using standard techniques in automated termination proof of TRSs, such a reduction quasi-order can be searched efficiently using SAT-solvers (e.g. [12]).

A rewriting induction derivation $\langle E_0, H_0\rangle \rightsquigarrow \langle E_1, H_1\rangle \rightsquigarrow \cdots$ is said to be *fair* if $\bigcup_{j \geq 0} \bigcap_{i \geq j} E_i = \emptyset$. In the following lemmas, let us fix a fair derivation $\langle E_0, H_0\rangle \rightsquigarrow \langle E_1, H_1\rangle \rightsquigarrow \cdots$ with $H_0 = \emptyset$. Let $E_\infty = \bigcup_i E_i$. As $H_0 = \emptyset$, it easily follows $H_i \subseteq E_\infty$ from the inference rules of rewriting induction. The next relations are used to characterize an ordering constraint induced by fair derivations.

▶ **Definition 5.** Let $\succsim$ be a well-founded quasi-order, $\underset{1}{\to}, \underset{2}{\to}$ binary relations, and $\underset{1,2}{\leftrightarrow} = \underset{1}{\leftrightarrow} \cup \underset{2}{\leftrightarrow}$.

- $x \overset{*}{\leftrightarrow}_{\succ 2}\, y$ iff there exists $x = x_0 \underset{2}{\leftrightarrow} x_1 \underset{2}{\leftrightarrow} \cdots \underset{2}{\leftrightarrow} x_n = y$ such that $\{x, y\} \succ_m \{x_i, x_{i+1}\}$ for every $x_i \underset{2}{\leftrightarrow} x_{i+1}$.
- $x \overset{*}{\leftrightarrow}_{\succsim 1, \succ 2}\, y$ iff there exists $x = x_0 \underset{1,2}{\leftrightarrow} x_1 \underset{1,2}{\leftrightarrow} \cdots \underset{1,2}{\leftrightarrow} x_n = y$ such that $x \succsim x_i$ or $y \succsim x_i$ for every x_i and $\{x, y\} \succ_m \{x_i, x_{i+1}\}$ for every $x_i \underset{2}{\leftrightarrow} x_{i+1}$.

Note that $\{x, y\} \succ_m \{x_i, x_{i+1}\}$ implies $x \succsim x_i$ or $y \succsim x_i$ (and $x \succsim x_{i+1}$ or $y \succsim x_{i+1}$).

Before showing the characterization of fair rewriting induction derivations, let us show a consequence of the ordering constraint.

▶ **Lemma 6** (conversion lemma). *Suppose* $x \underset{2}{\leftrightarrow} y$ *implies* $x \overset{*}{\leftrightarrow}_{\succsim 1, \succ 2} y$ *for any* x, y. *Then,* $x \underset{2}{\leftrightarrow} y$ *implies* $x \overset{*}{\leftrightarrow}_{\succsim 1} y$ *for any* x, y.

Proof. By induction on the multiset $\{x, y\}$ w.r.t. $\succ_m$. Suppose $x \underset{2}{\leftrightarrow} y$. Then by our assumption, $x = x_0 \underset{1,2}{\leftrightarrow} x_1 \underset{1,2}{\leftrightarrow} \cdots \underset{1,2}{\leftrightarrow} x_n = y$ for some $x_0, \ldots, x_n$ $(n \leq 0)$. If $x_i \underset{1}{\leftrightarrow} x_{i+1}$ for all $0 \leq i \leq n-1$, then the claim follows. Suppose $x_i \underset{2}{\leftrightarrow} x_{i+1}$ for some i. Then, by our assumption, $\{x, y\} \succ_m \{x_i, x_{i+1}\}$ holds. Thus, by induction hypothesis, $x_i \overset{*}{\leftrightarrow}_{\succsim 1} x_{i+1}$ holds. Let $x_i = z_0 \underset{1}{\leftrightarrow} z_1 \underset{1}{\leftrightarrow} \cdots \underset{1}{\leftrightarrow} z_p = x_{i+1}$. Then, every z_j satisfies $x_i \succsim z_j$ or $x_{i+1} \succsim z_j$. Hence, we have $x \succsim z_j$ or $y \succsim z_j$ as $\{x, y\} \succ_m \{x_i, x_{i+1}\}$. Thus, by replacing each $x_i \underset{2}{\leftrightarrow} x_{i+1}$ by $x_i \overset{*}{\leftrightarrow}_{\succsim 1} x_{i+1}$, we obtain $x \overset{*}{\leftrightarrow}_{\succsim 1} y$. ◀

Thus the characterization is directly connected to bounded (ground) convertibility, and consequently, to ground confluence via Lemma 2. To show the characterization, we need the following property of the operation Expd [1].

▶ **Proposition 7** (property of Expd). *Suppose* $\mathcal{R}$ *is a quasi-reducible TRS and* $u \in \mathcal{B}(s)$. *Then, for any ground constructor substitution* σ_{gc}, *we have* $s\sigma_{gc} \to_{\mathcal{R}} \circ \to_{\mathrm{Expd}_u(s,t)} t\sigma_{gc}$.

We now prove the announced characterization.

▶ **Lemma 8.** *For any* $s \doteq t \in E_\infty$ *and ground substitution* σ_g, $s\sigma_g \overset{*}{\leftrightarrow}_{\succsim \mathcal{R}, \succ E_\infty} t\sigma_g$ *holds.*

Proof. If σ_g is not a ground constructor substitution on $\mathcal{V}(s) \cup \mathcal{V}(t)$, then $s\sigma_g \xrightarrow{+}_{\mathcal{R}} s\rho_g$ (or $t\sigma_g \xrightarrow{+}_{\mathcal{R}} t\rho_g$) for some ground constructor substitution ρ_g. Then, we have $s\sigma_g \xrightarrow{+}_{\mathcal{R}} s\rho_g \leftrightarrow_{E_\infty} t\rho_g \xleftarrow{*}_{\mathcal{R}} t\sigma_g$. Thus, from $\mathcal{R} \subseteq \succ$, one easily obtains $s\sigma_g \overset{*}{\leftrightarrow}_{\succsim \mathcal{R}, \succ E_\infty} t\sigma_g$. It remains to consider the case that σ_g is a ground constructor substitution. By the fairness assumption, some inference rule is applied to $s \doteq t$ in some step. We distinguish three cases by the inference rule applied. Note that $H \subseteq E_\infty$ in the following cases.

(Expand) Then we have $\langle E \uplus \{s \doteq t\}, H \rangle \rightsquigarrow \langle E \cup \{s'_i \doteq t_i\}_i, \{s \to t\} \cup H \rangle$, where $\mathrm{Expd}_u(s,t) = \{s_i \to t_i\}_i$, $u \in \mathcal{B}(s)$, and $s_i \xrightarrow{*}_{H \cup \mathrm{inv}(H)\succsim} s'_i$ for each i. Since σ_g is a ground constructor substitution, by Proposition 7, we have $s\sigma_g \to_{\mathcal{R}} s_i\theta_g \to_{\mathrm{Expd}_u(s,t)} t\sigma_g$ for some θ_g and i. Thus, $s\sigma_g \to_{\mathcal{R}} s_i\theta_g \xrightarrow{*}_{H \cup \mathrm{inv}(H)\succsim} s'_i\theta_g \leftrightarrow_{E_\infty} t\sigma_g$ for each i. By $\mathcal{R} \subseteq \succ$, we have $s\sigma_g \succ s_i\theta_g$. Then, for any step $u_g \leftrightarrow_H v_g$ in $s_i\theta_g \overset{*}{\leftrightarrow}_{H \cup \mathrm{inv}(H)\succsim} s'_i\theta_g$, we have $s\sigma_g \succ u_g, v_g$, and hence $\{s\sigma_g, t\sigma_g\} \succ_m \{u_g, v_g\}$. Thus, $s\sigma_g \overset{*}{\leftrightarrow}_{\succsim \mathcal{R}, \succ E_\infty} t\sigma_g$.

(Simplify) Then we have $\langle E \uplus \{s \doteq t\}, H \rangle \rightsquigarrow \langle E \cup \{s' \doteq t\}, H \rangle$ for some E, H, where $s \to_{\mathcal{R} \cup H\succ} \hat{s} \xrightarrow{*}_{H \cup \mathrm{inv}(H)\succsim} s'$. Then, $s \to_{\mathcal{R} \cup H\succ} \hat{s} = s_1 \leftrightarrow_H s_2 \leftrightarrow_H \cdots \leftrightarrow_H s_k = s' \leftrightarrow_{E_\infty} t$ with $s_i \succsim s_{i+1}$ for $i = 1, \ldots, k-1$. We distinguish two cases.

1. Case $s \to_{\mathcal{R}} \hat{s}$. Then by $\mathcal{R} \subseteq \succ$, $s\sigma_g \succ \hat{s}\sigma_g$ and thus $s\sigma_g \succ s_i\sigma_g$ for $i = 1, \ldots, k$. Hence, we have $\{s\sigma_g, t\sigma_g\} \succ_m \{s_i\sigma_g, s_{i+1}\sigma_g\}$ for $i = 1, \ldots, k-1$ and $\{s\sigma_g, t\sigma_g\} \succ_m \{s'\sigma_g, t\sigma_g\}$. Thus, $s\sigma_g \overset{*}{\leftrightarrow}_{\succsim \mathcal{R}, \succ E_\infty} t\sigma_g$.
2. Case $s \to_{H\succ} \hat{s}$. Then $s \leftrightarrow_{E_\infty} \hat{s}$ with $s \succ \hat{s}$ and $s\sigma_g \succ s_i\sigma_g$ for all $i = 1, \ldots, k$. Hence, we have $\{s\sigma_g, t\sigma_g\} \succ_m \{s_i\sigma_g, s_{i+1}\sigma_g\}$ for $i = 1, \ldots, k-1$ and $\{s\sigma_g, t\sigma_g\} \succ_m \{s'\sigma_g, t\sigma_g\}$. It remains to show there exists $s \overset{*}{\leftrightarrow}_{\mathcal{R} \cup E_\infty} \hat{s}$ such that $s\sigma_g \succ u_g$ or $t \succ u_g$ for any midpoint u_g in $s \overset{*}{\leftrightarrow}_{\mathcal{R} \cup E_\infty} \hat{s}$ and $\{s\sigma_g, t\sigma_g\} \succ_m \{u_g, v_g\}$ for any $u_g \leftrightarrow_{E_\infty} v_g$ in $s \overset{*}{\leftrightarrow}_{\mathcal{R} \cup E_\infty} \hat{s}$. By $s \to_{H\succ} \hat{s}$, there exists $w \to \hat{w} \in H$ such that $s = C[w\theta]$ and $\hat{s} = C[\hat{w}\theta]$ for some context

C and substitution θ. If $\theta_g = \sigma_g \circ \theta$ is not a constructor ground substitution on $\mathcal{V}(w) \cup \mathcal{V}(\hat{w})$, then the claim follows as in the case σ_g is not a constructor ground substitution. Thus, suppose otherwise. Then, as *Expand* rule is applied to $w \doteq \hat{w}$ with $u \in \mathcal{B}(w)$, it follows using Proposition 7 that $w\theta_g \to_{\mathcal{R}} \circ \overset{*}{\rightarrow}_{H\cup \mathrm{inv}(H)^{\succsim}} \circ \leftrightarrow_{E_\infty} \hat{w}\theta_g$. Hence, $s\sigma_g = C[w\theta]\sigma_g = C\sigma_g[w\theta_g] \to_{\mathcal{R}} w_g \overset{*}{\rightarrow}_{H\cup \mathrm{inv}(H)^{\succsim}} \circ \leftrightarrow_{E_\infty} C\sigma_g[\hat{w}\theta_g] = C[\hat{w}\theta]\sigma_g = \hat{s}\sigma_g$ Then $s\sigma_g \succ w_g$. Furthermore, for any step $u_g \leftrightarrow_H v_g$ in $w_g \overset{*}{\rightarrow}_{H\cup \mathrm{inv}(H)^{\succsim}} \circ \leftrightarrow_{E_\infty} \hat{s}\sigma_g$, we have $\{s\sigma_g, t\sigma_g\} \succ_m \{u_g, v_g\}$. Thus, one obtains $s\sigma_g \overset{*}{\leftrightarrow}_{\succsim\mathcal{R},\succ E_\infty} t\sigma_g$.

(Delete) Then we have $\langle E \uplus \{s \doteq t\}, H\rangle \rightsquigarrow \langle E, H\rangle$, where $s = t$ or $s \leftrightarrow_H t$. The case $s = t$ is obvious. If $s \leftrightarrow_H t$ then there exists $s' \doteq t' \in E_\infty$ such that $s = C[s'\theta]$, $t = C[r'\theta]$ for some θ, and *Expand* is applied to $s' \doteq t'$. Thus $s\sigma_g \overset{*}{\leftrightarrow}_{\succsim\mathcal{R},\succ E_\infty} t\sigma_g$ is shown in the same way as the case (*Simplify*)-b. ◀

▶ **Lemma 9.** *If $\langle E_0, \emptyset\rangle \overset{*}{\rightsquigarrow} \langle \emptyset, H\rangle$ then E_0 is bounded ground convertible.*

Proof. We have $\mathcal{R} \subseteq \succ$. Clearly, the derivation $\langle E_0, \emptyset\rangle \overset{*}{\rightsquigarrow} \langle \emptyset, H\rangle$ is fair. Thus, by Lemma 8, for any $s \doteq t \in E_\infty$, ground substitution σ_g and ground context C, $C[s\sigma_g] \leftrightarrow_{E_\infty} C[t\sigma_g]$ implies $C[s\sigma_g] \overset{*}{\leftrightarrow}_{\succsim\mathcal{R},\succ E_\infty} C[t\sigma_g]$. Hence, by Lemma 6, $C[s\sigma_g] \leftrightarrow_{E_\infty} C[t\sigma_g]$ implies $C[s\sigma_g] \overset{*}{\leftrightarrow}_{\succsim\mathcal{R}} C[t\sigma_g]$. The claim follows as $E_0 \subseteq E_\infty$. ◀

▶ Remark. If E_0 is bounded ground convertible then $\mathcal{R} \models_{ind} E_0$. Thus, the lemma above implies the correctness of our rewriting induction system as an inductive theorem proving procedure. In particular, the soundness of the basic rewriting induction system (e.g. Figure 1 of [1]) follows. For this system several correctness proofs are known: e.g. the one using minimal counterexample [13] and the one based on retrogressive property [1].

By Lemmas 2 and 9, our method for ground confluence proving is obtained.

▶ **Theorem 10** (ground confluence check by rewriting induction)**.** *Let $\mathcal{D} = \{l(\epsilon) \mid l \to r \in \mathcal{R}\}$, $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$, $\mathcal{R}$ a quasi-reducible TRS, and $\succsim$ a reduction quasi-order such that $\mathcal{R} \subseteq \succ$. If $\langle \mathrm{CP}(\mathcal{R}), \emptyset\rangle \overset{*}{\rightsquigarrow} \langle \emptyset, H\rangle$ for some H, then $\mathcal{R}$ is ground confluent.*

4 Relaxing the Free Constructor Restriction

In the previous section, we have fixed a partition $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ as $\mathcal{D} = \{l(\epsilon) \mid l \to r \in \mathcal{R}\}$ and $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$. This setting and the quasi-reducibility assumption on $\mathcal{R}$ induce a rather strong constraint on $\mathcal{R}$. To see this, consider the following example.

▶ **Example 11.** Let

$$\mathcal{R} = \left\{ \begin{array}{lll} \mathsf{plus}(\mathsf{0}, y) & \to & y, \quad \mathsf{plus}(\mathsf{s}(\mathsf{0}), y) \to \mathsf{s}(y), \quad \mathsf{s}(\mathsf{s}(x)) \to x \end{array} \right\}.$$

Then $\{l(\epsilon) \mid l \to r \in \mathcal{R}\} = \{\mathsf{plus}, \mathsf{s}\}$. However, if we put $\mathcal{D} = \{\mathsf{plus}, \mathsf{s}\}$ and $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$, then $\mathsf{s}(\mathsf{0})$ is a basic ground term which is a normal form. Then $\mathcal{R}$ is not quasi-reducible. On the other hand, we could have put $\mathcal{D} = \{\mathsf{plus}\}$ and $\mathcal{C} = \{\mathsf{s}, \mathsf{0}\}$. In that case $\mathcal{R}$ is quasi-reducible.

In other words, the partition $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ in the previous section can deal with only TRSs $\mathcal{R}$ having *free constructors*, i.e. the case $\mathrm{T}(\mathcal{C}) \subseteq \mathrm{NF}(\mathcal{R})$. In this section, we relax this restriction. From now on, we assume some partition $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ has been fixed, and the set of rules $l \to r \in \mathcal{R}$ satisfying $l(\epsilon) \notin \mathcal{D}$ is denoted by $\mathcal{R}_c$. To extend the notion of quasi-reducibility to deal with TRSs with non-free constructors, we first prepare several notions.

▶ **Definition 12** (cover)**.** A set L of terms *covers* a term t if for any ground constructor substitution σ_{gc}, there exists $l \in L$ such that $\exists\theta.\, t\sigma_{gc} = l\theta$.

The set L (and its variants) appears under various names in the literature [10, 19, 20].

▶ **Definition 13** (pattern)**.** For any $f \in \mathcal{D}$, we put

$$\mathrm{Pat}(f, \mathcal{R}_c) = \{f(x_1, \ldots, x_{i-1}, w, x_{i+1}, \ldots, x_n) \mid 1 \leq i \leq n, w \in \mathrm{LHS}(\mathcal{R}_c)\}.$$

where $n = ar(f)$ and $x_1, \ldots, , x_n$ are pairwise distinct variables not in w.

The notion of quasi-reducibility is replaced as follows for arbitrary partition of $\mathcal{D} \uplus \mathcal{C}$.

▶ **Definition 14** (strongly quasi-reducible)**.** A TRS $\mathcal{R}$ is said to be a *strongly quasi-reducible* if for each $f \in \mathcal{D}$, $\mathrm{LHS}(f, \mathcal{R}) \cup \mathrm{Pat}(f, \mathcal{R}_c)$ covers $f(x_1, \ldots, x_n)$.

▶ **Example 15** (checking strong quasi-reduciblity)**.** Consider $\mathcal{R}$ in Example 11. Take $\mathcal{C} = \{\mathsf{s}, \mathsf{0}\}$ and $\mathcal{D} = \{\mathsf{plus}\}$. Then $\mathcal{R}_c = \{\mathsf{s}(\mathsf{s}(x)) \to x\}$. We have $\mathrm{LHS}(\mathsf{plus}, \mathcal{R}) = \{\mathsf{plus}(\mathsf{0}, y), \mathsf{plus}(\mathsf{s}(\mathsf{0}), y)\}$ and $\mathrm{Pat}(f, \mathcal{R}_c) = \{\mathsf{plus}(\mathsf{s}(\mathsf{s}(x)), y), \mathsf{plus}(x, \mathsf{s}(\mathsf{s}(y)))\}$. Now one can check $\{\mathsf{plus}(\mathsf{0}, y),\ \mathsf{plus}(\mathsf{s}(\mathsf{0}), y),\ \mathsf{plus}(\mathsf{s}(\mathsf{s}(x)), y),\ \mathsf{plus}(x, \mathsf{s}(\mathsf{s}(y)))\}$ covers $\mathsf{plus}(x, y)$, and thus $\mathcal{R}$ is strongly quasi-reducible.

The following lemma easily follows from these definitions.

▶ **Lemma 16.** *Any strongly quasi-reducible TRS is quasi-reducible.*

We also have to replace the operation Expd in our rewriting induction system.

▶ **Definition 17** ($\otimes$, Expd_u)**.** Let, for any $f \in \mathcal{D}_f$,

$$\mathcal{R}_c \otimes f = \{f(x_1, \ldots, l', \ldots, x_n) \to f(x_1, \ldots, r', \ldots, x_n) \mid l' \to r' \in \mathcal{R}_c\},$$

where $x_1, \ldots, x_n$ are supposed to be distinct variables not in l', r'. Let $s = C[u]$, $u \in \mathcal{B}(s)$ and $u(\epsilon) = f$. We put

$$\mathrm{Expd}_u(s, t) \quad = \quad \{C[r]\mu \to t\mu \mid \mu = \mathrm{mgu}(l, u), l \to r \in (\mathcal{R} \setminus \mathcal{R}_c) \cup (\mathcal{R}_c \otimes f)\}.$$

Note $l \to_{\mathcal{R}_c} r$ for any $l \to r \in \mathcal{R}_c \otimes f$ by definition.

▶ **Example 18.** Let $\mathcal{R}$ be in Example 11, and $s = \mathsf{plus}(x, \mathsf{s}(\mathsf{0}))$ and $t = \mathsf{plus}(\mathsf{s}(x), \mathsf{0})$. Then $\mathrm{Expd}_s(s, t) = \{\mathsf{s}(\mathsf{0}) \doteq \mathsf{plus}(\mathsf{s}(\mathsf{0}), \mathsf{0}),\ \mathsf{s}(\mathsf{s}(\mathsf{0})) \doteq \mathsf{plus}(\mathsf{s}(\mathsf{s}(\mathsf{0})), \mathsf{0}),\ \mathsf{plus}(y, \mathsf{s}(\mathsf{0})) \doteq \mathsf{plus}(\mathsf{s}(\mathsf{s}(\mathsf{s}(y))), \mathsf{0})\}$.

▶ **Lemma 19** (property of Expd)**.** *Suppose $\mathcal{R}$ is a strongly quasi-reducible TRS and $u \in \mathcal{B}(s)$. Then, for any ground constructor substitution σ_{gc}, we have $s\sigma_{gc} \to_{\mathcal{R}} \circ \to_{\mathrm{Expd}_u(s,t)} t\sigma_{gc}$.*

Proof. Since $u \in \mathcal{B}(s)$, $u = f(u_1, \ldots, u_n)$ for some $f \in \mathcal{D}$ and constructor terms $u_1, \ldots, u_n$. Then, since $\mathcal{R}$ is strongly quasi-reducible, for any ground constructor substitution σ_{gc}, there exists $l \in \mathrm{LHS}(f, \mathcal{R}) \cup \mathrm{Pat}(f, \mathcal{R}_c)$ such that $u\sigma_{gc}$ is an instance of l. Then, since one can assume $\mathcal{V}(l) \cap \mathcal{V}(u) = \emptyset$, w.l.o.g. one can let $u\sigma_{gc} = l\sigma_{gc}$. Then, there exist substitutions $\mu = \mathrm{mgu}(u, l)$ and ground substitution θ_g such that $\sigma_{gc} = \theta_g \circ \mu$, and we have $s\sigma_{gc} = C[u]\sigma_{gc} = C\sigma_{gc}[u\sigma_{gc}] = C\sigma_{gc}[l\sigma_{gc}] \to_{\mathcal{R}} C\sigma_{gc}[r\sigma_{gc}] = C[r]\mu\theta_g \to_{\mathrm{Expd}_u(s,t)} t\mu\theta_g = t\sigma_{gc}$. ◀

Now by replacing Proposition 7 with Lemma 19, the next theorem follows in the same way as Theorem 10 for the rewriting induction using the Expd given in Definition 17.

▶ **Theorem 20** (ground confluence for strongly quasi-reducible TRSs)**.** *Let $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ be an arbitrary partition. Let $\mathcal{R}$ be a strongly quasi-reducible TRS, and $\succsim$ a reduction quasi-order such that $\mathcal{R} \subseteq \succ$. If $\langle \mathrm{CP}(\mathcal{R}), \emptyset \rangle \overset{*}{\rightsquigarrow} \langle \emptyset, H \rangle$ for some H, then $\mathcal{R}$ is ground confluent.*

It is easy to see that Theorem 20 generalizes theorem 10, because of the following observation:

▶ **Lemma 21.** *Suppose $\mathcal{D} = \{l(\epsilon) \mid l \to r \in \mathcal{R}\}$ and $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$. Then (1) $\mathcal{R}_c = \emptyset$ and (2) any quasi-reducible TRS is strongly quasi-reducible.*

Input: many-sorted TRS $\mathcal{R}$

Output: Success/Failure

1. Compute (possibly multiple) candidates for the partition $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ of function symbols.
2. Compute (possibly multiple) candidates for strongly quasi-reducible $\mathcal{R}_0 \subseteq \mathcal{R}$.
3. Choose one $\mathcal{R}_0$ from the candidates, and remove $\mathcal{R}_0$ from the candidates list. If no candidate remains, then choose another candidate of partition to go Step 2 if exists, or else return Failure.
4. Find a reduction quasi-order $\succsim$ such that $\mathcal{R}_0 \subseteq \succ$. If it fails, go to Step 3 to examine another candidate.
5. Run rewriting induction for proving bounded ground convertibility of $\mathrm{CP}(\mathcal{R}_0)$ with $\succsim$. If it fails, go to Step 3 to examine another candidate.
6. Run rewriting induction for proving $\mathcal{R}_0 \models_{ind} (\mathcal{R} \setminus \mathcal{R}_0)$. If it fails, go to Step 3 to examine another candidate. If it succeeds, to return Success.

Figure 2 A ground confluence proving procedure based on rewriting induction.

5 Ground Confluence Proving Procedure

Before presenting our procedure, we introduce the last ingredient of our ground confluence proof, which one can prove easily:

▶ **Theorem 22.** *Let $\mathcal{R}$ be a TRS. Suppose $\mathcal{R}_0 \subseteq \mathcal{R}$ is ground confluent. If $\mathcal{R}_0 \models_{ind} \mathcal{R} \setminus \mathcal{R}_0$ then $\mathcal{R}$ is ground confluent.*

Hence, we divide the problem of ground confluence of $\mathcal{R}$ into the problem of ground confluence of $\mathcal{R}_0$ and the problem of inductive validity $\mathcal{R}_0 \models_{ind} \mathcal{R} \setminus \mathcal{R}_0$. In Figure 2, we present our procedure for proving ground confluence of many-sorted TRSs.

▶ **Example 23** (strongly quasi-reducible subsets)**.** Consider a TRS

$$\mathcal{R} = \left\{ \begin{array}{lclllcll} \mathsf{plus}(0, y) & \to & y & (a) & \mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y)) & (b) \\ \mathsf{plus}(x, 0) & \to & \mathsf{plus}(0, x) & (c) & \mathsf{plus}(x, \mathsf{s}(y)) & \to & \mathsf{plus}(\mathsf{s}(y), x) & (d) \end{array} \right\}$$

Then $\{(a), (b)\}$, $\{(c), (d)\}$ and $\mathcal{R}$ are all strongly quasi-reducible. However, the only choice of $\mathcal{R}_0 = \{(a), (b)\}$ is successful for finding ground confluence proof in our method.

▶ **Example 24** (strongly quasi-reducible subsets)**.** Consider a TRS

$$\mathcal{R} = \left\{ \begin{array}{lclllcll} \mathsf{plus}(0, y) & \to & y & (a) & \mathsf{plus}(\mathsf{s}(0), y) & \to & \mathsf{s}(y) & (b) \\ \mathsf{plus}(\mathsf{plus}(x, y), z) & \to & \mathsf{plus}(x, \mathsf{plus}(y, z)) & (c) & \mathsf{s}(\mathsf{s}(x)) & \to & \mathsf{s}(x) & (e) \\ \mathsf{plus}(x, y) & \to & \mathsf{plus}(y, x) & (e) & \mathsf{s}(x) & \to & \mathsf{s}(\mathsf{s}(x)) & (f) \end{array} \right\}$$

Then $\{(a), (b), (e)\}$, $\{(a), (f)\}$ and $\{(d), (f)\}$ are all strongly quasi-reducible, where the first one takes $\mathcal{C} = \{0, \mathsf{s}\}$ while the latter two take $\mathcal{C} = \{0\}$. However, the only choice of $\mathcal{R}_0 = \{(a), (b), (e)\}$ is successful for finding ground confluence proof in our method.

6 Implementation and Experiments

Our tool AGCP is written in SML/NJ. Some program code are incorporated from confluence prover ACP [6] and an inductive theorem prover [2]. The tool is obtained from

```
(FUN
    plus : Nat,Nat -> Nat
    s : Nat -> Nat
    0 : Nat
)
(VAR
    x : Nat
    y : Nat
)
(RULES
    plus(0,0) -> 0
    plus(s(x),y) -> s(plus(x,y))
    plus(x,s(y)) -> s(plus(y,x))
)
```

Figure 3 An example of input: specification of a many-sorted TRS.

http://www.nue.ie.niigata-u.ac.jp/tools/agcp/. The format of input TRSs basically follows old TPDB format of first-order TRSs; the only difference is the addition of sort declarations. In Figure 3, we present an example of input file.

Some heuristics implicit in the procedure are described below.

- We impose a timeout for each of Steps 1, 5 and 6.
- In Step 2, as the candidates for $\mathcal{R}_0 \subseteq \mathcal{R}$, we only take those with minimal set of rewrite rules i.e. those $\mathcal{R}_0$ such that any $\mathcal{R}_0' \subsetneq \mathcal{R}_0$ is not a quasi-reducible.
- In Step 4, we restrict ourselves to multiset path orderings based on total precedence. We encoded the condition $\mathcal{R}_0 \subseteq \succsim$ as a constraint on precedence and find a precedence which meets the condition by a SMT solver.
- In *Expand* inferences, among equations, one with the smallest size are expanded. If $l \succ r$ or $r \succ l$ holds, then the larger side is expanded. Among basic subterms, the older one is expanded. Here, a subterm is younger if it contains a newer created variable.
- In *Expand* and *Simplify* inferences, $\overset{*}{\to}_{H \cup \mathrm{inv}(H)^{\succsim}}$-part is tried only if successive applications of *Simplify* occur: just after *Expand*, the expanded side is simplified by $s\,(\overset{*}{\to}_{H \cup \mathrm{inv}(H)^{\succsim}} \circ \to_{\mathcal{R}_0 \cup H^{\succ}})^*\,s'$, and multiple *Simplify* steps are applied by $\to_{\mathcal{R}_0 \cup H^{\succ}} \circ (\overset{*}{\to}_{H \cup \mathrm{inv}(H)^{\succsim}} \circ \to_{\mathcal{R}_0 \cup H^{\succ}})^*$.

We have tested our ground confluence prover with 121 (many-sorted) TRSs—23 are constructed in the course of our study and 98 are incorporated from Cops[1]. Since Cops is a database of confluence problems, their signatures are unsorted. Also, there is no declaration of the signature. Hence, we have inspected the problems and identified a set of function symbols and attached them sorts naturally guessed. More than half Cops problems whose appropriate sorts are hardly imagined are dropped. Among newly constructed 23 problems, 4 problems are incorporated from the literature [7, 14, 18]. Others are constructed by starting with basic TRSs such as addition of natural numbers, and then add variations and extensions of them, trying to make them possibly ground confluent, where this collection includes Examples 1, 15, 23 and 24.

[1] Confluence Problem Database http://cops.uibk.ac.at/.

Table 1 Summary of experiments.

sources	number	success	timeout	time (msec)
Example 1	–	✓	–	8
Example 15	–	✓	–	8
Example 23	–	✓	–	11
Example 24	–	✓	–	10
Cops `confluent`	88	62	4	–
Cops `non_confluent`	3	1	0	–
Cops `!confluent !non_confluent`	7	3	0	–
Crafted	23	20	2	–

Tests are performed on a PC with one 2.50GHz CPU and 4G memory. We impose 60 (5, 1) seconds time limit total (resp. rewriting induction proof, computation of constructors).

Table 1 shows a summary of experiments. The columns below 'sources' and 'number' denote the number of TRSs and their sources. 'Cops `confluent`' ('Cops `non_confluent`', 'Cops `!confluent !non_confluent`') denotes problems incorporated from Cops problems that have been proved or non-proved (non-)confluent by state-of-the-art confluence provers. Note such confluence provers prove (non-)confluent of unsorted versions of the problems, but that of the many-sorted ones follow by persistency [5]. Similarly, among 'Crafted' problems, 11 problems are proved to be confluent and the others non-confluent by ACP [6]. The columns below 'Success' show results for each example in the present paper (✓ for success), and the numbers of problems from the collections that succeed. The columns below 'timeout' ('time (msec)') show the number of occurrences of timeout (run time shown in milliseconds, respectively).

Among 121 problems, our prover succeeded in proving ground confluence of 86 problems. Our procedures failed on some particular types of term rewriting systems. Firstly, those that have a defined symbol specified by non-terminating rules such as $\mathsf{nats} \to \mathsf{cons}(\mathsf{0}, \mathsf{inc}(\mathsf{nats}))$. In such a case, the non-terminating rule is not an inductive theorem and hence it is included to the rule part. However, in the current approach, the rule part needs to be terminating, and thus our procedure failed to deal with such a case. Similarly, if AC-rules needs to be act as rewrite rules, then our method does not work—our method can deal with AC-rules only if they are inductive theorems:

▶ **Example 25** (Cops 183).

$$\mathcal{R} = \left\{ \begin{array}{lcllcl} +(\mathsf{0}, x) & \to & x & +(x, \mathsf{0}) & \to & x \\ +(\mathsf{1}, -(\mathsf{1})) & \to & \mathsf{0} & +(-(\mathsf{1}), \mathsf{1}) & \to & \mathsf{0} \\ -(\mathsf{0}) & \to & \mathsf{0} & -(-(x)) & \to & x \\ -(+(x, y)) & \to & +(-(x), -(y)) & & & \\ +(+(x, y), z) & \to & +(x, +(y, z)) & +(x, y) & \to & +(y, x) \end{array} \right\}$$

Here, for example, the computation of $+(+(\mathsf{1}, \mathsf{1}), +(-(\mathsf{1}), -(\mathsf{1}))) \xrightarrow{*} \mathsf{0}$ needs to use AC-rules. Thus AC-rules are included in the rule part, and hence its termination proof fails. Our approach can not handle problems of this type.

Some failures are due to incapability of non-ground-confluence checking. We expect some simple non-ground-confluence check should be useful, but currently it is not included in our tool. It also seems inclusion of stronger termination criteria would have stopped some failures in early stages of proofs, and inclusion of lemma generation methods in inductive theorem proving would have solved at least one problem. Other reasons of failure include incapability of dealing with non-left-linear rules in strong quasi-reducibility checking.

Five timeouts are raised in rewriting induction proofs and one is in computation of constructor symbols. Two timeouts in a problem helped to switch the choice of $\mathcal{R}_0$ so as to succeed in that problem.

All details of the experiments are available on the webpage `http://www.nue.ie.niigata-u.ac.jp/tools/agcp/experiments/fscd16/`.

7 Conclusion

We have reported a ground confluence prover based on a variant of rewriting induction. We have also proved the correctness of our method. In contrast to many existing works on ground confluence, we focused on the pure many-sorted TRSs. Obviously, one can also use confluence provers such as [6, 27] to guarantee ground confluence, and to use more stronger inductive theorem proving methods or lemma generation methods such as [2, 3, 4, 22, 25] at inductive theorem proving part, in order to get a more powerful tool. Developing such a powerful tool stands as a long-term goal.

Acknowledgements. Thanks are due to the anonymous reviewers for helpful comments. This work is partially supported by JSPS KAKENHI Nos. 15K00003, 25280025.

References

1 T. Aoto. Dealing with non-orientable equations in rewriting induction. In *Proc. of 17th RTA*, volume 4098 of *LNCS*, pages 242–256. Springer-Verlag, 2006.

2 T. Aoto. Designing a rewriting induction prover with an increased capability of non-orientable equations. In *Proc. of 1st SCSS*, RISC Technical Report, pages 1–15, 2008.

3 T. Aoto. Sound lemma generation for proving inductive validity of equations. In *Proc. of 28th FSTTCS*, volume 2 of *LIPIcs*, pages 13–24. Schloss Dagstuhl, 2008.

4 T. Aoto and S. Stratulat. Decision procedures for proving inductive theorems without induction. In *Proc. of 16th PPDP*, pages 237–248. ACM Press, 2014.

5 T. Aoto and Y. Toyama. Persistency of confluence. *Journal of Universal Computer Science*, 3(11):1134–1147, 1997.

6 T. Aoto, Y. Yoshida, and Y. Toyama. Proving confluence of term rewriting systems automatically. In *Proc. of 20th RTA*, volume 5595 of *LNCS*, pages 93–102. Springer-Verlag, 2009.

7 K. Becker. Proving ground confluence and inductive validity in constructor based equational specifications. In *Proc. of 4th TAPSOFT*, volume 668 of *LNCS*, pages 46–60. Springer-Verlag, 1993.

8 A. Bouhoula. Simultaneous checking of completeness and ground confluence for algebraic specifications. *ACM Transactions on Computational Logic*, 10(2):20:1–33, 2009.

9 A. Bouhoula and F. Jacquemard. Verifying regular trace properties of secuirty protocols with explicit destructors and implicit induction. In *Proc. of FCS-ARSPA*, pages 27–44, 2007.

10 A. Bouhoula, E. Kounalis, and M. Rusinowitch. Automated mathematical induction. *Journal of Logic and Computation*, 5(5):631–668, 1995.

11 Y. Chiba, T. Aoto, and Y. Toyama. Program transformation by templates based on term rewriting. In *Proc. of 7th PPDP*, pages 59–69. ACM Press, 2005.

12 M. Codish, V. Lagoon, and P. J. Stuckey. Solving partial order constraints for LPO termination. In *Proc. of 17th RTA*, volume 4098 of *LNCS*, pages 4–18. Springer-Verlag, 2006.

13 N. Dershowitz and U. S. Reddy. Deductive and inductive synthesis of equational programs. *Journal of Symbolic Computation*, 15:467–494, 1993.

14 L. Fribourg. A strong restriction of the inductive completion procedure. *Journal of Symbolic Computation*, 8:253–276, 1989.

15 H. Ganzinger. Ground term confluence in parametric conditional equational specifications. In *Proc. of 4th STACS*, volume 247 of *LNCS*, pages 286–298, 1987.

16 R. Göbel. Ground confluence. In *Proc. of 2nd RTA*, volume 256 of *LNCS*, pages 156–167, 1987.

17 N. Hirokawa and D. Klein. Saigawa: A confluence tool. In *Proc. of 1st IWC*, page 49, 2012.

18 D. Kapur, P. Narendran, and F. Otto. On ground-confluence of term rewriting systems. *Information and Computation*, 86:14–31, 1990.

19 D. Kapur, P. Narendran, and H. Zhang. On sufficient-completeness and related properties of term rewriting systems. *Acta Informatica*, 24(4):395–415, 1987.

20 D. Kapur, P. Narendran, and H. Zhang. Automating inductionless induction using test sets. *Journal of Symbolic Computation*, 11(1–2):81–111, 1991.

21 K. Sato, K. Kikuchi, T. Aoto, and Y. Toyama. Correctness of context-moving transformations for term rewriting systems. In *Proc. of 25th LOPSTR*, volume 9527 of *LNCS*, pages 331–345. Springer-Verlag, 2015.

22 S. Shimazu, T. Aoto, and Y. Toyama. Automated lemma generation for rewriting induction with disproof. *JSSST Computer Software*, 26(2):41–55, 2009. In Japanese.

23 T. Sternagel and A. Middeldorp. Conditional confluence (system description). In *Proc. of Joint 25th RTA and 12th TLCA*, pages 456–465. Springer-Verlag, 2014.

24 Terese. *Term Rewriting Systems*. Cambridge University Press, 2003.

25 T. Walsh. A divergence critic for inductive proof. *Journal of Artificial Intelligence Research*, 4:209–235, 1996.

26 F. Winkler and B. Buchberger. A criterion for eliminating unnecessary reductions in the Knuth-Bendix algorithm. In *Proc. of the Colloq. on Algebra, Combinatorics and Logic in Computer Science, Vol. II*, pages 849–869. Springer-Verlag, 1985.

27 H. Zankl, B. Felgenhauer, and A. Middeldorp. CSI – A confluence tool. In *Proc. of 23rd CADE*, volume 6803 of *LNAI*, pages 499–505. Springer-Verlag, 2011.

Globular: An Online Proof Assistant for Higher-Dimensional Rewriting*

Krzysztof Bar[1], **Aleks Kissinger**[2], **and Jamie Vicary**[3]

1 **Department of Computer Science, University of Oxford, Oxford, UK**
krzysztof.bar@cs.ox.ac.uk
2 **iCIS, Radboud University Nijmegen, Nijmegen, The Netherlands**
aleks@cs.ru.nl
3 **Department of Computer Science, University of Oxford, Oxford, UK**
jamie.vicary@cs.ox.ac.uk

Abstract

This article introduces Globular, an online proof assistant for the formalization and verification of proofs in higher-dimensional category theory. The tool produces graphical visualizations of higher-dimensional proofs, assists in their construction with a point-and-click interface, and performs type checking to prevent incorrect rewrites. Hosted on the web, it has a low barrier to use, and allows hyperlinking of formalized proofs directly from research papers. It allows the formalization of proofs from logic, topology and algebra which are not formalizable by other methods, and we give several examples.

1998 ACM Subject Classification I.2.3 Deduction and Theorem Proving

Keywords and phrases higher category theory, higher-dimensional rewriting, interactive theorem proving

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.34

1 Introduction

This paper is a system description for Globular [20], an online tool for formalizing and verifying proofs in semistrict globular higher category theory. It operates from the perspective of *higher-dimensional rewriting*, with terms represented as graphical structures, and proofs constructed and visualized as sequences of rewrites on these structures. At the time of writing, the tool operates up to the level of 3-categories and is being actively developed (in parallel with the corresponding theory) to support for 4-categories and higher.

Globular is the first proof assistant of its kind, and it allows many proofs from higher category theory to be formalized, verified and visualized in a way that would not be practical in any other tool. The closest comparable tools are Quantomatic [7], which does diagrammatic rewriting for monoidal categories, and the formalisations of homotopy type theory in the proof assistants Coq and Agda [5]. The latter can indeed be used to perform logical and homotopy-theoretical proofs from a higher-categorical perspective. However, this approach diverges from ours in that it is based on the syntax of Martin-Löf type theory rather than diagrams, and identity types naturally lead one to treat higher-dimensional *invertible* structures (e.g. ∞-groupoids) as first-class citizens, rather than the more general structures we'll consider. Another comparable tool is *Orchard* [3], which allows the formalization of proofs in opetopic

* This work was supported by the European Research Council grant 320571.

(as opposed to globular) higher categories; this tool can handle ∞-categories, and has many attractive properties, although the opetopic approach to higher categories is far less dominant than the globular approach, and its associated graphical calculus is less attractive for many purposes. The higher dimensional rewriting implemented by Globular draws inspiration from the polygraphic approach to rewriting [10, 14], but extends it to allow for non-strict higher-categorical structures.

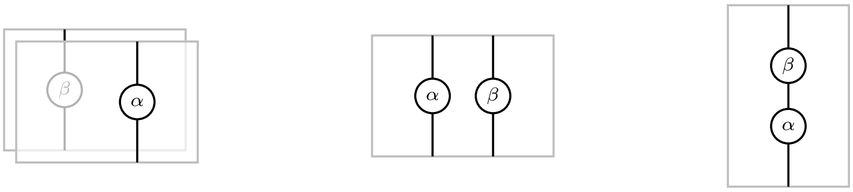
Globular was designed to make it as quick and easy as possible for users to go from zero to proving theorems and sharing proofs. Hence, it is entirely web-based, with all logic taking place client-side in the user's web browser. The most commonly used procedures run in linear time with little overhead, so this is practical on modest hardware even for large diagrams. Proofs can be stored on the remote server for later reference, or downloaded for storage locally. Permanent hyperlinks to formalized proofs can be generated and embedded as links in research papers, allowing readers instant access to the formalization without the usual barriers-to-use of downloading, installing and maintaining an executable. The tool launched in December 2015, and has been well-received by the community, with 4126 sessions across 884 unique users in the first 2 months since deployment[1].

In Section 2, we give a brief overview of the mathematical foundations of Globular, namely higher-dimensional category theory and rewriting. In Section 3 we exhibit all of the core functionality of the tool via a simple example. In Section 4 we discuss the implementation, including the architecture and relevant data structures and procedures for rewriting. We conclude by surveying a variety of interesting proofs in Globular by the authors and others, with direct links for viewing online.

2 Mathematical foundations

Higher category theory is the study of *n-categories*. As well as objects and morphisms familiar from traditional category theory, which are we call 0-cells and 1-cells, an n-category also has morphisms between morphisms (2-cells), morphisms between those (3-cells), and so on, up to level n. An n-category has a n distinct composition operations, which allow cells to be combined to produce new cells.

A convenient notation for working with n-categories is the *graphical calculus*, in which a k-cell is represented as an $(n-k)$-dimensional geometrical structure[2]. Composition then corresponds to 'gluing' of these structures along the different axes of n-dimensional space. For example, in a 3-category, we represent 3-cells as points, 2-cells as lines, 1-cells as regions, and 0-cells as 'volumes'. Given 3-cells α and β, we could form the following composite 3-cells by composing along three different axes:



(1)

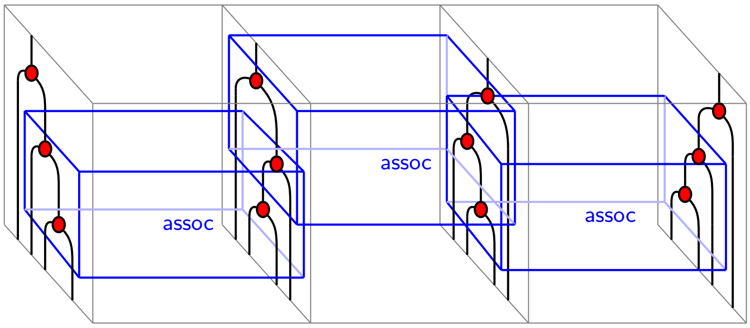
In this way we can draw diagrams to represent arbitrary composites, in principle in any dimension (although for $n > 3$, visualizing the resulting geometrical structure of course becomes nontrivial).

[1] Usage statistics from Google Web Analytics.

[2] This is rigorously developed only for $n \leq 3$ [2, 6].

We take a *rewriting* perspective on higher category theory. Suppose a $(k+1)$-cell X has source and target k-cells α and α' respectively. Then we interpret X as a way to rewrite α into α'. Since composition in higher category theory is local, this also works for composite cells: for example, we can apply X to any of the composites in (1) to obtain a new composite with α replaced by α'.

The attractive feature of this perspective is that there is no fundamental difference between the notions of *composition* and *proof.* A proof that some diagram D of k-cells can rewritten into some other diagram D' amounts to building a composite $(k+1)$-cell with source D and target D', using just the 'axiom' cells of a given theory. For instance, if we have a 3-cell called 'assoc' which captures an associativity rule of 2-cells, we can prove a theorem about associativity as a composition of 3-cells:

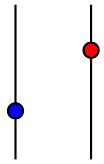


That is, we can *define* a composite $(k+1)$-cell as a rewrite sequence on composite k-cells. This gives a recursive definition of composition, which terminates with a family of 'basic' rewrite operations, which the user must specify. This is the essence of Globular's approach to higher category theory.

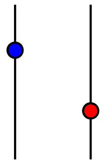
In higher category theory, we have some freedom to decide what it means for two things to be 'the same'. At one extreme are 'fully weak' n-categories, where all of the axioms governing the composition of cells (such as associativity and unit axioms) hold only up to higher-dimensional cells. For example, for 1-cells f, g, h, rather than requiring associativity: $f \circ (g \circ h) = (f \circ g) \circ h$, we merely assert the existence of a (weakly) invertible family of 'associator' 2-cells $(f \circ g) \circ h \to f \circ (g \circ h)$. These in turn must satisfy various coherence properties, which we again interpret only up to higher-dimensional cells (which *themselves* must satisfy coherence properties, and so on). While these structures arise naturally in many contexts, the amount of bureaucracy that arises from this structure makes it hard to work with weak n-categories as purely syntactic objects.

At the other extreme are the *strict n-categories* which require all the axioms involving composition of cells to hold as on-the-nose equalities. These are quite easy to define [11], and admit an evident notion of finite presentation, called a *polygraph* or *computad*, and have a reasonably well-behaved *higher-dimensional rewrite theory* [4]. However, for $n > 2$, it is *not* the case that every weak n-category is equivalent to a strict one. To see where this richness of weak categories comes from, we consider the interchange law, which in a 2-category acts as follows as a rewrite on composite 2-cells:

$$(\boldsymbol{f} \circ_1 1_B) \circ_2 (1_{A'} \circ_1 \boldsymbol{g}) \quad :=$$

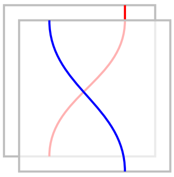


$$\xrightarrow{I}$$

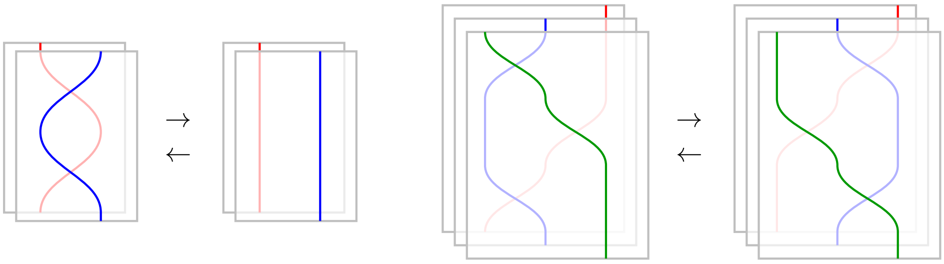


$$=: \quad (1_A \circ_1 \boldsymbol{g}) \circ_2 (\boldsymbol{f} \circ_1 1_{B'})$$

When we stop at two dimensions, there is no problem treating this ‘node-sliding’ rule simply as an equation between diagrams. But seen as a 3-cell in a 3-category, the source and target of I become the bottom and top slice of a 3D picture, the nodes become wires, and the ‘sliding’ becomes a braiding:



By the invertibility and naturalness properties, these braidings then behave exactly how you would expect genuine topological braids to behave. For instance, the following higher rewrites exist:

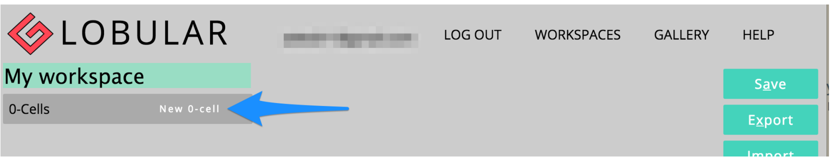


In general, overcrossings and undercrossings are distinct, so it is possible for wires to become tangled. Requiring interchangers to be identities, as in the theory of strict 3-categories, trivializes this part of the theory, and means that it is no longer fully general, in the precise sense that not every 3-category is equivalent to a strict 3-category.

It follows that the strict n-categorical setting in which the polygraph community work is not sufficiently general to reason about arbitrary n-categories. The solution is to work instead with *semistrict* n-categories, which allows a small amount of weak structure, sufficient to ensure that every weak n-category is equivalent to semistrict n-category. For $n = 3$, *Gray categories* have this property; they are defined as 3-categories in which all weak structure is the identity, *except* for interchangers[3]. The version of Globular which is operational at the time of writing implements the axioms of a Gray category.

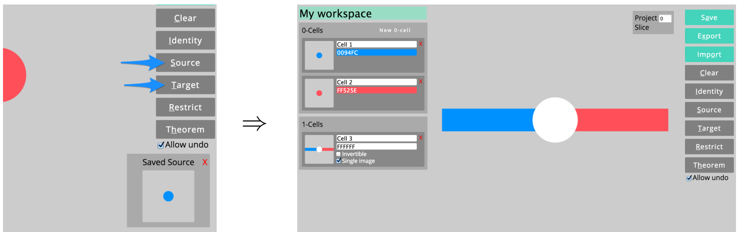
3 Using Globular

Constructing a theory and proving theorems in Globular is an inductive process, whereby lower-dimensional objects are used to construct higher-dimensional objects. This is done by building up a *signature*, i.e. a collection of generators, in parallel with increasingly higher-dimensional diagrams. From an empty signature, the only thing to do is add new 0-cells:



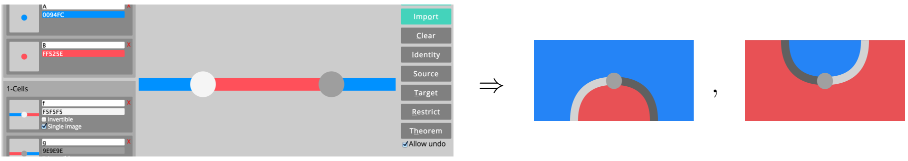
[3] A definition of semistrict n-category for $n > 3$ has not yet been generally accepted.

Once we have some 0-cells, these can be made the sources and targets of new 1-cells:



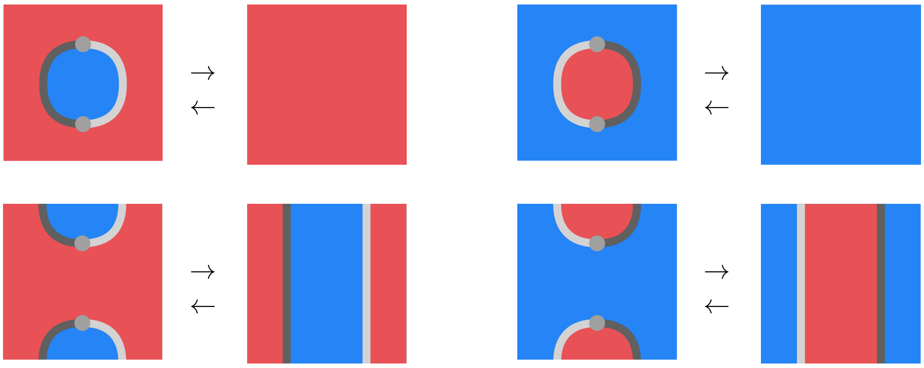
Cell 1
0094FC
Cell 2
FF525E
Cell 3
FFFFFF
□ Invertible
☑ Single image
Project 0

At this point things start to get interesting, since 1-cells can be *attached* to each other to form non-trivial diagrams. These diagrams can then form the sources and targets of new 2-cells:



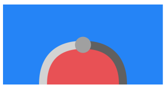
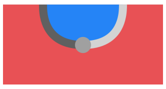
A
0094FC
B
FF525E
f
F5F5F5
□ Invertible
☑ Single image
g
9E9E9E

In turn, these 2-cells can be composed to form larger diagrams, which and form the sources and targets of new 3-cells. We can either interpret these new 3-cells as new generators, or as *equations* between 2d diagrams. For example, we can make our 'cap' and 'cup' 2-cells invertible by adding the following 3-cells to our theory:



These invertible 'cup' and 'cap' 2-cells yield a familiar categorical structure.

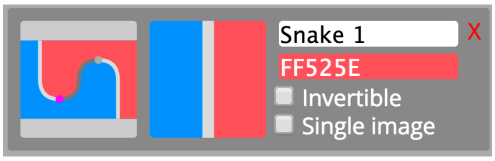
▶ **Definition 1.** In a 2-category, an *equivalence* is a pair of objects A and B, a pair of 1-cells $A \xrightarrow{F} B$ and $B \xrightarrow{G} A$ and invertible 2-cells $F \circ G \xrightarrow{\alpha} \mathrm{id}_A$ and $\mathrm{id}_A \xrightarrow{\beta} G \circ F$, denoted as follows:

$\alpha \equiv$  $\beta \equiv$ 

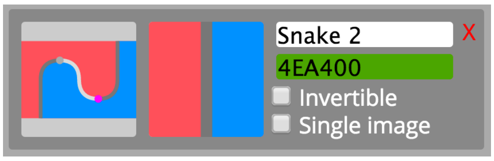
A special case is where the 2-category is **Cat**, in which case this yields the usual notion of equivalence of categories. Then the following is a well-known fact about equivalences in a 2-category:

▶ **Theorem 2.** *In a 2-category, every equivalence gives rise to a dual equivalence.*

An equivalence is called a *dual equivalence* if it additionally satisfies the *snake equations*, shown here as theorems in Globular:

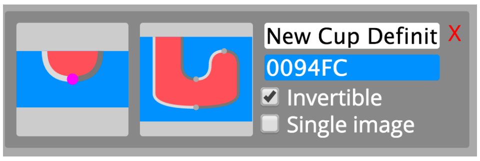


Snake 1 X
FF525E
□ Invertible
□ Single image



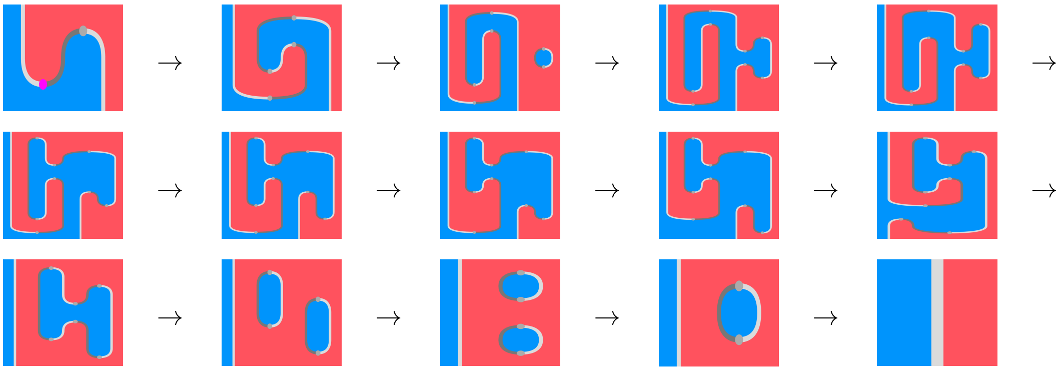
Snake 2 X
4EA400
□ Invertible
□ Single image

We can prove these theorems by replacing the ‘cup’ with a ‘sock’, defined in terms of the old cup and cap:

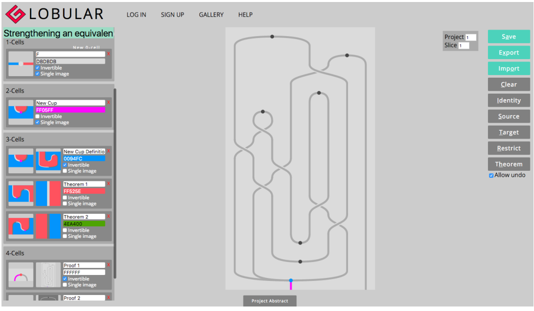


New Cup Definit X
0094FC
☑ Invertible
□ Single image

We can show that our new ‘cup’ satisfies the snake equation, with the original ‘cap’. To prove the first snake equation, we perform the following (non-trivial!) sequence of rewrites in Globular:



This proof is itself a 3-cell. In Globular, we can either browse through it slice-by-slice, or we can see the overall structure of the proof as a single diagram, by choosing ‘Project=1’ in the interface:



Strengthening an equivalen

1-Cells

New 0-cell

F X
DBDBDB
☑ Invertible
☑ Single image

2-Cells

New Cup X
FF05FF
□ Invertible
☑ Single image

3-Cells

New Cup Definitio X
0094FC
☑ Invertible
□ Single image

Theorem 1 X
FF525E
□ Invertible
□ Single image

Theorem 2 X
4EA400
□ Invertible
□ Single image

4-Cells

Proof 1 X
FFFFFF
☑ Invertible
□ Single image

Proof 2 X

Project 1
Slice 1

☑ Allow undo

This projects out one dimension so we call look at this entire 3-cell ‘side-on’. The nodes represent applications of rewrite rules, and the wires represent 2-cells. From this view, we can refactor the proof by eliminating redundant steps (e.g. a rewrite immediately followed by its inverse) or by re-ordering rewrites that are applied to independent parts of the diagram.

Once a proof has been constructed, it can be saved privately to the server, or made public by *publishing* it. This assigns the workspace a permanent unique link, which can be shared with others or linked from a research paper. For example, the proof in this section is based on the formalization available here: globular.science/1512.007.

4 Implementation

In this section we give an overview of the implementation of Globular, and the basic structures and algorithms that underlie its operation. The tool itself can be used simply by visiting:

`http://globular.science`

The proof assistant itself runs client-side in the user's web browser, and is written in Javascript. A Node.js back-end serves the client pages, and administers a system of user accounts allowing users to register, store private proofs that are under construction, and (irrevocably) publish proofs that they would like to share publicly. The project is open-source, and the code is available at globular.science/source. Graphics are implemented in SVG.

4.1 Data types

The fundamental structures that Globular makes use of are *signatures*, which are a lists of basic generating cells that the user has specified as to define a theory, and *diagrams*, which are particular composites of generators from a given signature. These two types can be defined very compactly in a mutually-recursive fashion. For clarity, we write these both as type families in dependent type-style notion. Let 'g : Set' declare a finite set g (which we then treat as a type), let $\mathrm{List}(\alpha)$ be the type of lists, and $\langle \alpha_1, \alpha_2, \ldots \rangle$ the type of tuples where types in α_j are allowed to depend on α_i for $i < j$. Let $\mathrm{Sig}(0)$ and $\mathrm{Diag}(0, *)$ both be the unit type $\{*\}$. Then, for $n > 0$:

$$\mathrm{Sig}(n:\mathbb{N}) := \left\langle \begin{array}{l} g : \mathrm{Set}, \\ \sigma : \mathrm{Sig}(n-1), \\ s, t : g \to \mathrm{Diag}(n-1, \sigma) \end{array} \right\rangle \qquad \mathrm{Diag}(n:\mathbb{N}, \sigma : \mathrm{Sig}(n)) := \left\langle \begin{array}{l} s : \mathrm{Diag}(n-1, \sigma), \\ \delta : \mathrm{List}(\ \langle a : \sigma.g,\ \ c : \mathrm{List}(\mathbb{N}) \rangle\ ) \end{array} \right\rangle$$

An n-signature $\Sigma : \mathrm{Sig}(n)$ therefore consists of an $(n-1)$-signature $\Sigma.\sigma$, and a set of generators $\Sigma.g$, such that each $x : \Sigma.g$ has a source and target $(n-1)$-diagrams $\Sigma.s(x)$ and $\Sigma.t(x)$ respectively, which each contain cells from the $(n-1)$-signature $\Sigma.\sigma$.

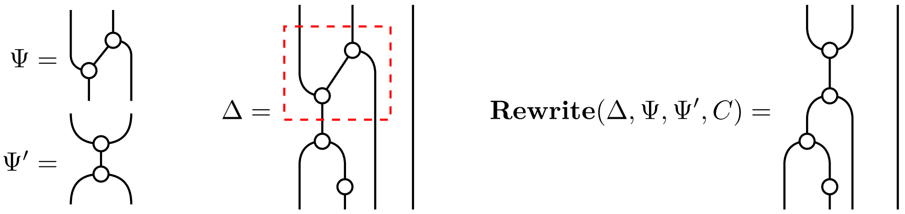
Given a signature $\sigma : \mathrm{Sig}(n)$, a diagram $\Delta : \mathrm{Diag}(n, \sigma)$ consists of a source $(n-1)$-diagram $\Delta.s$, and a list of n-cells that act sequentially on that source. The kth n-cell is given by a pair $\Delta.\delta[k]$, whose first element $\Delta.\delta[k].a$ is a generating cell drawn from the signature σ, and whose second element $\Delta.\delta[k].c$ is a list of numbers which specify the coordinates at which the chosen generating rewrite acts. For example, a 2-diagram consists of a list of 2-cells which are stacked vertically, and this coordinate consists of a single number giving the horizontal position of each 2-cell. We leave the target $(n-1)$-cell implicit, as it can be recovered from the other data (e.g. via the **Slice** procedure below).

4.2 Procedures

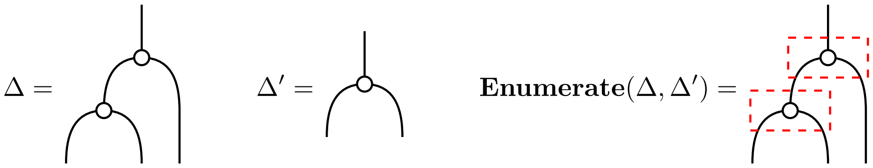
Here we give the type specifications of the basic procedures that manipulate our diagram structures, along with brief descriptions of their functionality.

- **Match**$\big(\Delta : \mathrm{Diag}(n, \sigma), \Delta' : \mathrm{Diag}(n, \sigma)\big) : \mathrm{Bool}$
  Determines whether two diagrams are equal. For Δ, Δ' we first recursively compare whether $\Delta.s$ and $\Delta'.s$ match. If not, return **false**. Otherwise, we compare corresponding elements of $\Delta.\delta$ and $\Delta'.\delta$, if there is a pair δ_k, δ'_k such that either types $\delta_k.a, \delta'_k.a$ or coordinates do not match, then return **false**, otherwise return **true**.

- **Identity**$\big(\Delta : \mathrm{Diag}(n,\sigma)\big) : \mathrm{Diag}(n+1,\sigma)$
  Given an n-diagram, this operation transforms it into an identity $(n+1)$-diagram $\Delta'(n+1,\sigma)$. The set of generators $\Delta'.\delta$ is empty, while both $\Delta'.s$ and $\Delta'.t$ are set to Δ.
- **Rewrite**$(\Delta : \mathrm{Diag}(n,\sigma), \Psi : \mathrm{Diag}(n,\sigma), \Psi' : \mathrm{Diag}(n,\sigma), C : \mathrm{List}(\mathbb{N})) : \mathrm{Diag}(n,\sigma)$
  Here Δ is the diagram that is being rewritten, Ψ is the source of the rewrite, Ψ' is the target of the rewrite, and C is the list of coordinates of where the rewrite is to be applied. $|\Psi.\delta|$ consecutive rewrites in $\Delta.\delta$ starting from position C_{n-1} are removed, with the rewrites in $\Psi'.\delta$ inserted, with their coordinates offset by C. We illustrate this with a simple example, where C is denoted by the dashed rectangle:

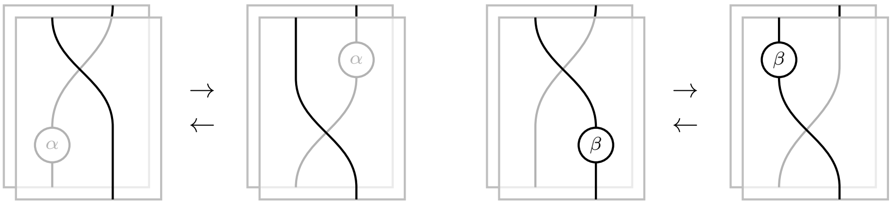


- **Attach**$(\Delta : \mathrm{Diag}(n,\sigma), \Delta' : \mathrm{Diag}(k,\sigma), P : \{s,t\}, C : \mathrm{List}(\mathbb{N})) : \mathrm{Diag}(n,\sigma)$
  Attaches the diagram Δ' to the diagram Δ, where P is a boolean distinguishing between attachment to the source or target boundary of Δ', and C are the coordinates within this boundary of where Δ' is to be attached.
- **Slice**$(\Delta : \mathrm{Diag}(n,\sigma), k : \mathbb{N}) : \mathrm{Diag}(n-1,\sigma.\sigma)$
  Rewrite the source boundary $\Delta.s$ using the leading k n-cells in the ordered set $\Delta.\delta$, to obtain an intermediate diagram in the rewrite sequence.
- **Enumerate**$\big(\Delta : \mathrm{Diag}(n,\sigma), \Delta' : \mathrm{Diag}(n,\sigma)\big) : \mathrm{List}(\mathrm{List}(\mathbb{N}))$
  Enumerates the locations at which Δ' occurs as a subdiagram of Δ. For example, for Δ and Δ' 2-dimensional diagrams as given, the procedure returns a list of length 2:



The implementation is as follows. First we loop through the elements of the rewrite list $\Delta.\delta$. For the rewrite at depth k, we recursively call **Enumerate**(**Slice**$(\Delta, k-1), \Delta'.s)$. If the result is the empty list, we increment k and retry. If the result is nonempty, at most 1 can be consistent with the structure of Δ, and we then compare types and coordinates of the corresponding generators in $\Delta.\delta$ and $\Delta'.\delta$. If they match, we append k to the coordinate list returned by the recursive call, and we add the coordinate list as a witness to the instance of Δ' being a sub-diagram of Δ.

Globular also has procedures which generate cell coming from the semistrict n-category structure on demand. These consist of *interchangers*, which we've already seen, and *pull-throughs*, which capture the naturality of interchangers:



These cells are generated as they are required because they in fact form an infinite family of cells. This is because f, g and the wires depicted above might be basic generators, but could also be diagrams of generators.

5 Examples

Here we give examples of formalized proofs from algebra and topology. In each case we briefly describe the mathematical context of the proof, and give some details of its formalization. Direct hyperlinks are provided to the formalized proofs on the Globular website; to navigate these proofs, use the *Project* and *Slice* controls at the top-right, and move your mouse cursor over the different parts of the main diagram to understand its components. Documentation on how to use Globular is available [20]. To our knowledge, none of these results have previously been formalized by any existing tool.

▶ **Example 3** (Frobenius implies associative, globular.science/1512.004, length 12)**.** *In a monoidal category, if multiplication and comultiplication morphisms are unital, counital and Frobenius, then they are associative and coassociative.* We formalize this in Globular using a 2-category with a single 0-cell, since this is algebraically equivalent to a monoidal category. Such a proof would be traditionally written out as a series of pictures; for example, see the textbook [8]. Globular produces these pictures automatically.

▶ **Example 4** (Strengthening an equivalence, globular.science/1512.007, length 14)**.** *In a 2-category, an equivalence gives rise to an adjoint equivalence.* This is a classic result from the category theory community [1, 18]; it can be considered one of the first nontrivial theorems of 2-category theory. We investigate it in further detail in Section 3.

▶ **Example 5** (Swallowtail comes for free, globular.science/1512.006, length 12)**.** *In a monoidal 2-category, a weakly-dual pair of objects gives rise to a strongly-dual pair, satisfying the swallowtail equations.* This theorem plays an important role in the singularity theory of 3-manifolds [17]. For the formalization, we model a monoidal 2-category as a 3-category with one 0-cell.

▶ **Example 6** (Pentagon and triangle implies $\rho_I = \lambda_I$, globular.science/1512.002, length 62)**.** *In a monoidal 2-category, a pseudomonoid object satisfies $\rho_I = \lambda_I$.* A *pseudomonoid* is a higher algebraic structure categorifying the concept of monoid; it has the property that a pseudomonoid in **Cat** is the same as a monoidal category. Such a structure is known to be coherent [9], in the sense that all equations commute, and here we give an explicit proof of the equation $\rho_I = \lambda_I$, which played an important role in the early study of coherence for monoidal categories.

▶ **Example 7** (The antipode is an algebra homomorphism, globular.science/1512.011, length 68)**.** *For a Hopf algebra structure in a braided monoidal category, the antipode is an algebra homomorphism.* Hopf algebras are algebraic structures which play an important role in representation theory and physics [12, 19]. Proofs involving these structures are usually presented in *Sweedler notation*, a linear syntax which represents coalgebraic structures using strings of formal variables with subscripts; we do not know of any existing approaches to formal verification for Sweedler proofs. This formalization in Globular is translated from a Sweedler proof given in [15]. For the formalization, we model a braided monoidal category as a 3-category with one 0-cell and one 1-cell.

▶ **Example 8** (The Perko knots are isotopic, globular.science/1512.012, length 251)**.** *The Perko knots are isotopic.* The Perko knots are a pair of 10-crossing knots stated by Little in 1899 to be distinct, but proven by Perko in 1974 to be isotopic [16]. Here we give the isotopy proof, adapted from [13]. A nice feature is that the second and third Reidemeister moves do not have to be entered, since they are already implied by the 3-category axioms. The proof consists of a series of 251 atomic deformations, which rewrite the first Perko knot into the second. By stepping through the proof one rewrite at a time, the isotopy itself can be visualized as a movie.

Acknowledgements. We would like to thank John Baez, Manuel Bärenz, Bruce Bartlett, Eugenia Cheng, Chris Douglas, Eric Finster, Nick Gurski, André Henriques, Samuel Mimram and Dominic Verdon for useful discussions.

References

1 John C. Baez and Aaron D. Lauda. Higher-dimensional algebra V: 2-groups. *Theory and Applications of Categories*, 12:423–491, 2004. URL: `http://www.tac.mta.ca/tac/volumes/12/14/12-14abs.html`.

2 John W. Barrett, Catherine Meusburger, and Gregor Schaumann. Gray categories with duals and their diagrams. *J. Diff. Geom., to appear.* URL: `http://arxiv.org/abs/1211.0529`.

3 Eric Finster. The *Orchard* proof assistant. URL: `https://github.com/ericfinster/orchard`.

4 Yves Guiraud. Polygraphs for termination of left-linear term rewriting systems. 2007. URL: `http://arxiv.org/abs/cs/0702040`.

5 HoTT Formalisations in Coq and Agda. URL: `http://homotopytypetheory.org/coq`.

6 André Joyal and Ross Street. The geometry of tensor calculus, I. *Adv. Math.*, 88(1):55–112, 1991. `doi:10.1016/0001-8708(91)90003-p`.

7 Aleks Kissinger and Vladimir Zamdzhiev. Quantomatic: A proof assistant for diagrammatic reasoning. In *CADE-25 – 25th International Conference on Automated Deduction*, volume 9195 of *LNCS*. Springer, 2015. `doi:10.1007/978-3-319-21401-6\_22`.

8 Joachim Kock. *Frobenius Algebras and 2D Topological Quantum Field Theories.* Cambridge University Press (CUP), 2003. `doi:10.1017/cbo9780511615443`.

9 Stephen Lack. A coherent approach to pseudomonads. *Adv. Math.*, 152(2):179–202, 2000. `doi:10.1006/aima.1999.1881`.

10 Yves Lafont. Algebra and geometry of rewriting. *Applied Categorical Structures*, 15(4):415–437, 2007. `doi:10.1007/s10485-007-9083-6`.

11 Tom Leinster. A survey of definitions of n-category. *Theory and Applications of Categories*, 10(1):1–70, 2002. `http://arxiv.org/abs/math/0107188`. URL: `http://tac.mta.ca/tac/volumes/10/1/10-01abs.html`.

12 Shahn Majid. *A Quantum Groups Primer.* Cambridge University Press (CUP), 2002. `doi:10.1017/cbo9780511549892`.

13 MathForum. Perko pair knots. URL: `http://mathforum.org/mathimages/index.php/Perko_pair_knots`.

14 Samuel Mimram. Towards 3-dimensional rewriting theory. *Logical Methods in Computer Science*, 10(2), 2014. `doi:10.2168/LMCS-10(2:1)2014`.

15 Bodo Pareigis. In Stefaan Caenepeel and Fred Van Oystaeyen, editors, *Hopf Algebras in Noncommutative Geometry and Physics*, chapter On Symbolic Computations in Braided Monoidal Categories, pages 269–280. CRC Press, 2004.

16 Kenneth A. Perko. On the classification of knots. *Proceedings of the AMS*, 45(2):262–262, 1974. `doi:10.1090/s0002-9939-1974-0353294-x`.

17 Piotr Pstragowski. On dualizable objects in monoidal bicategories. Master's thesis, Bonn University, 2014. URL: http://arxiv.org/abs/1411.6691.

18 Saavedra Rivano. *Catégories Tannakiennes*, volume 265 of *Lecture Notes in Mathematics*. Springer Berlin Heidelberg, 1972. doi:10.1007/bfb0059108.

19 Ross Street. *Quantum Groups*. Cambridge University Press (CUP), 2007. doi:10.1017/cbo9780511618505.

20 The Globular Team. Globular documentation. URL: https://ncatlab.org/nlab/show/Globular.

Interaction Automata and the ia2d Interpreter*

Stéphane Gimenez[1] and David Obwaller[2]

1 **Institute of Computer Science, University of Innsbruck, Innsbruck, Austria**
`stephane.gimenez@uibk.ac.at`

2 **Institute of Computer Science, University of Innsbruck, Innsbruck, Austria**
`david.obwaller@student.uibk.ac.at`

Abstract

We introduce interaction automata as a topological model of computation and present the conceptual plane interpreter `ia2d`. Interaction automata form a refinement of both interaction nets and cellular automata models that combine data deployment, memory management and structured computation mechanisms. Their local structure is inspired from pointer machines and allows an asynchronous spatial distribution of the computation. Our tool can be considered as a proof-of-concept piece of abstract hardware on which functional programs can be run in parallel.

1998 ACM Subject Classification F.1.2 Modes of computation

Keywords and phrases Interaction nets, computation models, parallel computation, functional programming

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.35

1 Introduction

We present a candidate computation model with a quite natural, although non-standard memory model reminiscent of cellular automata topologies which takes spatial distances and related data migration costs into consideration. Our associated tool interpreter `ia2d` is written in Haskell and is available online[1]. This interpreter takes as input a program written as an *interaction-net system*, a generic, topology-independent language. This program is then executed in parallel on a planar instance of the presented computation model, and the result of the computation is returned in the same interaction-net language, along with some detailed resource usage statistics. Operations on binary-encoded natural numbers, implementations of *merge sort* and of the *bitonic sorter*, which has a theoretical parallel time complexity of $O(log^2(n))$, are provided as examples.

Our work is aimed at filling a gap that exists between the standard interaction-net model and real distributed memory schemes. These schemes depart from the traditional random access memory schemes in that they do not allow a uniform constant-time access to data stored remotely. Our long term objective is an abstract execution model for asynchronous computation adapted to both hardware components and network computing that automatically distributes the computation over the available computation units and incorporate latency and bandwidth management. We present here a simple and somewhat naive approach which is nevertheless partially successful.

We know that interaction nets are a target of choice to run functional programs in parallel [16, 15, 4, 2, 3], or to express concurrency such as in process algebras [18, 19]. Various

* This work was partially supported by FWF (Austrian Science Fund) project number P 25781-N18.

[1] The ia2d interpreter: `http://bitbucket.org/inarch/ia2d`

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 35; pp. 35:1–35:11
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

implementations of interaction nets exist [20, 21, 1], some of which use a particular technique called geometry of interaction [22]. But all consider traditional computer architectures as a base model, with random access memory schemes. Our goal is to use micro and macro parallelism extensively – beyond the standard multi-threading "compromise" – to run generic programs which have not been instrumented with hardware, network, or architecture-specific parallel constructions.

Towards this same goal, an experimental implementation of interaction nets on parallel hardware such as GPUs has been developed [10]. A variant of interaction nets called *hard interaction nets* has also been introduced to model asynchronous hardware components [14], but we do not know of any practical way to use this model to run standard programs. A compiler to hard interaction nets still has to be developed.

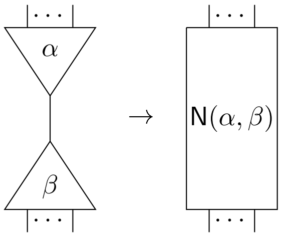
The model which we introduce is meant to incorporate spatial considerations in the most generic way. Notably, when its topological constraints are lifted, we show that it corresponds closely to the standard interaction-net model.

2 Interaction Nets

We present in this section *interaction nets* and their reduction which is based on graph-rewriting techniques. This overview should be sufficient in order to understand the programs provided as examples along with our tool. For a more detailed introduction, the reader is referred to the seminal paper by Yves Lafont [11].

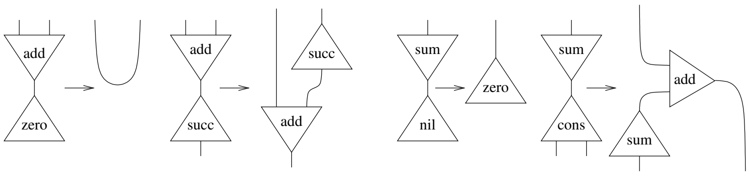
An *interaction net* is a graph, with vertices called *nodes* that are labeled with *symbols*, and edges called *wires*. Each node has a *principal port* as well as a certain number of *auxiliary* ports that is fixed for a given symbol. This number of auxiliary ports is called the *arity* of the symbol. A node is typically drawn as a triangle with the principal port at the tip and the auxiliary ports on the opposite side. Node ports can be connected together with a wire. When two nodes are connected on their principal ports, we call the pair an *active pair* or a *redex*. One or both ends of a wire may also be connected to *free* ports, which do not belong to any cell. The set of free ports of a net is called its *interface*.

An *interaction net system* is a pair (S, R) of *symbols* and *reduction rules*. Given an interaction net built using nodes labeled with symbols S, we can rewrite the net according to the reduction rules R. A reduction rule describes how an active pair of nodes can be rewritten by removing the active pair from the graph and substituting it by a net with the same interface. Such a rule is generally represented as follows:

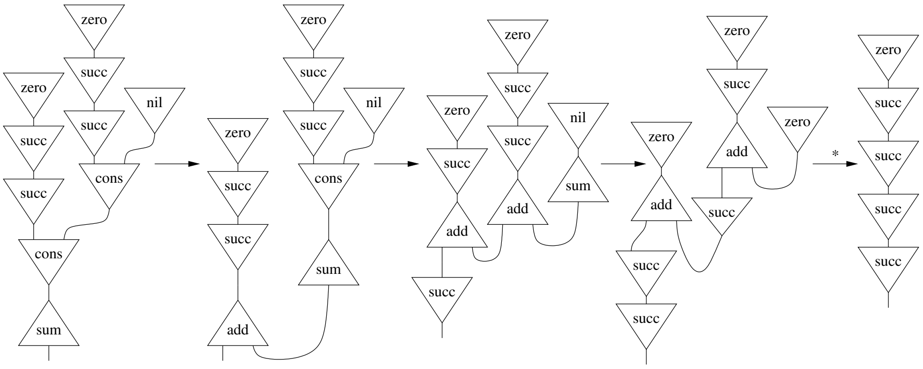


To ensure determinism only one such rule is allowed for a given pair of symbols. If $\alpha = \beta$ the right-hand side of the rule is required to be top-down symmetric. A strong confluence property, namely the diamond property, holds for the reduction of interaction nets. The order in which we choose to reduce the active pairs of the net is not important as it does not change the outcome of the reduction. Furthermore, because the reduction steps are performed locally and cannot overlap, they can be performed in parallel.

When programming with interaction nets we use labeled nodes to represent both constructors and operations. In the following example, we use the symbols zero (nullary) and succ (unary) to represent the constructors for natural numbers, cons (binary) and nil (nullary) to represent lists, and the symbols add (binary) and sum (unary) for the addition operations on natural numbers and the sum operation on lists of naturals respectively. We can define each operation using a pair of rules as follows:



The following example computation illustrates how a given interaction net can be rewritten using the rules given above. We start with a simple net that represents a list of two natural numbers connected to a sum operation, which will compute the sum of the two numbers.



The net on the left shows the list of natural numbers cons$(2,$ cons$(2,$ nil$()))$ connected to the sum operator. After two parallel reduction steps, denoted as $\rightarrow$, we obtain three redexes, namely two instances of add $\bowtie$ succ and sum $\bowtie$ nil. After exhaustively applying reduction rules until no redexes are left, as denoted by $\xrightarrow{*}$, we obtain the final net, the natural number 4 represented as net.

3 Interaction Automata

3.1 Definition of the Computation Model

We start by introducing a notion of web, which defines the topology on which interaction automata will be built.

▶ **Definition 1.** A *web* is defined as a pair $W = (L, \nu)$, where L is a set of locations, called *support*, and $\nu : L \to \mathcal{P}(L)$ a map that associates a set of locations, called *neighborhood*, to every location.

Interaction agents in our model will be nodes. Their purpose is indicated by a particular symbol and they store a certain number of pointers to other locations. This number has to match with a particular *arity* that was attributed to the symbol. Locations represent positions at which two nodes are expected to interact.

▶ **Definition 2.** For a given arity-attributed set of symbols S and a set of locations L, we define *nodes* according to the syntax $n ::= \omega(l_0) \mid s(l_1, \ldots, l_k)$ where $s \in S$, k is the arity of s, and the $l_i \in L$ form a list of locations called pointers. The set of all possible nodes is written $N(S, L)$.

The ω notation can be considered as a particular "pointer-target" symbol, which contains a reference that links back to the origin of the pointer.

▶ **Definition 3.** We define the forward-reference multiset of a node as $\mathsf{f}(\omega(l_0)) = \varnothing$ and $\mathsf{f}(s(l_1, \ldots, l_k)) = \{l_1, \ldots, l_k\}$ and its backward-reference multiset as $\mathsf{b}(\omega(l_0)) = \{l_0\}$ and $\mathsf{b}(s(l_1, \ldots, l_k)) = \varnothing$. These notations are straightforwardly extended to multisets of nodes through multiset union.

▶ **Definition 4.** Given a set of symbols S, a *domain* $D \subseteq L$ and a disjoint *interface* $I \subseteq L$, we define $\Gamma(S, D, I)$ as the set of maps from locations to multisets of nodes $\mu : L \to \mathcal{M}(N(S, L))$, called *configurations*, such that the cardinal of $\mu(l)$ is respectively 2 if $l \in D$, or 1 if $l \in I$, or 0 otherwise. Additionally we require that for all $l_s, l_t \in L$ the multiplicity of l_t in $\mathsf{f}(\mu(l_s))$ matches the one of l_s in $\mathsf{b}(\mu(l_t))$.

The latter constraint ensures that every forward-reference attached to a symbol node matches with a backward-reference attached to an ω-node.

▶ **Definition 5.** A *topological configuration* of a web $W = (L, \nu)$ with symbols in S is the combination of a domain D, an interface I and a configuration $\gamma \in \Gamma(S, D, I)$ that satisfies neighborhood compatibility, i.e., $l_t \in \mathsf{f}(\gamma(l_s)) \Rightarrow l_t \in \nu(l_s)$, for all $l_s, l_t \in L$.

We will rely on an abstract reduction scheme to automatically endow any given web with a transfer function from configurations to configurations.

▶ **Definition 6.** An *abstract transition scheme* over a set of symbols S is a binary relation $\mu_l \to \mu_r$ over configurations $\mu_l \in \Gamma(S, D_l, I)$ and $\mu_r \in \Gamma(S, D_r, I)$ which is invariant upon permutations of locations in L. The sets D_l, D_r and I are arbitrary disjoint subsets of L.

D_l and D_r represent respectively the sets of freed and allocated locations during a reduction step that is defined by the abstract transition scheme. If we additionally require that D_l has cardinal 1 and that the singleton $\mu(l)$ stored at any $l \in I$ is an ω-node, the scheme is called *atomic*.

▶ **Definition 7.** An *interaction automaton* is defined as a quadruple $A = (L, \nu, S, \to)$ where $W = (L, \nu)$ is a chosen web, S is a set of symbols, and $\to$ is an abstract transition scheme over S.

▶ **Definition 8.** Given an interaction automaton A, a *parallel transition* $\gamma_l \overset{A}{\twoheadrightarrow} \gamma_r$ occurs between two topological configurations of its web, $\gamma_l \in \Gamma(S, D_l, I)$ and $\gamma_r \in \Gamma(S, D_r, I)$, if both configurations are pointwise multiset unions of configurations $\gamma_l = \bigoplus_i \mu_i^l \oplus \mu$ and $\gamma_r = \bigoplus_i \mu_i^r \oplus \mu$, such that each pair of sub-configurations satisfy the abstract transition scheme relation $\mu_i^l \to \mu_i^r$.

If the chosen abstract transition scheme is atomic, in the absence of topological constraints (unrestricted neighborhoods and an infinite number of locations), we can guarantee that the parallel transition relation, despite being non-deterministic, satisfies the diamond property up to a relocation of cells.

Bounded Interaction Automata and Interaction Grids. *Bounded interaction automata* are interaction automata for which the sizes of neighborhoods are globally bounded by a constant.

Example: *Interaction grids* of dimension d and neighborhood radius m are denoted by $\mathbb{G}^d_m$. They are particular interaction automata whose sets of locations are defined as $L = \mathbb{Z}^d$ and neighborhood maps as $\nu(\vec{x}) = \{\vec{x} + \vec{u}, |\vec{u}| \leq m\}$. We chose to work with the Manhattan distance in our implementation.

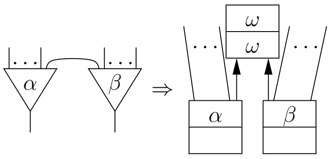
3.2 Implementation of Interaction Nets on Interaction Automata

We now show that interaction-net computation can be performed within the abstract interaction automata computation model without topological constraints.

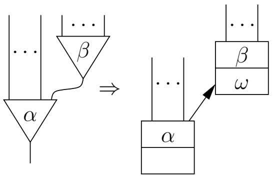
With topological constraints, nothing guarantees that allocations are always possible within the appropriate neighborhoods nor specify how the allocations should be done. They can be performed randomly, but the reduction is of course likely to block due to a local lack of space on webs with small connectivity. The allocations could also be done strategically if we rely on topological knowledge or external ways to gauge the occupancy of the web and its capacity. Our experiments however tend to show that a simple allocation strategy is enough in order to run interaction-net programs of a particular complexity class on webs with a matching connectivity.

The implementation of a given interaction-net system is quite straightforward. A glimpse at the output of our tool should be sufficient to illustrate the general principles which we sketch here.

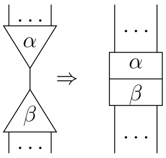
- Cells containing two ω-nodes are used to translate wires between two auxiliary ports.



- Mixed cells containing an ω-node and a symbol node are used to translate wires between an auxiliary port and a principal port.



- Cells containing two symbol nodes are used to translate wires from a principal port to another principal port.



- An interface cell containing a single ω-node is used to encode a wire that links one auxiliary port of an interaction-net node to one free port.
- An interface cell containing a single symbol node is used to encode a wire that links one principal port of an interaction-net node to one free port.

- The particular case of a wire that links two free ports is encoded by mapping two corresponding interface locations to nodes annotated with a particular *forwarder* symbol. Both nodes forward to the same additional location designated for this wire that contains two target ω-nodes.

Interaction-net node data is stored in the cell dedicated to the wiring of its principal port. Pointers are then set up such that related ports and wires reference each other.

Among the set of symbols S, additionally to the symbols of the chosen interaction-net system, as mentioned, a particular forwarder symbol (unary), which we denote by & in our tool, can be used to lengthen the wires if necessary. Each occurrence contains a pointer to the next hop on the path to the destination cell. In particular it is used in the transition schemes associated to right-hand sides that connect two interface ports with a wire directly.

Given the above encoding for nets, reductions rules are turned to abstract transition schemes $\mu_l \to \mu_r$ easily. The left-hand side of an interaction-net rule is encoded as a configuration μ_l with any single-location domain that stores the two interaction-net nodes that are part of the redex, and any interface that contain the required number of auxiliary ports present in the redex. The right-hand side is encoded as a configuration μ_r with a domain whose cardinal corresponds to the number of wires, except for those which connect a free port to a port of an interaction-net node. The usual constraints on interaction-net rules, including the symmetry of the rules that define interaction between two identical symbols, ensure that the defined abstract transition scheme is invariant upon permutations of the locations which were arbitrarily chosen as port identifiers.

Assuming neighborhoods are "sufficiently large to always contain free cells", the interaction-net reduction can be simulated on an interaction automaton. Different topologies are expected to lead to different performances. We implemented an interpreter for the planar topology $\mathbb{G}^2_m$.

3.3 Limits

For grids $\mathbb{G}^d_m$ (and similar topologies for which the size of iterated n-neighborhoods is polynomially bounded in n), due to data propagation constraints, the number of spawned redexes after a parallel running time t is necessarily bounded by a polynomial of degree d, the dimension of the grid. We cannot hope that the asymptotic speed-up offered by the parallelization will exceed this limit.

We have not yet investigated other topologies. Some multiscale webs including limited-bandwidth but long-distance links seem sufficiently realistic and could provide exponential speed-ups in many cases.

4 Implementation

The source code of `ia2d` is available at `https://bitbucket.org/inarch/ia2d`. For installation and usage instructions please consult the `README.md` file included in the source repository.

4.1 General Design Principles

Our automaton implementation works on a 2-dimensional grid of cells, each of which can hold up to two nodes. Parallel transitions of the automaton are performed in a loop. Each of these parallel transitions is preceded by a migration pass and an input/output pass as described hereafter.

Migration. The migration pass rearranges the cells in a way that avoids too high densities of cells, while at the same time keeping connected nodes within range of each other. As currently implemented the strategy favors migrations to locations whose neighborhoods contain the most empty cells. The final selection of a destination location among identically weighted candidates relies on randomization to avoid directional bias. Like the rest of the computation, the implemented migration strategy is local. Decisions are made by looking only at cells within a fixed radius.

Input and Output. For input and output the automaton uses special nodes in and out. During the input/output pass, the in nodes lazily insert nodes from the supplied program onto the grid when their interaction with other nodes is required. Conversely, the out nodes take nodes which are part of the normal form off the grid to produce a result. Currently, the collected result is printed at once on the terminal when the automaton stops, but input and output could also be streamed in real time.

Parallel Transitions. The main reduction pass rewrites cells that contain interaction-net redexes, as defined by the user, assuming enough free cells are locally available to store the right-hand sides of the reduction rules. The combination of a migration pass, an input/output pass and a parallel transition is repeated until the normal form of the input net has been entirely collected or an error such as memory exhaustion occurs.

Ressource Stress. The size of the grid has no effect if it is sufficiently large, but if the memory capacity is really tight the grid may become too densely populated, up to saturation. The firing of redexes that require allocations are in this case delayed until some space is made available locally by other reductions. Parallel execution which generally requires more space than sequential execution is therefore affected and gradually sequentializes to some extent. It may also occur that the memory capacity is simply too small for the computation, in which case the reduction will fail as it would on a normal computer.

Failures. Reduction can stop half-way if space is unavailable or cannot be freed locally. We developed this is software in order to understand when it happens in practice, how it can be avoided, and what theoretical results are necessary. For comparison, in traditional computation, the practical answer to a shortage of resources, (i.e., an insufficient amount of memory to run a certain algorithm) is simply to "fail" rather than to try to adapt to the situation. In the more tricky case of parallel computation, where a simple amount of memory is not the only parameter, we still have to decide whether it is worthwhile to adapt.

4.2 Input Language

As input language we use a variant of the Pin language [8]. The Pin language is a flat representation of graphs. We write the rule for an active pair between nodes α and β as $\alpha(x_1, \ldots, x_n) \bowtie \beta(y_1, \ldots, y_m) \Rightarrow N$ where N is a comma-separated list of *net components*. Net components are either:

- wires $x \sim y$,
- connections to nodes $x \sim \gamma(y_1, \ldots, y_k)$,
- or active pairs $\gamma_1(x_1, \ldots, x_n) \sim \gamma_2(y_1, \ldots, y_m)$.

Moreover, any variable used in a reduction rule should occur exactly twice in this rule. For example, one rule used to define binary addition is the following:

$$\mathsf{add}(y, r) \bowtie \mathsf{succ}(x) \Rightarrow r \sim \mathsf{succ}(t), \mathsf{add}(y, t) \sim x$$

In the actual input to the program we write $\bowtie$ as `><`, $\sim$ as `~`, $\Rightarrow$ as `=>` and we separate net components with commas and rules with semicolons. The following source code defines addition on natural numbers in unary encoding and the sum over a list of natural numbers:

```
add(y,r) >< zero() => r~y;
add(y,r) >< succ(x) => r~succ(t), add(y,t)~x;

sum(r) >< nil() => r~zero();
sum(r) >< cons(x, xs) => x~add(t, r), xs~sum(t);
```

A complete source file consists of an optional header of import statements and a list of rules and input nets. By storing the above library in a file called `nat_unary.inet`, we can write the computation of the sum of the natural numbers 4 and 2 as follows:

```
import "nat_unary.inet"
x ~ succ(succ(succ(succ(zero())))),
y ~ succ(succ(zero)),
cons(x, cons(y, nil())) ~ sum(r)
```

In this net, r is the only variable which occurs only once. It is the name associated to the result of this computation.

Along with the `ia2d` source code, a number of code examples are provided in the `examples` directory. The reader should refer to these files for more involved programming examples.

Synthesized Rules. Most interaction-net programs use δ nodes for duplicating and ϵ nodes for erasing parts of nets. The implementation of the automaton also makes use of forwarder nodes to connect two auxiliary ports or to extend the connections between distant nodes. The rules for δ, ϵ are synthesized and need not be provided by the programmer. For any user-defined symbol l of arity k the following rules are generated.

$$\begin{aligned}
\epsilon() \bowtie l(z_1, \dots, z_k) &\quad\Rightarrow\quad \epsilon() \sim z_1, \dots, \epsilon() \sim z_k \\
\delta(x, y) \bowtie l(z_1, \dots, z_k) &\quad\Rightarrow\quad x \sim l(x_1, \dots, x_k), y \sim l(y_1, \dots, y_k), \\
&\qquad\quad z_1 \sim \delta(x_1, y_1), \dots, z_k \sim \delta(x_k, y_k)
\end{aligned}$$

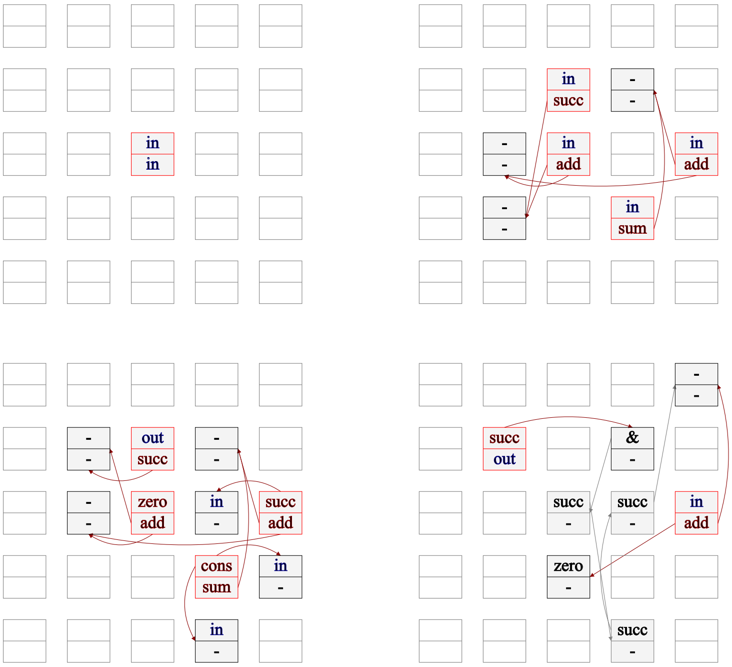
In order to duplicate cyclic data structures the rules $\epsilon() \bowtie \epsilon() \Rightarrow$ (with an empty list as right-hand side) and $\delta(x_1, y_1) \bowtie \delta(x_2, y_2) \Rightarrow x_1 \sim x_2, y_1 \sim y_2$ are provided as well.

4.3 Resource Usage Reports

When invoked on the command-line, besides the normal form of the provided net, `ia2d` outputs the number of individual transition steps and parallel reductions passes which were performed, along with more detailed reports for every reduction pass. (Please note that the output is truncated for brevity and might change in future versions.)

```
$ ia2d examples/bsort_example.inet --grid-size=16x16
SEQ STEPS: 664
PAR STEPS: 48
PASSES BREAKDOWN:
  ...
  1 x 27 fired
  1 x 29 fired
  1 x 29 fired and 1 delayed
  5 x 30 fired
  ...
DURATION:
  0.537617s
```

From the above output we can tell that ia2d had to perform 48 parallel reduction passes to reduce the input program. If we furthermore supply a parameter for the --svg flag, the intermediate states of the grid are made available as SVG files. As an illustration, we provide below four snapshots of the SVG output for the sum.inet program, which is available in the repository.



In the first picture, the grid is initially populated by just two in nodes which form the only redex of the input program. After three steps, as seen in the second picture, the grid contains more redexes and some input is still to be written onto the grid by the in nodes. The transition between the second and third picture is a migration pass, where existing in nodes are replaced with symbols from the input, new in nodes are added, and one different cell is migrated to a different position. The last picture shows the grid in a state where part of the output is already taken off the grid by an out node, while at the same time the final add node is still waiting for additional input data.

4.4 Results

The following table shows the number of parallel reduction passes needed to sort the leaves of a full binary tree with the *bitonic sorter* on the 2-dimensional grid model and compares it to the sequential number of steps that would be required in the standard interaction-net model.

number of leaves	sequential steps	parallel steps (average)	speed-up
2	23	10.0	2.30
4	70	15.0	4.67
8	212	22.0	9.64
16	620	31.9	19.44
32	1740	52.8	32.95

The size of the grid was chosen sufficiently large not to slow the computation. We observe a quick increase of speed-ups as long as the neighborhood radius does not influence the allocation too much. Beyond a certain input size we see that the potential quadratic increase of the speed-up is not yet met on this example with a naive migration strategy.

5 Conclusion

We introduced a parallel computation model with an entirely localized reduction on top of a memory scheme with a limited connectivity and a locally bounded storage and computation capacity. We run functional programs on this model and showed that reasonable memory management strategies can also be implemented locally.

We plan to incorporate new features such as nested pattern matching [9, 7] to our input language and support more traditional programming syntaxes.

Despite the relatively smooth executions obtained with a very simplistic memory allocation strategy, there is plenty of room for efficiency improvements that would reduce the total migration costs. Integration with complexity-analysis techniques such as [13, 6] should help to implement really accurate allocation strategies.

We only considered 2-dimensional uniform grid supports in our experimentation. In the future, we would also like to support 3 or n dimensional grids and more elaborate topologies.

A last remaining challenge is to use a minimal set of symbols and reduction rules. We know that there exist interaction-net systems with a very restricted set of symbols that are universal [12]. In particular preliminary investigations have been made concerning the usage of such minimal sets of symbols to specifically encode functional programs by means of linear logic [17, 4, 5].

References

1 Andrea Asperti, Cecilia Giovannetti, and Andrea Naletto. The bologna optimal higher-order machine. *Journal of Functional Programming*, 6(6):763–810, 1996. `doi:10.1017/S0956796800001994`.

2 Horatiu Cirstea, Germain Faure, Maribel Fernandez, Ian Mackie, and François-Régis Sinot. From functional programs to interaction nets via the rewriting calculus. *Electronic Notes in Theoretical Computer Science*, 174(10):39–56, 2007.

3 Maribel Fernández, Ian Mackie, Shinya Sato, and Matthew Walker. Recursive functions with pattern matching in interaction nets. *ENTCS*, 253(4):55–71, 2009. `doi:10.1016/j.entcs.2009.10.017`.

4 Stéphane Gimenez. *Programmer, calculer et raisonner avec les réseaux de la logique linéaire*. PhD thesis, 2009. URL: `http://pps.jussieu.fr/~gimenez/these.html`.

5 Stéphane Gimenez. Towards generic inductive constructions in systems of nets. In *13th International Workshop on Termination (WST 2013)*, page 51, 2013.

6 Stéphane Gimenez and Georg Moser. The complexity of interaction. In *POPL*, pages 243–255. ACM, 2016. doi:10.1145/2837614.2837646.

7 Abubakar Hassan, Eugen Jiresch, and Shinya Sato. An implementation of nested pattern matching in interaction nets. *arXiv preprint arXiv:1003.4562*, 2010.

8 Abubakar Hassan, Ian Mackie, and Shinya Sato. Interaction nets: programming language design and implementation. *Electronic Communications of the EASST*, 10, 2008.

9 Abubakar Hassan and Shinya Sato. Interaction nets with nested pattern matching. *Electronic Notes in Theoretical Computer Science*, 203(1):79–92, 2008.

10 Eugen Jiresch. Towards a GPU-based implementation of interaction nets. In *DCM*, pages 41–53, 2014. doi:10.4204/EPTCS.143.4.

11 Yves Lafont. Interaction nets. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 95–108. ACM, 1989.

12 Yves Lafont. Interaction combinators. *Information and Computation*, 137(1):69–101, 1997. doi:10.1006/inco.1997.2643.

13 Ugo Dal Lago. A short introduction to implicit computational complexity. In *Lectures on Logic and Computation – ESSLLI 2010, ESSLLI 2011*, pages 89–109, 2011. doi:10.1007/978-3-642-31485-8_3.

14 Sylvain Lippi. Universal hard interaction for clockless computation. Dem Glücklichen schlägt keine Stunde! *Fundamenta Informaticae*, 91(2):357–394, 2009. doi:10.3233/FI-2009-0048.

15 Ian Mackie. Interaction nets for linear logic. *Theoretical Computer Science*, 247(1-2):83–140, 2000. doi:10.1016/S0304-3975(00)00198-5.

16 Ian Mackie. Efficient lambda-evaluation with interaction nets. In *RTA*, volume 3091 of *Lecture Notes in Computer Science*, pages 155–169, 2004. doi:10.1007/b98160.

17 Ian Mackie and Jorge Sousa Pinto. Encoding linear logic with interaction combinators. *Information and Computation*, 176(2):153–186, 2002. doi:10.1006/inco.2002.3163.

18 Damiano Mazza. Multiport interaction nets and concurrency. In *CONCUR*, pages 21–35, 2005. doi:10.1007/11539452_6.

19 Damiano Mazza. *Interaction Nets: Semantics and Concurrent Extensions*. PhD thesis, 2006. URL: https://www-lipn.univ-paris13.fr/~mazza/papers/Thesis.pdf.

20 Jorge Sousa Pinto. Sequential and concurrent abstract machines for interaction nets. In *FOSSACS*, pages 267–282. Springer-Verlag, 2000. doi:10.1007/3-540-46432-8_18.

21 Jorge Sousa Pinto. Parallel evaluation of interaction nets with MPINE. In *RTA*, pages 353–356, 2001. doi:10.1007/3-540-45127-7_26.

22 Jorge Sousa Pinto. Parallel implementation models for the lambda-calculus using the geometry of interaction. In *TLCA*, pages 385–399, 2001. doi:0.1007/3-540-45413-6_30.

Automating the First-Order Theory of Rewriting for Left-Linear Right-Ground Rewrite Systems*

Franziska Rapp[1] **and Aart Middeldorp**[2]

1 **Department of Computer Science, University of Innsbruck, Innsbruck, Austria**
franziska.rapp@uibk.ac.at

2 **Department of Computer Science, University of Innsbruck, Innsbruck, Austria**
aart.middeldorp@uibk.ac.at

Abstract

The first-order theory of rewriting is decidable for finite left-linear right-ground rewrite systems. We present a new tool that implements the decision procedure for this theory. It is based on tree automata techniques. The tool offers the possibility to synthesize rewrite systems that satisfy properties that are expressible in the first-order theory of rewriting.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases first-order theory, ground rewrite systems, automation, synthesis

Digital Object Identifier 10.4230/LIPIcs.FSCD.2016.36

1 Introduction

Dauchet and Tison [4] proved that the first-order theory of rewriting is decidable for ground rewrite systems. In this theory one can express properties like confluence

$$\forall\, s\, \forall\, t\, \forall\, u\, (s \to^* t \,\wedge\, s \to^* u \implies \exists\, v\, (t \to^* v \,\wedge\, u \to^* v))$$

and normalization

$$\forall\, s\, \exists\, t\, (s \to^* t \,\wedge\, \neg\, \exists\, u\, (t \to u))$$

The decision procedure in [4] is based on tree automata techniques and easily extends to left-linear right-ground rewrite systems. Key ingredients are tree automata that accept encodings of n-ary relations on terms, ground tree transducers for certain binary relations on terms, as well as several closure properties corresponding to the logical operations of the first-order theory of rewriting.

We implemented the decision procedure in the new *First-Order Rewriting Tool*, FORT for short. It takes as input a left-linear right-ground TRS $\mathcal{R}$ and a first-order formula φ and reports whether $\mathcal{R}$ satisfies the property expressed by φ. We extended the theory with additional predicates in order to increase expressibility. For instance, the formula

$$\forall\, s\, \forall\, t\, \forall\, u\, (s \to^! t \,\wedge\, s \mathrel{+\!\!\!\!\!\rightarrow} u \implies t = u \,\vee\, \exists\, v\, (u \to^+ v \,\wedge\, v \xrightarrow{\epsilon} t))$$

can be handled by our tool. In addition, FORT can be used to synthesize left-linear right-ground rewrite systems that satisfy a property expressible in the first-order theory of rewriting, using a simple generate-and-test algorithm. Additional input parameters guide the search.

* This work is supported by FWF (Austrian Science Fund) project P27528.

1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016).
Editors: Delia Kesner and Brigitte Pientka; Article No. 36; pp. 36:1–36:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The remainder of this paper is organized as follows. In Section 2 the first-order theory of rewriting is formally defined and the supported predicates are presented. The tree automata techniques underlying the decision procedure implemented in FORT are covered in Section 3 and in Section 4 we explain how formulas are mapped to the constructions of the preceding section. Synthesis is the topic of Section 5. In Section 6 we briefly explain how FORT is used and present some implementation details. The power of FORT is illustrated by several experiments in Section 7. We conclude in Section 8 with related work and ideas for future work.

2 First-Order Theory of Rewriting

We assume familiarity with term rewriting [1]. We consider first-order logic over a language $\mathcal{L}$ without function symbols. The language contains the following binary predicate symbols:

$$\to \qquad \to^+ \qquad \to^* \qquad \to^! \qquad \mathrel{+\!\!\!\!\!\rightarrow} \qquad \xrightarrow{\epsilon} \qquad \xrightarrow{>\epsilon} \qquad \leftrightarrow \qquad \leftrightarrow^* \qquad \downarrow \qquad =$$

As models we consider finite TRSs $(\mathcal{F},\mathcal{R})$ such that the set of ground terms $\mathcal{T}(\mathcal{F})$ is non-empty, which is equivalent to the requirement that the signature $\mathcal{F}$ contains at least one constant symbol. The set of ground terms serves as domain for the variables in formulas over $\mathcal{L}$. The interpretation of the predicate symbol $\to$ in $(\mathcal{F},\mathcal{R})$ is the one-step rewrite relation $\to_{\mathcal{R}}$ over $\mathcal{T}(\mathcal{F})$. The other predicate symbols are interpreted in a similar fashion: $\to^!$ denotes normalization: $s \to^!_{\mathcal{R}} t$ if $s \to^*_{\mathcal{R}} t$ and t is a normal form of $\mathcal{R}$, $\mathrel{+\!\!\!\!\!\rightarrow}$ denotes a parallel rewrite step, $\xrightarrow{\epsilon}$ denotes one-step rewriting at the root position, $\xrightarrow{>\epsilon}$ denotes one-step rewriting below the root, and $\leftrightarrow^*$ denotes convertibility. The predicate symbols $\to^*$, $\to^!$, $\leftrightarrow$, and $\downarrow$ can be expressed in terms of the other predicate symbols:

$$\begin{aligned} s \to^* t &\iff s \to^+ t \lor s = t & \qquad s \leftrightarrow t &\iff s \to t \lor t \to s \\ s \to^! t &\iff s \to^* t \land \neg\exists\, u\,(t \to u) & \qquad s \downarrow t &\iff \exists\, u\,(s \to^* u \land t \to^* u) \end{aligned}$$

We included them for convenience. For the same reason, $\mathcal{L}$ contains symbols denoting standard properties of (terms of) TRSs:

$$\begin{aligned} \mathsf{CR}(t) &\iff \forall\, u\,\forall\, v\,(t \to^* u \land t \to v \implies u \downarrow v) & \qquad \mathsf{CR} &\iff \forall\, t\ \mathsf{CR}(t) \\ \mathsf{WCR}(t) &\iff \forall\, u\,\forall\, v\,(t \to u \land t \to v \implies u \downarrow v) & \qquad \mathsf{WCR} &\iff \forall\, t\ \mathsf{WCR}(t) \\ \mathsf{WN}(t) &\iff \exists\, u\,(t \to^! u) & \qquad \mathsf{WN} &\iff \forall\, t\ \mathsf{WN}(t) \\ \mathsf{UN}(t) &\iff \forall\, u\,\forall\, v\,(t \to^! u \land t \to^! v \implies u = v) & \qquad \mathsf{UN} &\iff \forall\, t\ \mathsf{UN}(t) \\ \mathsf{NFP}(t) &\iff \forall\, u\,\forall\, v\,(t \to u \land t \to^! v \implies u \to^! v) & \qquad \mathsf{NFP} &\iff \forall\, t\ \mathsf{NFP}(t) \end{aligned}$$

$$\mathsf{NF}(t) \iff \neg\exists\, u\,(t \to u) \qquad \mathsf{UNC} \iff \forall\, t\,\forall\, u\,(t \leftrightarrow^* u \land \mathsf{NF}(t) \land \mathsf{NF}(u) \implies t = u)$$

The acronyms stand for the Church-Rosser property or confluence (CR), the weak Church-Rosser property or local confluence (WCR), (weak) normalization (WN), unique normal forms (UN), the normal form property (NFP), and unique normal forms with respect to conversion (UNC). On the other hand, the unary predicate symbol $\mathsf{Fin}_\circ$ defined as

$$\mathsf{Fin}_\circ(t) \iff \{u \mid t \circ u\} \text{ is finite}$$

for every boolean combination $\circ$ of the binary predicate symbols was added to $\mathcal{L}$ as it strictly increases expressibility. For example, it allows to express the important termination (strong normalization) property:

$$\begin{aligned} \mathsf{SN}(t) &\iff \mathsf{Fin}_{\to^+}(t) \land \neg\exists\, u\,(t \to^* u \land u \to^+ u) \\ \mathsf{SN} &\iff \forall\, t\ \mathsf{Fin}_{\to^+}(t) \land \neg\exists\, u\,(u \to^+ u) \end{aligned}$$

It should be stressed that the above properties are restricted to *ground* terms. So CR stands for ground-confluence, which is different from confluence, even in the presence of ground terms; consider e.g. the TRS consisting of the rules $\mathsf{f}(x) \to x$ and $\mathsf{f}(x) \to \mathsf{c}$.

3 Tree Automata

Most of the results presented in this section are well-known. We refer to [3] for further details. We consider bottom-up tree automata $\mathcal{A} = (\mathcal{F}, Q, Q_f, \Delta)$ consisting of a finite signature $\mathcal{F}$ with $\mathcal{T}(\mathcal{F}) \neq \varnothing$, a finite set Q of states with $Q \cap \mathcal{F} = \varnothing$, a set $Q_f \subseteq Q$ of final states, and a set Δ of transition rules. Every transition rule has one of the following two shapes:

- $f(\alpha_1, \ldots, \alpha_n) \to \beta$ with $f \in \mathcal{F}$ and $\alpha_1, \ldots, \alpha_n, \beta \in Q$, or
- $\alpha \to \beta$ with $\alpha, \beta \in Q$.

Transition rules of the second shape are called ϵ-transitions. We view $\mathcal{A}$ as a ground TRS $(\mathcal{F} \cup Q, \Delta)$. The induced rewrite relation is denoted by $\to_{\mathcal{A}}$ or simply $\to$ if $\mathcal{A}$ can be inferred from the context. The language $L(\mathcal{A})$ accepted by $\mathcal{A}$ is the set of ground terms in $\mathcal{T}(\mathcal{F})$ that can be rewritten to a final state. A subset $T \subseteq \mathcal{T}(\mathcal{F})$ is *regular* if there exists a tree automaton $\mathcal{A}$ with $L(\mathcal{A}) = T$.

Every regular language is accepted by a tree automaton $\mathcal{A}$ without ϵ-transitions. Moreover, it may be assumed that $\mathcal{A}$ is *deterministic* (no two different transition rules of $\mathcal{A}$ have the same left-hand side) and *completely defined* (for every $f(\alpha_1, \ldots, \alpha_n)$ there exists a transition rule with this left-hand side).

▶ **Theorem 1.** *The class of regular languages is effectively closed under union, intersection, and complement.*

The constructions employed in the proof are well-known. In the decision procedure for the first-order theory of rewriting for left-linear right-ground TRSs they are used to encode the propositional connectives $\wedge$, $\vee$, and $\neg$.

▶ **Theorem 2.** *The following problems are decidable:*

instance: *a tree automaton* $\mathcal{A}$ *and a ground term* t
question: $t \in L(\mathcal{A})$*?*

instance: *a tree automaton* $\mathcal{A}$
question: $L(\mathcal{A}) = \varnothing$*?*

instance: *tree automata* $\mathcal{A}$ *and* $\mathcal{B}$
question: $L(\mathcal{A}) \subseteq L(\mathcal{B})$*?*

To cope with the predicate symbols in our language we use tree automata that operate on n-tuples of ground terms.

▶ **Definition 3.** For a signature $\mathcal{F}$ we let $\mathcal{F}^{(n)} = (\mathcal{F} \cup \{\bot\})^n$. Here, $\bot \notin \mathcal{F}$ is a fresh constant. The arity of a symbol $f_1 \cdots f_n \in \mathcal{F}^{(n)}$ is the maximum of the arities of $f_1, \ldots, f_n$. Given terms $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$, the term $\langle t_1, \ldots, t_n \rangle \in \mathcal{T}(\mathcal{F}^{(n)})$ is defined inductively:

$$\langle t_1, \ldots, t_n \rangle = \begin{cases} t_1 \cdots t_n & \text{if } t_1, \ldots, t_n \text{ are constants} \\ f(u_1, \ldots, u_m) & \text{otherwise} \end{cases}$$

where $f = \mathsf{root}(t_1) \cdots \mathsf{root}(t_n)$ has arity m and $u_i = \langle s_1, \ldots, s_n \rangle$ with

$$s_j = \begin{cases} t_j|_i & \text{if } i \in \mathcal{P}\mathsf{os}(t_j) \\ \bot & \text{otherwise} \end{cases}$$

The following example illustrates the above definition.

▶ **Example 4.** Consider the ground terms $s = \mathsf{f}(\mathsf{g}(\mathsf{a}), \mathsf{f}(\mathsf{b}, \mathsf{b}))$, $t = \mathsf{g}(\mathsf{g}(\mathsf{a}))$, and $u = \mathsf{f}(\mathsf{b}, \mathsf{g}(\mathsf{a}))$. We have $\langle s, t, u \rangle = \mathsf{fgf}(\mathsf{ggb}(\mathsf{aa}\bot), \mathsf{f}\bot\mathsf{g}(\mathsf{b}\bot\mathsf{a}, \mathsf{b}\bot\bot))$.

▶ **Definition 5.** A *regular relation* is an n-ary relation R on ground terms such that its encoding $\{\langle t_1, \ldots, t_n \rangle \mid (t_1, \ldots, t_n) \in R\}$ is regular. The class of all n-ary regular relations is denoted by RR_n.

An RR_1 relation is just a regular language. The following result is an immediate consequence of Theorem 1.

▶ **Theorem 6.** *The class of regular relations is effectively closed under boolean operations.*

▶ **Definition 7.** Let R be an n-ary relation over $\mathcal{T}(\mathcal{F})$.

- If $n \geqslant 2$ and $1 \leqslant i \leqslant n$ then the i-th *projection* $\Pi_i(R)$ of R is the relation

$$\{(t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n) \mid (t_1, \ldots, t_n) \in R\}$$

- If $1 \leqslant i \leqslant n+1$ then the i-th *cylindrification* $C_i(R)$ of R is the relation

$$\{(t_1, \ldots, t_{i-1}, u, t_i, \ldots, t_n) \mid (t_1, \ldots, t_n) \in R \text{ and } u \in \mathcal{T}(\mathcal{F})\}$$

- If σ is a permutation on $\{1, \ldots, n\}$ then $\sigma(R)$ denotes the relation

$$\{(t_{\sigma(1)}, \ldots, t_{\sigma(n)}) \mid (t_1, \ldots, t_n) \in R\}$$

▶ **Theorem 8.** *The class of regular relations is effectively closed under projection, cylindrification, and permutation.*

In the decision procedure projection is used to encode existential quantification, whereas cylindrification is used to combine relations of different arity. An important limitation of binary regular relations is that they are not closed under transitive closure [3, Exercise 3.1].

▶ **Definition 9.** A *ground tree transducer* (*GTT* for short) is a pair $\mathcal{G} = (\mathcal{A}, \mathcal{B})$ of tree automata over the same signature $\mathcal{F}$. A pair of ground terms (s, t) is accepted by $\mathcal{G}$ if $s \rightarrow_{\mathcal{A}}^{*} \cdot {}_{\mathcal{B}}^{*}\!\!\leftarrow t$. The set of all accepted pairs is denoted by $R(\mathcal{G})$. A binary relation R on ground terms is a *GTT relation* if $R = R(\mathcal{G})$ for some GTT.

▶ **Theorem 10.** *The class of GTT relations is effectively closed under inverse and transitive closure.*

▶ **Theorem 11.** *If $\mathcal{G}_1 = (\mathcal{A}_1, \mathcal{B}_1)$ and $\mathcal{G}_2 = (\mathcal{A}_2, \mathcal{B}_2)$ are GTTs with disjoint sets of states and $\mathcal{G} = (\mathcal{A}_1 \cup \mathcal{A}_2, \mathcal{B}_1 \cup \mathcal{B}_2)$ then $R(\mathcal{G})^+ = (R(\mathcal{G}_1) \cup R(\mathcal{G}_2))^+$.*

▶ **Lemma 12.** *Every GTT relation is a regular relation.*

The final result in this section is an easy generalization of a result of Dauchet and Tison [4, 5]. The proof is given in the appendix.

▶ **Lemma 13.** *If R is a binary regular relation then Fin_R is a regular predicate.*

4 Automation

In this section we explain how first-order formulas are translated into tree automata manipulations. We start the discussion with the binary relation symbols of $\mathcal{L}$. Throughout this section, $\mathcal{R}$ denotes a finite left-linear right-ground TRS over a finite signature $\mathcal{F}$.

▶ **Lemma 14.** *The relations* $=$, $\to_{\mathcal{R}}$, $\xrightarrow{\epsilon}_{\mathcal{R}}$, *and* $\xrightarrow{>\epsilon}_{\mathcal{R}}$ *are regular.*

Rather than giving the proof, we illustrate the constructions on a simple example.

▶ **Example 15.** Consider the TRS $\mathcal{R}$ consisting of the rules $\mathsf{f}(x,\mathsf{a}) \to \mathsf{g}(\mathsf{b})$ and $\mathsf{g}(\mathsf{a}) \to \mathsf{f}(\mathsf{a},\mathsf{b})$. The (encoding of the) identity relation $=$ is accepted by the tree automaton

$$\mathsf{aa} \to \top \qquad\qquad \mathsf{bb} \to \top \qquad\qquad \mathsf{gg}(\top) \to \top \qquad\qquad \mathsf{ff}(\top,\top) \to \top$$

For $\xrightarrow{\epsilon}_{\mathcal{R}}$ we use the following tree automaton (with $\checkmark$ as unique final state):

$$\begin{array}{rrrr}
\mathsf{fg}(\alpha,\beta) \to \checkmark & \mathsf{ab} \to \alpha & \mathsf{a}\bot \to \gamma & \mathsf{gf}(\delta,\epsilon) \to \checkmark \\
\mathsf{a}\bot \to \beta & \mathsf{bb} \to \alpha & \mathsf{b}\bot \to \gamma & \mathsf{aa} \to \delta \\
 & \mathsf{gb}(\gamma) \to \alpha & \mathsf{g}\bot(\gamma) \to \gamma & \bot\mathsf{b} \to \epsilon \\
 & \mathsf{fb}(\gamma,\gamma) \to \alpha & \mathsf{f}\bot(\gamma,\gamma) \to \gamma &
\end{array}$$

By combining the previous two automata and adding the transitions

$$\mathsf{gg}(\checkmark) \to \checkmark \qquad\qquad \mathsf{ff}(\checkmark,\top) \to \checkmark \qquad\qquad \mathsf{ff}(\top,\checkmark) \to \checkmark$$

we obtain a tree automaton that accepts $\to_{\mathcal{R}}$. For $\xrightarrow{>\epsilon}_{\mathcal{R}}$ we add instead

$$\begin{array}{ccc}
\mathsf{gg}(\checkmark) \to \checkmark' & \mathsf{ff}(\checkmark,\top) \to \checkmark' & \mathsf{ff}(\top,\checkmark) \to \checkmark' \\
\mathsf{gg}(\checkmark') \to \checkmark' & \mathsf{ff}(\checkmark',\top) \to \checkmark' & \mathsf{ff}(\top,\checkmark') \to \checkmark'
\end{array}$$

and make $\checkmark'$ the unique final state.

▶ **Lemma 16.** *The relation* $\mathrel{\Vert\!\!\!\to}_{\mathcal{R}}$ *is a GTT relation.*

▶ **Example 17.** The GTT $\mathcal{G} = (\mathcal{A},\mathcal{B})$ with

$$\begin{array}{ccccc|ccc}
\mathcal{A} & \mathsf{a} \to \alpha & \mathsf{g}(\alpha) \to \alpha & \mathsf{a} \to \beta & \mathsf{f}(\alpha,\beta) \to \iota_1 & \mathsf{g}(\gamma) \to \iota_1 & \mathsf{a} \to \delta & \mathcal{B} \\
 & \mathsf{b} \to \alpha & \mathsf{f}(\alpha,\alpha) \to \alpha & & \mathsf{g}(\beta) \to \iota_2 & \mathsf{f}(\delta,\gamma) \to \iota_2 & \mathsf{b} \to \gamma &
\end{array}$$

accepts the relation $\mathrel{\Vert\!\!\!\to}_{\mathcal{R}}$ for the TRS $\mathcal{R}$ of the previous example.

▶ **Lemma 18.** *The relations* $\to_{\mathcal{R}}^{*}$, $\to_{\mathcal{R}}^{+}$, *and* $\leftrightarrow_{\mathcal{R}}^{*}$ *are regular relations.*

Proof. The relation $\to_{\mathcal{R}}^{*} = \mathrel{\Vert\!\!\!\to}_{\mathcal{R}}^{+}$ is a GTT relation according to Lemma 16 and Theorem 10, and hence regular according to Lemma 12. To obtain the regularity of $\to_{\mathcal{R}}^{+}$, the construction in the proof of Lemma 12 [3, Proposition 3.2.7] is slightly modified. For $\leftrightarrow_{\mathcal{R}}^{*} = (\mathrel{\Vert\!\!\!\to}_{\mathcal{R}} \cup {}_{\mathcal{R}}\!\mathrel{\leftarrow\!\!\!\Vert})^{+}$ we additionally use Theorem 11. ◀

Note that we cannot use the equivalence of $\leftrightarrow_{\mathcal{R}}^{*}$ and $(\to_{\mathcal{R}} \cup {}_{\mathcal{R}}\!\leftarrow)^{+} \cup =$ in the above proof since $\to_{\mathcal{R}} \cup {}_{\mathcal{R}}\!\leftarrow$ is not a GTT relation and binary regular relations are not closed under transitive closure.

For the remaining binary relations we use the formulas given in Section 2 in combination with the closure properties of Section 3. We have to be careful when combining relations with union and intersection because there may be a mismatch in the dimension of the involved relations. Consider the binary relations $R_1 = \{(s,t) \mid s \to t\}$ and $R_2 = \{(t,u) \mid t \to u\}$. Their union is the ternary relation $R_1 \cup R_2 = \{(s,t,u) \mid s \to t \text{ or } t \to u\}$, so before applying the union construction we have to cylindrify R_1 and R_2 suitably. Also permutation is needed if the order of terms in the participating relations is different.

Existential quantification is implemented using projection (Definition 7 and Theorem 8). Formulas are normalized in such a way that implications and universal quantifications are eliminated using the known equivalences ($\varphi \Rightarrow \psi \equiv \neg\,\varphi \vee \psi$ and $\forall\,x\,\varphi \equiv \neg\,\exists\,x\,\neg\varphi$). Furthermore, negations are pushed inside and double negations are eliminated. We do not transform formulas into prenex normal form since this typically increases the dimension of the relations, with a corresponding increase in the computation time for the closure operations.

We illustrate the decision procedure on a small example.

▶ **Example 19.** Suppose we want to determine whether a given left-linear right-ground TRS $\mathcal{R}$ is (weakly) normalizing. We use the formula $\phi = \forall\,s\,\exists\,t\,(s \to^* t \wedge \neg\,\exists\,u\,(t \to u))$ for this purpose. The computed (tree automata representations of the) relations are listed below:

1. GTT relation for $\{(s,t) \mid s \mathrel{-\!\!\|\!\!\!\to}_{\mathcal{R}} t\}$ (Lemma 16)
2. GTT relation for $\{(s,t) \mid s \to_{\mathcal{R}}^* t\}$ (Theorem 10)
3. RR_2 relation for $\{(s,t) \mid s \to_{\mathcal{R}}^* t\}$ (Lemma 12)
4. RR_2 relation for $\{(t,u) \mid t \to_{\mathcal{R}} u\}$ (Lemma 14)
5. RR_1 relation for $\{t \mid \exists\,u\,(t \to_{\mathcal{R}} u)\}$ (projection Π_2)
6. RR_1 relation for $\{t \mid \neg\,\exists\,u\,(t \to_{\mathcal{R}} u)\}$ (complement)
7. RR_2 relation for $\{(s,t) \mid \neg\,\exists\,u\,(t \to_{\mathcal{R}} u)\}$ (cylindrification C_1)
8. RR_2 relation for $\{(s,t) \mid s \to_{\mathcal{R}}^* t \wedge \neg\,\exists\,u\,(t \to_{\mathcal{R}} u))\}$ (intersection)
9. RR_1 relation for $\{s \mid \exists\,t\,(s \to_{\mathcal{R}}^* t \wedge \neg\,\exists\,u\,(t \to_{\mathcal{R}} u))\}$ (projection Π_2)
10. RR_1 relation for $\{s \mid \neg\,\exists\,t\,(s \to_{\mathcal{R}}^* t \wedge \neg\,\exists\,u\,(t \to_{\mathcal{R}} u))\}$ (complement)

Then $\mathcal{R}$ satisfies ϕ if and only if the final language is empty (Theorem 2).

The logic supports the shorter formula $\forall\,s\,\exists\,t\,(s \to^! t)$ to test for normalization. The relation $\{(s,t) \mid s \to^! t\}$ is implemented as $\{(s,t) \mid s \to^* t \wedge \mathsf{NF}(t)\}$. For the NF predicate our tool FORT employs the explicit construction presented in Comon [2, Section 4.2] rather than the implicit encoding $\neg\,\exists\,u\,(t \to u)$ given in the preceding example. For small TRSs there is little difference, but if the left-hand sides have a larger depth or contain functions symbols of a higher arity, the explicit construction is more efficient.

5 Synthesis

Our tool can also be used to synthesize TRSs that satisfy properties given by the user. This may be interesting for finding counterexamples and non-trivial TRSs for exam exercises as well as competitions. Furthermore, we envisage the use of FORT in courses on term rewriting.

The synthesis algorithm for *ground* TRSs is shown in Figure 1. Of the five input parameters, the formula ϕ must be supplied whereas the other four are optional. If the signature $\mathcal{F}$ is not given, we enumerate all finite signatures in a dovetailing manner and call the procedure repeatedly. The default values for the other optional parameters are $n = 3$, $m = 4$, and $d = m - 1$. The outer while loop incrementally constructs the signature $\mathcal{F}$. The current candidate $\mathcal{G} \subseteq \mathcal{F}$ is used to compute the set of terms T from which the rewrite rules are constructed. Terms in T must satisfy the given depth and size restrictions. In the for loop we enumerate all candidate ground TRSs that have no more than n rules, starting from the ones with the fewest rules. In the inner while loop we consider the set of ground TRSs R' consisting of i rewrite rules. We select the smallest $\mathcal{R} \in R'$ in some fixed total order $>$ on TRSs and test whether $\mathcal{R}$ is *normalized*, which means in this context that no smaller

```
Input:   • closed first-order formula $\phi$ over $\mathcal{L}$
         • signature $\mathcal{F}$                    • maximum size of terms $m$
         • maximum number of rules $n$                • maximum depth of terms $d$
Output:  • TRS $\mathcal{R}$ satisfying $\phi$
$\mathcal{G} := \varnothing$;
while $\mathcal{F} \setminus \mathcal{G} \neq \varnothing$ do
    $\mathcal{G} := \mathcal{G} \cup \{(f,a)\}$ with $(f,a) \in \mathcal{F} \setminus \mathcal{G}$ of minimum arity $a$;
    $T := \{t \in \mathcal{T}(\mathcal{G}) \mid |t| \leqslant m$ and $\mathsf{depth}(t) \leqslant d\}$;
    $R := T \times T$;
    for $i = 0$ to $\min(n, |R|)$ do
        $R' := \{S \subseteq R \mid S$ consists of $i$ pairs$\}$;
        while $R' \neq \varnothing$ do
            select smallest $\mathcal{R} \in R'$ (with respect to some total order $>$);
            if $\mathcal{R}$ is normalized and $\mathcal{R} \vDash \phi$ then return $\mathcal{R}$;
            $R' := R' \setminus \{\mathcal{R}\}$
        od
    od
od;
return failure
```

Figure 1 Procedure for synthesizing TRSs.

(with respect to $>$) TRS exists (in the original R') that can be obtained from $\mathcal{R}$ by renaming constant symbols (within $\mathcal{G}$). This test is done to avoid calling the decision tool on a TRS for which an earlier call on a renamed version of the TRS was unsuccessful. If $\mathcal{R}'$ passes this test we use the decision tool to determine whether $\mathcal{R}$ satisfies the formula ϕ.

The procedure is easily extended to left-linear right-ground TRSs. We have an additional optional input parameter v for the number of variables. Its default value is one. A set T' of linear terms with variables sorted in some fixed order (to keep the set small) is computed and the set of candidate TRSs is extended to $T' \times T$.

6 Interface

Precompiled binaries to run FORT from the command line are available from

`http://cl-informatik.uibk.ac.at/software/FORT`

FORT can be used as decision tool via the following command

```
./fort -D <file> [<verbosity>] "<formula>"
```

where `<formula>` specifies the property that should be tested on the TRS given in `<file>`. The syntax of TRSs follows the standard TPDB format.[1] The optional `<verbosity>`

[1] `https://www.lri.fr/~marche/tpdb/format.html`

parameter takes a value in $\{0, 1, 2\}$. The higher the value, the more detailed the output will be. The default value is 0. The syntax of the formula is explained in the file `description.txt`. The quotation marks around the formula are essential. The basic command to synthesize TRSs is

```
./fort -S <file>
```

where <file> must contain the line

```
(FORMULA <formula>)
```

and may contain the lines

```
(SIGNATURE <F>)   (VARIABLES <V>)   (RULES <R>)   (SIZE <S>)   (DEPTH <D>)
```

corresponding to the optional parameters that were described in the preceding section. The signature `<F>` is specified as a comma separated list of function symbols and arities. For example, `(a 0, g 1, h 1)` specifies a constant `a` and two unary symbols `g` and `h`. The information in `<file>` can also be passed to FORT via the command line options

```
-f "<F>"        -v <V>        -r <R>        -s <S>        -d <D>
```

7 Experiments

In this section we illustrate FORT on a number of examples. After reporting on some experiments we performed with the decision tool, we turn to the synthesis part.

The combined confluence[2] and termination[3] problem databases contain 65 left-linear right-ground TRSs. When testing for confluence, FORT reports that 42 of these TRSs are (ground-)confluent, 14 are non-(ground-)confluent, and the remaining 9 TRSs exceed the given time limit of 60 seconds. (Ground-confluence coincides with confluence on our collection of TRSs.) Not surprisingly, designated confluence provers[4] like ACP, CSI, and Saigawa are considerably faster than FORT, but only CSI subsumes FORT on the 65 left-linear right-ground TRSs (45/19/1 in less than 2 minutes, compared to the 8 minutes of FORT). Next we compare the following three different but equivalent formulations of confluence:

```
forall s, t, u (s ->* t & s ->* u => t join u)       631s, 10 timeouts    (1)
forall s, t, u (s ->* t & s -> u => t join u)        484s,  9 timeouts    (2)
forall t, u (t <->* u => t join u)                   895s, 13 timeouts    (3)
```

The total time spent by FORT (using a 60 seconds time limit) is given in the right column. The computation of $\leftrightarrow^*$ is much more expensive than $\rightarrow^*$. Moreover, intersecting the RR_3 relations $R_1 = \{(s,t,u) \mid s \rightarrow^* t\}$ and $R_2 = \{(s,t,u) \mid s \rightarrow^* u\}$ is more expensive than intersecting R_1 with $R_3 = \{(s,t,u) \mid s \rightarrow u\}$. The numbers explain why the second formula (called *semi-confluence* in [1]) is selected as translation of the predicate CR.

[2] http://cops.uibk.ac.at/
[3] http://termination-portal.org/wiki/TPDB
[4] http://coco.nue.riec.tohoku.ac.jp/tools/

Table 1 Intermediate automata sizes for TRS #74 of the confluence database.

property	$\to$	$\to^*$	$\leftrightarrow^*$	$\cap_1$	$\cap_2$	$\cap_3$	$\cap_4$	time
(1)		82		1162	2184	33088	1723	5.2s
(2)	37	82		472	2184	15187	910	2.1s
(3)		82	1634		2184	43076	19237	27.1s

To give an idea of the sizes of the computed tree automata, we present detailed information in Table 1 for TRS #74

$$\begin{array}{llll}
\mathsf{a} \to \mathsf{c} & \mathsf{f}(x,\mathsf{c}) \to \mathsf{f}(\mathsf{c},\mathsf{c}) & \mathsf{d} \to \mathsf{f}(\mathsf{a},\mathsf{c}) & \mathsf{f}(\mathsf{a},\mathsf{b}) \to \mathsf{d} \\
\mathsf{b} \to \mathsf{c} & \mathsf{f}(\mathsf{c},x) \to \mathsf{f}(\mathsf{c},\mathsf{c}) & \mathsf{d} \to \mathsf{f}(\mathsf{c},\mathsf{b}) &
\end{array}$$

of the confluence database. Here $\cap_1$ corresponds to the conjunction in the subformula that represents the peak (absent for property (3)) and $\cap_2$ the one from the join $t \downarrow u$. The next two columns correspond to the implication in the three properties before ($\cap_3$) and after ($\cap_4$) trimming the automata. As can be seen, trimming intermediate automata, although time consuming, has a significant impact.

We performed a similar experiment for the normal form property (NFP):

`forall s, t, u (s -> t & s ->! u => t ->* u)`	187s, 4 timeouts
`forall t, u (t <->* u & NF(u) => t ->* u)`	863s, 13 timeouts
`forall t (WN(t) => CR(t))`	496s, 7 timeouts

justifying the selection of the first formula.

The time required to synthesize a TRS varies vastly, depending on the property, the additional options, and the size of the smallest TRS fulfilling the property. Hence, the options should be chosen with care.

▶ **Example 20.** We start with looking for non-confluent TRSs having unique normal forms (`UN & ~CR`). With the option (`VARIABLES 0`) the ground TRS

$$\mathsf{a} \to \mathsf{a} \qquad\qquad \mathsf{b} \to \mathsf{a} \qquad\qquad \mathsf{b} \to \mathsf{g}(\mathsf{a})$$

is generated in about 18 seconds. Generating a one-rule TRS with the option (`RULES 1`) produces $\mathsf{g}(\mathsf{g}(x)) \to \mathsf{h}(\mathsf{g}(\mathsf{g}(\mathsf{a})))$ in a bit more than 2 seconds.

▶ **Example 21.** Next we consider the formula

$\neg\forall t\ \mathsf{Fin}_{\overset{\epsilon}{\leftarrow}}(t)$ `(FORMULA ~forall t Fin(e<-,t))`

which distinguishes ground TRSs from left-linear right-ground (but not ground) ones. Without any options FORT produces the single rule $\mathsf{g}(x) \to \mathsf{c}$ in a fraction of a second. The formula

$\neg\forall t\ \mathsf{Fin}_{\neq}(t)$ `(FORMULA ~forall t Fin(~=,t))`

is true for TRSs that are not ARSs. FORT produces the empty TRS over the signature consisting of two constants and a unary function symbol. To ensure the existence of a function symbol of arity $n > 1$ we can use the formula $\exists\, s\, \exists\, t\, (s \mathrel{\rightarrow\!\!\!\!\!\!\!\parallel\;\;} t \wedge \neg(s \to t) \wedge \neg(s = t))$, which results in the single rule $\mathsf{a} \to \mathsf{b}$ over a signature containing an additional binary function symbol f.

▶ **Example 22.** Finding a locally confluent but not confluent TRS $\mathcal{R}$ is easy. FORT produces the two-rule TRS

$$\mathsf{g}(\mathsf{g}(\mathsf{c})) \to \mathsf{c} \qquad\qquad \mathsf{g}(\mathsf{g}(\mathsf{c})) \to \mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{c})))$$

when giving the formula `(WCR & ~CR)`. The well-known abstract counterexample by Kleene

$$\begin{tikzcd}
\mathsf{a} & \mathsf{b} \arrow[l] \arrow[r, bend left] & \mathsf{c} \arrow[l, bend left] \arrow[r] & \mathsf{d}
\end{tikzcd}$$

is found by restricting the search to ARSs. This can be done either by extending the formula to `(WCR & ~CR & forall t Fin(~=,t))` or by using the option `(DEPTH 0)`. Moreover, the default value for the maximal number of rewrite rules has to be increased to at least four `(RULES 4)`. If we impose the condition that $\mathcal{R}^{-1}$ is terminating (cf. [11]), the TRS

$$\mathsf{a} \to \mathsf{b} \qquad\qquad \mathsf{a} \to \mathsf{g}(\mathsf{a}) \qquad\qquad \mathsf{b} \to \mathsf{g}(\mathsf{g}(\mathsf{b}))$$

is produced with `(WCR & ~CR & forall t Fin(<-,t) & ~exists u (u +<- u))`.

8 Conclusion

Concerning related work, we are not aware of any other tree-automata based tool for synthesizing TRSs nor of any tool that allows properties to be specified by an arbitrary first-order formula in the theory of rewriting. Jiresch [6] developed a synthesis tool to attack the well-known open problems (RTA LOOP #13) concerning the sufficiency of certain restricted joinability conditions on critical pairs of left-linear TRSs.

Zantema [11] developed the tool Carpa+ for synthesizing TRSs that satisfy properties which can be encoded as SMT problems. The TRSs that can be synthesized form a small extension of the class of ARSs: A single unary function symbol f is permitted and rules must have the form $a \to b$, $a \to f(b)$, or $f(a) \to b$, where a and b are constants. The properties are restricted to those that can be encoded into the conjunctive fragment of SMT-LRA (linear real arithmetic). The predecessor tool Carpa synthesized combinations of ARSs with help of a SAT solver. It was used to show the necessity of certain conditions in abstract confluence results [8, Section 5].

We conclude by mentioning some ideas for future work. The most interesting extension is the generation of witnesses for existential formulas or formulas with free variables. The efficiency of FORT can certainly be improved, e.g. by converting its sequential code into multi-threaded code. Optimizing the synthesis algorithm for ARSs is easily possible. Support for combinations of TRSs (e.g., to express commutation) is also useful. The enumeration algorithm can be improved, for instance by having the values that restrict the search space increase over time. The question whether rewrite strategies can be incorporated is less easy to answer. A major obstacle is that the subterm relation is not expressible in the first-order theory of rewriting (cf. [3, Exercise 3.13]). Going beyond left-linear right-ground TRSs is a non-trivial matter. Dropping either restriction, one quickly faces an undecidable first-order theory, even when one-step rewriting ($\to$) is the only predicate symbol [9, 7, 10]. Formalizing the underlying theory into an interactive theorem prover like Isabelle/HOL will be a major undertaking, but a necessary step to ensure that the answers and TRSs produced by FORT are correct.

Acknowledgements. We thank Bertram Felgenhauer for reporting a mistake in the earlier implementation of projection.

References

1 F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

2 H. Comon. Sequentiality, monadic second-order logic and tree automata. *I&C*, 157(1-2):25–51, 2000. doi:10.1006/inco.1999.2838.

3 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 2007. URL: http://www.grappa.univ-lille3.fr/tata.

4 M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc. 5th LICS*, pages 242–248, 1990. doi:10.1109/LICS.1990.113750.

5 M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable (extended version). Technical Report I.T. 197, LIFL, 1990.

6 E. Jiresch. A term rewriting laboratory with systematic and random generation and heuristic test facilities. Master's thesis, Vienna University of Technology, 2008. URL: http://www.logic.at/staff/jiresch/thesis/thesis.pdf.

7 J. Marcinkowski. Undecidability of the first order theory of one-step right ground rewriting. In *Proc. 8th RTA*, volume 1232 of *LNCS*, pages 241–253, 1997. doi:10.1007/3-540-62950-5_75.

8 A. Stump, H. Zantema, G. Kimmell, and R. El Haj Omar. A rewriting view of simple typing. *LMCS*, 9(1), 2012. doi:10.2168/LMCS-9(1:4)2013.

9 R. Treinen. The first-order theory of linear one-step rewriting is undecidable. *TCS*, 208(1-2):179–190, 1998. doi:10.1016/S0304-3975(98)00083-8.

10 S. Vorobyov. The undecidability of the first-order theories of one step rewriting in linear canonical systems. *I&C*, 175(2):182–213, 2002. doi:10.1006/inco.2002.3151.

11 H. Zantema. Automatically finding non-confluent examples in term rewriting. In *Proc. 2nd IWC*, pages 11–15, 2013. URL: http://cl-informatik.uibk.ac.at/iwc/iwc2013.pdf.

A Proof of Lemma 13

Proof. The proof is based on the sketch in Dauchet and Tison [5]. Let $\mathcal{A} = (\mathcal{F}^{(2)}, Q, Q_f, \Delta)$ be the tree automaton that accepts the binary regular relation R. The set Q_∞ is defined as follows:

$$Q_\infty = \{q \in Q \mid \langle \bot, t \rangle \to_{\mathcal{A}}^* q \text{ for infinitely many terms } t \in \mathcal{T}(\mathcal{F})\}$$

The set Q_∞ is computed by constructing a graph with states as nodes. For every transition rule $\bot f(q_1, \dots, q_n) \to q$ in Δ we add transitions from $q_1, \dots, q_n$ to q. Now, $q \in Q_\infty$ if there exists a cycle from which q can be reached.

Next we define an automaton $\mathcal{A}' = (\mathcal{F}, Q \cup \bar{Q}, \bar{Q}_f, \Delta \cup \Delta')$. Here, $\bar{Q}$ is a copy of Q where every state is dashed: $q \in \bar{Q}$ if and only if $q \in Q$. For every transition rule $fg(q_1, \dots, q_n) \to q \in \Delta$ we have the following rules in Δ':

$$fg(q_1, \dots, q_n) \to \bar{q} \qquad \text{if } q_i \in Q_\infty \text{ for some } i > \text{arity}(f) \tag{1}$$

$$fg(q_1, \dots, \bar{q}_i, \dots, q_n) \to \bar{q} \qquad \text{for all } 1 \leqslant i \leqslant n \tag{2}$$

Finally we define the automaton $\mathcal{B} = (\mathcal{F}, Q_\mathcal{B}, Q_{\mathcal{B}f}, \Delta_B)$ as the complement of the second projection $\Pi_2(R(\mathcal{A}'))$ of $\mathcal{A}'$. Since the construction for projection does not change (final) states, we have $Q_\mathcal{B} = Q \cup \bar{Q}$ and $Q_{\mathcal{B}f} = Q_\mathcal{B} \setminus \bar{Q}_f$. We show $L(\mathcal{B}) = \{t \in \mathcal{T}(\mathcal{F}) \mid \mathsf{Fin}_R(t)\}$.

$\subseteq$ Let $t = f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F})$ and assume that $\mathsf{Fin}_R(t)$ does not hold, i.e., the set $\mathcal{U} = \{u \in \mathcal{T}(\mathcal{F}) \mid (t, u) \in R\}$ is infinite. Since the signature $\mathcal{F}$ is finite, infinitely many terms u in $\mathcal{U}$ have a depth greater than t. Hence there exists a position $p \notin \mathcal{P}\mathsf{os}(t)$ such that the set $\mathcal{U}' = \{u \in \mathcal{U} \mid p \in \mathcal{P}\mathsf{os}(u)\}$ is infinite. For every $u \in \mathcal{U}'$ we have $\langle t, u\rangle|_p = \langle \bot, u|_p\rangle$. Since $\langle t, u\rangle$ is accepted by $\mathcal{A}$ and Q is finite, there must exist a state q' such that $\langle \bot, u|_p\rangle \rightarrow^*_{\mathcal{A}} q'$ for infinitely many terms $u \in \mathcal{U}'$. Therefore $q' \in Q_\infty$. By construction of $\mathcal{A}'$, there must be a transition rule $f'g(q_1, \ldots, q_m) \rightarrow \bar{q}$ in Δ' of type (1) such that $q_i = q'$ for some $i > \mathsf{arity}(f')$. Pick any $u \in \mathcal{U}'$ such that $\langle \bot, u|_p\rangle \rightarrow^*_{\mathcal{A}} q'$. We distinguish two cases, depending on the position p.

- If $u|_p$ is a direct subterm of u then $f = f'$, $i = p$, and $\bar{q} \in \bar{Q}_f$ by construction of $\mathcal{A}'$. Note that $p > \mathsf{arity}(f)$ because $p \notin \mathcal{P}\mathsf{os}(t)$. Since $(t, u) \in R(\mathcal{A}')$, $t \in \Pi_2(R(\mathcal{A}'))$ and thus $t \notin L(\mathcal{B})$ according to the definition of $\mathcal{B}$.
- Otherwise, the dash of $\bar{q}$ is propagated upwards using transition rules of type (2), such that we obtain $\langle t, u\rangle \rightarrow^*_{\mathcal{A}'} \bar{q}_f$ for some $\bar{q}_f \in \bar{Q}_f$. Hence $(t, u) \in R(\mathcal{A}')$ and we complete the proof as in the previous case.

$\supseteq$ Let $t = f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F})$ and assume $t \notin L(\mathcal{B})$. By construction of $\mathcal{B}$ there exists a term $u = g(u_1, \ldots, u_m) \in \mathcal{T}(\mathcal{F})$ such that $\langle t, u\rangle \in L(\mathcal{A}')$. Hence there exists a transition rule $fg(q_1, \ldots, q_k) \rightarrow \bar{q}_f$ in Δ' with $k = \max(n, m)$ and $\bar{q}_f \in \bar{Q}_f$. We distinguish two cases, depending on the transition rule.

- Suppose the transition rule is of type (2). Hence there exists a state $q_i \in \bar{Q}$ with $1 \leqslant i \leqslant n$ such that $\langle t_i, u_i\rangle \rightarrow^*_{\mathcal{A}'} q_i$ and there must be a position $p \in \mathcal{P}\mathsf{os}(u_i) \setminus \mathcal{P}\mathsf{os}(t_i)$ such that $\langle \bot, u_i|_p\rangle \rightarrow^*_{\mathcal{A}'} q$ for some state $q \in Q_\infty$. By definition of Q_∞, $\langle \bot, s\rangle \rightarrow^*_{\mathcal{A}'} q$ and thus also $\langle t, u[s]_{ip}\rangle \rightarrow^*_{\mathcal{A}'} \bar{q}_f$ for infinitely many terms $s \in \mathcal{T}(\mathcal{F})$. Hence the set $\{u \mid (t, u) \in R\}$ is infinite and therefore $\mathsf{Fin}_R(t)$ does not hold.
- Suppose the transition rule is of type (1). So $q_i \in Q_\infty$ for some $i > n$ and thus $m > n$. Hence $\langle \bot, u_i\rangle \rightarrow^*_{\mathcal{A}'} q_i$ and there exist infinitely many other terms s such that $\langle \bot, s\rangle \rightarrow^*_{\mathcal{A}'} q_i$ and $\langle t, u[s]_i\rangle \rightarrow^*_{\mathcal{A}'} \bar{q}_f$. Hence the set $\{u \mid (t, u) \in R\}$ is infinite and $\mathsf{Fin}_R(t)$ does not hold as before. ◀